caB2B v1.0

caBench-to-Bedside

Technical Guide

Document Change History

Version Number	Date	Contributor	Description
Initial draft	6 March 6 07	Chandrakant Talele Srinath K	First draft
1.0-beta	29 July 07	Rahul Ner Chandrakant Talele Srinath K	Updated for beta release
V1.0	12 Sept 07	Rahul Ner Chandrakant Talele	Updated for V1.0

Updated November 18, 2007

Development Team		
Development	***Programmer's guide***	***Program Management***
Chandrakant Talele	Chandrakant Talele	Dr. Rakesh Nagarajan
Srinath K	Srinath K	Srikanth Adiga
Rahul Ner	Rahul Ner	
Deepak Shingan		
Chetan Patil		
Hrishikesh Rajpathak		

Contacts and Support	
Training contact	cab2b@mga.wustl.edu
Customer support contact	cab2b@mga.wustl.edu

Table of Contents

Chapter 1 Introduction to caB2B Technical Guide

Chapter Organization and Content

This chapter explains the overall structure of the technical guide of caBench-To-Bedside (caB2B). Topics in this chapter include:

- Overview of the Guide
- Organization of the Guide
- Document text convention

Overview of the Guide

The technical guide for caBench-To-Bedside explains key functionalities provided, overall architecture and administrative activities involved with the caB2B application. As shown in table 1 (below), chapter 1 explains the organization of the manual, chapter 2 provides an overview of caB2B requirements, chapter 3 talks about the high level design and components of the application and chapter 4 explains the administrative functions of application client. Installation / deployment of caB2B is covered under a separate guide ***caB2B Installation Guide*** present at http://gforge.nci.nih.gov/frs/?group_id=172

Organization of the Technical Guide

Chapter	*Chapter Contents*
Chapter 1 Introduction to guide	Overall structure of the manual
Chapter 2 Overview of the Software	Overview of caBench-To-Bedside
Chapter 3 Design and Architecture	Overall design and architecture of caB2B
Chapter 4 Administration	Details of administrative activities

Table 1 Organization of technical guide

Document Text Conventions

Following table shows various typefaces to differentiate between regular text and replaceable items, property keys etc. This illustrates how conventions are represented in this guide.

Convention	*Description*	*Example*
Arial boldface type	Represents a key of a property	All applications names should be

Convention	Description	Example
	from a property file	written against **all.applications**
Courier New italics type	Represents name of a property file	*cab2b.properties*
Note:	Highlights a concept of particular interest	**Note:** This concept is used throughout the installation manual.
COURIER NEW CAPITAL CASE	Represents a database table / column or a SQL	PATH table stores all paths
Courier New boldface type	A command to execute	Command **run** will launch client
SELECT M.IDENTIFIER	Represents a SQL query	SELECT M.IDENTIFIER FROM PATH
<>	Indicate parameters whose value should be provided.	Replace **<user_home>** with its proper value

Table 2 Document Conventions

Chapter 2 Overview of the Software

The NCI caBIG™ project creates a common, extensible informatics platform that integrates diverse data types and supports interoperable analytic tools. caBIG™ is developing separate applications that will facilitate individual steps involved in micro-array analysis. These applications are also useful to bio-informaticians.

caGrid is the infrastructure for caBIG that helps integrate these applications. caGrid can be used to perform investigations involving the integration of data and analytical services from diverse research communities.

caBench-to-Bedside (caB2B) is a caGrid client that permits bench scientists, translational researchers, and clinicians to leverage caBIG™ compatible data and analytical services through a graphical user interface. Its metadata-based query interface enables end users to search virtually any caGrid data service.

Software Overview

Key software functions

caB2B can be used by the physician scientist to perform operations such as the following:

- **Query any caGrid data service to obtain data**.
  The data service can be a single data service, multiple data services, or a combination of the two services using semantically interoperable Common Data Elements (CDEs).

  For example, caB2B allows investigators to query tissue banks at multiple cancer centers, design studies that focus on very specific tumor subtypes and to target less common tumors by pooling bio-specimen resources. This alleviates the problem of small sample size in a study

- **Collect data and create experiments.**
  caB2B also enables investigators to perform novel in silico experiments using micro-array data sets that have already been collected. These capabilities facilitate the process of identifying genes that are up-regulated or down-regulated in specific cancers and also enable investigators to view data in the context of biological pathways. Investigators also gain further understanding of the complex system of cancer biology by identifying genes important to the development and treatment of cancer. This may lead to more effective identification of novel drug targets, and improved treatment strategies.

- **Visualize analysis results** by using various viewers such as charts.

Please visit caB2B website at http://cab2b.wustl.edu/ for latest updates, downloads and

documents.

Security

caB2B v1.0 does not support any type of user login, credential and security features. These features will be available in subsequent releases.

Chapter 3 caB2B Design and Architecture

This section describes the operations a user can perform using caB2B along with the overall architecture and high level design of the caB2B. This chapter covers following

- Overall Architecture
- Client side components
- Server side components
- Metadata repository

For more details, refer to cab2b design document at https://gforge.nci.nih.gov/frs/?group_id=172

Architecture

This section describes the overall architecture and high level design of the caB2B.
The caB2B application is a highly user interaction-rich application that will allow the user to perform the following:

- Search and query different grid enabled data services to acquire data sets of interest
- Save data sets and create an 'experiment' to analyze and visualize information
- Perform different analyses using different grid enabled analytical services
- Visualize analysis results using a rich collection of windows
- Execute workflow jobs, time-consuming queries, or analyses asynchronously
- Share experimental results amongst multiple caB2B users

The caB2B application has a client-server based architecture.

The caB2B client is a desktop application (implemented in Java Swing) which provides the user a graphical user interface to search for data sets of interest, create experiments, and view different analysis results.

The caB2B server performs backend activities associated with user interactions. The server caches static data like classes and attributes from domain models and their associations as well as query execution results. Figure 1 shows the overall architecture of caB2B.

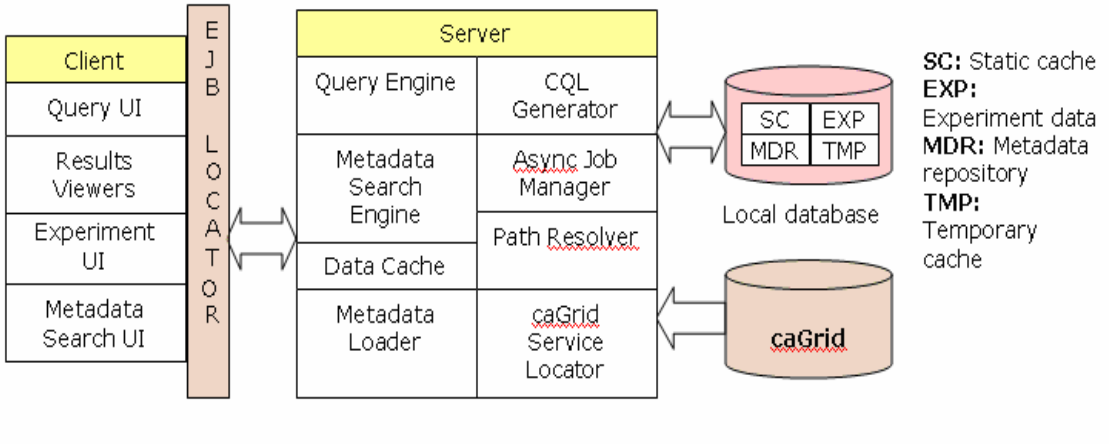

Figure 1 caB2B Client-Server Architecture

Client side components

Metadata Search UI Allows the user to search the application metadata for data categories of interest. The UI provides advanced search options like searches based on concept codes, permissible values etc.

Query UI is a user interface to find data for creation of virtual experiments. It provides the ability to build complex queries using a diagrammatic representation of the query.

Results Viewers are the viewers used to view data present in an experiment and results of an analysis performed.

Experiment UI is used to arrange experiments in the projects, create subsets of data, invoke analytical services and apply various types of filtering.

Server side components

Metadata Loader is a module which loads domain models (UML model) from caDSR to local database.

Path resolver Helps user in specifying appropriate paths between categories in a query.

CQL generator Builds DCQL based on user input.

Query Engine executes a DCQL

Data cache Caching mechanism used to cache frequently used data objects

Metadata repository is explained in the section below.

Metadata repository (MDR)

One of the basic requirements of caB2B is to be able to download a UML model of any application from the caDSR and provide capabilities to build a query to fetch data from that data source. To do this, it is necessary to first understand the concept of the metadata repository (MDR). MDR stores the metadata for a UML model with associated semantic annotations like CDEs including permissible values. It gets this metadata by decomposing the annotated UML model obtained from caDSR. caB2B uses Dynamic Extensions framework to store the UML model along with its semantic annotations.

Dynamic extensions

Dynamic Extensions is a framework that allows the dynamic creation of business objects in the form of entities and attributes. Following are the Dynamic Extensions (DE) terms found in this document:

- Entity is a UML class.
- Attribute is a UML attribute.
- Association is a relationship between any two entities.

The metadata definition of entity and attribute includes:

- Model Properties (i.e. Data type, Precision etc.)
- Semantic properties (i.e. concept codes)
- Value domain specification (CDE public id, permissible values etc.)

Path Storage

The caB2B server pre-calculates the paths between all pairs of classes in the UML model and stores them in the MDR. Classes from different applications are connected based on their attribute's CDE match. This involves matching the concept codes of the classes and their attributes in order. Finally, given the amount of information it stores, it is also possible to get all the paths between two classes across two different UML models based on semantic interoperability.

An association denotes a possible traversal from one entity (the source entity) to another (the target entity). Each association is stored in the table **`ASSOCIATION`**. There are two types of associations

- Intramodel associations link two entities in the same application and are stored in the table **`INTRA_MODEL_ASSOCIATION`**. This table contains a single column **`DE_ASSOCIATION_ID`** that indicates which DE association corresponds to this intramodel association. This table can be thought of as a mapping from **`ASSOCIATION`** to DE's **`DYEXTN_ASSOCIATION`**.
- Intermodel associations link two entities from different applications based on the semantic links between CDEs contained by the two entities. These are stored in the table **`INTER_MODEL_ASSOCIATION`**.

A path is a list of associations or more specifically, a list of **`IASSOCIATION.ID`** ‘s . Such an association can be an intermodel association; thus we can have paths that traverse several entities across multiple models.

To find a path, first determine all paths between the source and target entities. Then use the path that consists of the desired associations; the associations in the path are a '_' delimited string of **`IASSOCIATION.ID`** ‘s.

What is a Category?

Category is a collection of attributes from one or more UML classes. These UML classes may be from the same or different applications. UML classes in a category should be directly or indirectly connected using UML associations.

To illustrate of the usage of category, consider the following use case: Get all genes with annotation which are associated with a given "Gene" through pubMed literature abstract i.e. get list of genes having literature relationship correlation value > 0.5 and have relationship with given gene. The UML diagram for the classes in the query is

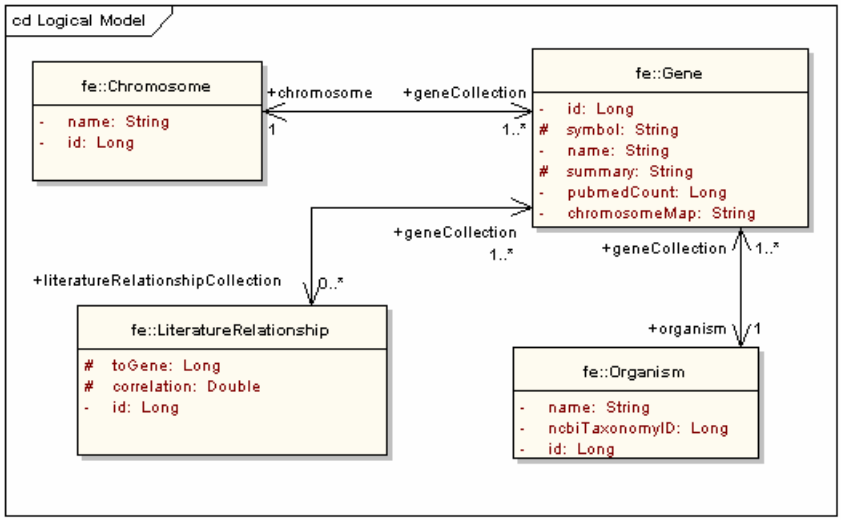

Figure 2 diagram for classes in category

To build the example query, user would

1. Search the four classes individually
2. Add limits on each of them
3. Connect all the classes in the DAG view

Limitations of the above process:

- UML Class is a collection of attributes that only developers and bioinformaticians are familiar with.
- The steps described above are cumbersome and time-consuming.
- Each user who wishes to perform this query has to follow this process every time

In certain use cases, each user will define limits on specific attributes of certain logically related classes and connect them by similar paths. In such cases, those attributes can be grouped together to build predefined units with unambiguous paths to save users' time. These predefined units are categories.

Benefits

- Ability to apply limits on attributes of several UML classes in one go
- Paths among classes in a category will be predefined in metadata. Thus, the user need not find paths required to traverse logically related classes every time.
- End-user sees attributes in a single logical unit even though they belong to different classes due to modeling constraints
- Users with limited knowledge of UML domain models can query on categories.
- Advanced users can also use categories as building blocks for their complex queries

Chapter 4 Administration

This chapter focuses on the administrative part of caB2B. It covers the following topics:

- Providing memory size
- Configuring the caB2B server for connection
- Configuring data service instances
- Curating Paths
- Creating Category

Out of these tasks first three are client side tasks whereas last two are server side tasks. To perform any server side tasks, you need to make sure that the cab2b-server is not running.

Configuration file: cab2b.properties

For following administrative tasks you need to understand *cab2b.properties* file.

- Providing the memory size
- Configuring caB2B server to connect to
- Configuring data service instances

To modify client side configurations, create *cab2b.properties* in **<user_home>\conf.** **<user_home>** is a directory referred by **user.home** system property in java. For windows, it is <windows_installation_drive>\Documents and Settings\<user_name>**.**

Following is a typical *cab2b.properties* file :

```
#############################################################################
# All applications which are to be queried from caB2B
# NOTE: Metadata of all of the above mentioned application MUST be present
# in caB2B server you are connecting to.
#############################################################################
all.applications=geneConnect,caArray

# URLs of dataservices
geneConnect.ServiceURL=http://128.252.227.94:9092/wsrf/services/cagrid/GeneConnect
caArray.ServiceURL=http://caarraydb.nci.nih.gov/wsrf/services/caGrid/CaArraySvc

#############################################################################
#               caB2B server details
#   IP and the port where JNDI service is running
#############################################################################
caB2B.server.ip=cab2b.wustl.edu
caB2B.server.port=1099
```

Providing the memory size

If you are launching the caB2B client using cab2bClientSetup.zip then you can override the default memory usage. It can be done by changing value against **-Xmx** attribute in **`run.bat`** or **`run.sh`**. By default it is 512MB.

Note: This setting is unavailable if you are launching the client using java-webstart.

Configuring caB2B server to connect

By default each caB2B client connects to caB2B server running at Washington University in St. Louis. Each caB2B client can be configured to connect to a specific caB2B server. This can be done by modifying *`cab2b.properties`*. Refer to section cab2b.properties file to understand configurations mentioned in this section.

You need to put the server name or IP against **caB2B.server.ip** and the port on which JNDI service is running on the caB2B server against **caB2B.server.port** as follows

```
caB2B.server.ip=cab2b.wustl.edu
caB2B.server.port=1099
```

Configuring data service instances

Refer to section cab2b.properties file to understand configurations mentioned in this section.

In the *`cab2b.properties`* file value, of **all.applications** should be names of all the applications in comma separated form. This specifies application you wish to query using cab2b.

Note: It is necessary that cab2b server which you are connecting to has metadata about all applications you are specifying in cab2b.properties.

To configure service instance to query, you need to put data service URL against **application_name.ServiceURL** property. You can query multiple instances of the same application by putting data service URL in comma separated form as follows

```
all.applications=caFE
caFE.ServiceURL=http://server1/wsrf/services/cagrid/caFE,http://server2/wsrf/services/cagrid/caFE
```

Curating Paths

Complex models have many classes with associations, and there are multiple possibilities of traversing the UML model to interconnect two classes. Some paths may be illogical, incorrect, or inappropriate in all instances while some may be relevant based on the context of a query. Thus, to facilitate end user query formation, caB2B allows for the curation of paths between any two classes based on the available paths between those two classes given a particular UML model. This section describes how to curate such paths.

A "path" connects two entities and is represented as a list of associations. Paths are stored in the **PATH** table. The caB2B administrator can define one or more paths that should be used by default to connect a set of entities. Such a set of paths is called a "curated path". Curated paths are stored in table **CURATED_PATH**. It contains the entities that each curated path connects. Since a path can be a part of several curated paths, and, by definition, a curated path is a set of paths, the relationship between **PATH** and **CURATED_PATH** is many-many. The mapping table for this many-many relationship is **CURATED_PATH_TO_PATH**. Currently curated paths are defined by directly inserting appropriate rows in the tables **CURATED_PATH** and **CURATED_PATH_TO_PATH**. For example, to define a curated path {*Gene->Protein, Gene->mRNA*} to connect the three entities *Gene, Protein* and *mRNA* in the GeneConnect model, the administrator would:

1. Obtain entity ids of these entities. This can be done using following SQL for *Gene*:
   ```
   SELECT M.IDENTIFIER,M.NAME FROM DYEXTN_ABSTRACT_METADATA M
   JOIN DYEXTN_ENTITY E ON M.IDENTIFIER = E.IDENTIFIER WHERE NAME LIKE '%gene%'
   ```
   For example, suppose the entity ids for *Gene*, *Protein*, and *mRNA* are 1, 2, and 3 respectively.

2. Find path ids of the two paths *Gene->Protein*, *Gene->mRNA*. This can be done using following SQL for *Gene->Protein*:
   ```
   SELECT PATH_ID,INTERMEDIATE_PATH FROM PATH WHERE FIRST_ENTITY_ID=1 AND
   LAST_ENTITY_ID=2
   ```
   Suppose the path ids are 5 and 6, respectively, for *Gene->Protein*, *Gene->mRNA*.

3. Get the next available id to insert into `CURATED_PATH` using following SQL
   ```
   SELECT MAX(CURATED_PATH_ID) + 1 FROM  CURATED_PATH
   ```
   Suppose this returns you 10. This is the **CURATED_PATH_ID** for this curated path.

4. Insert a row in **CURATED_PATH** as follows:
   ```
   INSERT INTO CURATED_PATH (CURATED_PATH_ID, ENTITY_IDS, SELECTED)
   VALUES (10, '1_2_3',1)
   ```

   ENTITY_IDS is a '_' delimited string of sorted entity ids; selected is boolean indicating if this curated path is to be applied by default in an auto-connect.

5. Insert rows in **CURATED_PATH_TO_PATH** as follows:
   ```
   INSERT INTO CURATED_PATH_TO_PATH (CURATED_PATH_ID,PATH_ID) VALUES (10,5);
   INSERT INTO CURATED_PATH_TO_PATH (CURATED_PATH_ID,PATH_ID) VALUES (10,6);
   ```

Creating a Category

A category is defined as a collection of attributes from one or more classes within or across models that make intuitive sense to end users. Note that as a result, the path to connect classes in a category must also be specified. Attributes and the path are specified in a well-formed XML file called the **category XML**. All categories are first defined as a Category XML, and are then imported into the metadata repository. Classes present in a category must form a tree where each class is a node and each link is a path from parent node to child node. The structure of this file is as follows:

```
<?xml version="1.0" encoding="windows-1250"?>
<Category>
    <CategorialClass name="" IdOfPathFromParentToThis="-1"> <!-- root -->
        <Attribute name = />
        <Attribute name = />
        .
        .
        .
        <CategorialClass name="" IdOfPathFromParentToThis = "23451">
          <Attribute name = />
          <Attribute name = />
          .
          .
          <CategorialClass name="" IdOfPathFromParentToThis = "53451">
            .
               .
                  .
          </CategorialClass>
        </CategorialClass>
        .
        .
    </CategorialClass>
    <!--  here are SubCategories  -->
    <Category>
    .
    .
    </Category>
</Category>
```

Figure 3 Category XML structure

Each class in the tree corresponds to a CategorialClass tag. The CategorialClass tag under the Category tag corresponds to the root class. For each CategorialClass, the administrator needs to specify the path to be taken from the enclosing CategorialClass (i.e. the parent node) to this node. The administrator may acquire these path ids in the following way:

1. Obtain entity ids of these entities. This can be done using following SQL for the each class, The example for the *Gene* class from the GeneConnect model is as follows:
   ```
   SELECT M.IDENTIFIER,M.NAME FROM DYEXTN_ABSTRACT_METADATA M JOIN DYEXTN_ENTITY E
   ON M.IDENTIFIER = E.IDENTIFIER WHERE NAME LIKE '%gene%'
   ```
   For example, suppose the entity id for *Gene* and *Protein* is 1 and 2 respectively.

2. Find path ids of the two paths *Gene->Protein*. This can be done using following SQL for *Gene->Protein*:
   ```
   SELECT PATH_ID,INTERMEDIATE_PATH FROM PATH WHERE FIRST_ENTITY_ID=1 AND
   LAST_ENTITY_ID=2
   ```

Name of CategorialClass is fully qualified name of the class. Attribute name is name of the attribute in the enclosing class. An optional display name for each attribute may also be provided. The following is an example of the **Category XML** file for the category “Genomic identifiers”

```
<?xml version="1.0" encoding="windows-1250"?>
<Category name="Genomic Identifiers">
  <CategorialClass name="edu.wustl.geneconnect.domain.Gene" IdOfPathFromParentToThis="-1"> <!-- root -->
    <Attribute name = "ensemblGeneId" displayName = "ensemblgeneID"/>
    <Attribute name = "unigeneClusterId" displayName = "uniGeneClusterId"/>
    <Attribute name = "entrezGeneId" displayName = "entrezGeneId"/>
    <CategorialClass name="edu.wustl.geneconnect.domain.MessengerRNA" IdOfPathFromParentToThis = "716">
      <Attribute name = "ensemblTranscriptId" displayName = "mRNAEnsemblTranscriptId"/>
      <Attribute name = "genbankAccession" displayName = "mRNAgenBankAccessionNumber"/>
      <Attribute name = "refseqId" displayName = "mRNArefSeqId"/>
      <CategorialClass name="edu.wustl.geneconnect.domain.Protein" IdOfPathFromParentToThis = "616">
            <!-- Path -> Gene-mRNA-protein -->
            <Attribute name = "ensemblPeptideId" displayName = "ensemblPeptideId"/>
            <Attribute name = "refseqId" displayName = "proteinRefSeqId"/>
            <Attribute name = "uniprotkbPrimaryAccession" displayName = "proteinUniProtKBPrimaryAccession"/>
            <Attribute name = "genbankAccession" displayName = "proteinGenBankAccession"/>
      </CategorialClass>
    </CategorialClass>
  </CategorialClass>
</Category>
```

Figure 4 Example of Category XML file

Once a category XML file is created, it needs to be loaded into the caB2B server’s database. caB2B provides classes to perform this function. Namely, CategoryXmlParser parses this file and generates an InputCategory object. PersistCategory converts InputCategory to a Category hibernate-object which will be saved by CategoryOperations**.** This flow in explained in sequence diagram below

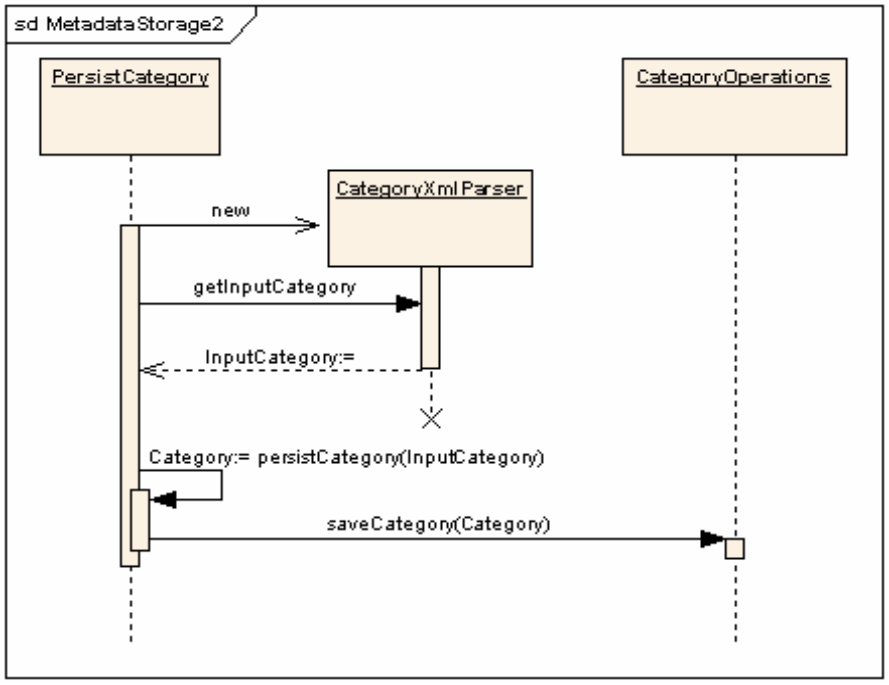

Figure 5 Sequence diagram saving a category

Thus, to load a new category, the following code may be inserted into the main method of a class (say *CategoryLoader*):

```
CategoryXmlParser parser = new CategoryXmlParser();
InputCategory inputCategory = parser.getInputCategory(filePath);
PersistCategory persistCategory = new PersistCategory();
Category category = persistCategory.persistCategory(inputCategory,null);
CategoryOperations categoryOperations = new CategoryOperations();
categoryOperations.saveCategory(category);
```

Note that this class must be defined in the package *edu.wustl.cab2b.server.category*. The administrator may then load a category by running the program *CategoryLoader* with class path `server.classpath. server.classpath` is defined in **<home_dir>**/build/build.xml where **<home_dir>** is the location of the caB2B server deployment.

References

1. caB2B web site
   http://cab2b.wustl.edu/

2. caB2B design document:
   http://gforge.nci.nih.gov/frs/download.php/2349/cab2b_design_document.pdf

3. caB2B installation guide:
   http://gforge.nci.nih.gov/frs/download.php/2353/caB2B_Installation_Guide.pdf

4. caB2B 1.0 release
   http://gforge.nci.nih.gov/frs/shownotes.php?release_id=1542

Glossary

Term	Definition
`user.home`	The home directory for the logged-in user
caBIG	cancer Biomedical Informatics Grid https://cabig.nci.nih.gov/
caGrid	The underlying service oriented infrastructure that supports caBIG™ is referred to as caGrid. https://cabig.nci.nih.gov/workspaces/Architecture/caGrid
caDSR	Cancer Data Standards Repository
DCQL	Distributed caGrid Query Language, also called as federated query http://www.cagrid.org/mwiki/index.php?title=Federated_Query:DCQL
Hibernate	Hibernate is a powerful, high performance object/relational persistence and query service which lets you develop persistent classes following object-oriented idiom - including association, inheritance, polymorphism, composition, and collections http://www.hibernate.org/
java-webstart	Using Java Web Start technology, standalone Java software applications can be deployed with a single click over the network. For more details refer http://java.sun.com/products/javawebstart/
metadata	Definitional data that provides information about or documentation of other data.
NCI	National Cancer Institute
NCICB	National Cancer Institute Center for Bioinformatics
UML	Unified Modeling 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

Appendix A : Dynamic Extension and MDR

Overview

One of the most important components of the DE project is its metadata repository. MDR can contain metadata about dynamic extensions or static UML models. Each DE is also a UML model. The MDR is very important component not just for DE, but also for applications like caB2B and caTissue Suite. The basic backbone of MDR is as shown in Figure 1.

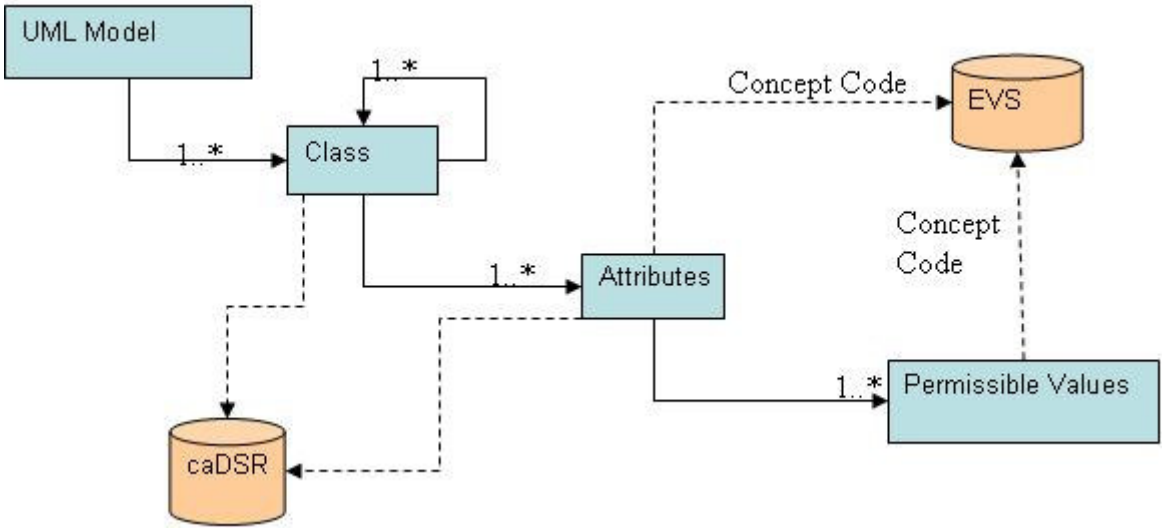

Figure 6 Metadata Repository backbone

MDR contains the following metadata for a domain model:

- Classes
- Attributes
- Data type
- Concept codes
- Description
- Permissible values
-

Ifthe domain model is created using the dynamic extensions user interface, the MDR will contain the UI display properties and the database mapping information for each attribute. The metadata for the user interface contains:

- Type of UI Control
- Properties like height, width, password like string and so forth
- Mandatory or optional attribute

Table to which the entity maps and column to which the attribute maps

UML Metadata

This contains all the information present in the UML model like class, attributes, and associations including the permissible values. Following diagram shows the classes involved in entity creation along with the relationships involved in these classes.

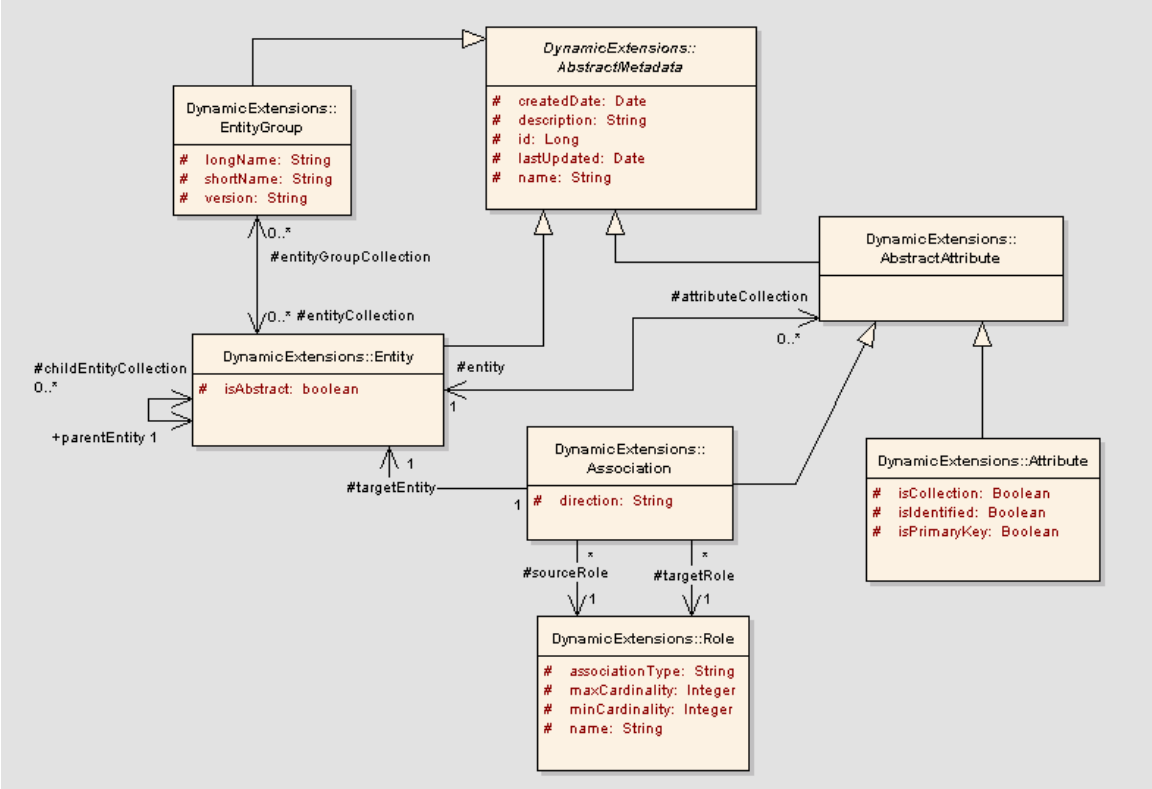

Figure 7 Dynamic extension basic metadata

AbstractMetadata: This is an abstract base class from which the backbone metadata objects are derived. This class contains generic attributes which are part of all objects (like create date, last updated and so forth).

EntityGroup: An entity group is a logical collection of entities. For example, all classes of an application are loaded under one entity group. It contains multiple entities.

Entity: This class represents an UML class. An entity is associated to itself to specify its parent entity. An entity can have zero or one parent entity. An entity can also have zero or more children entities.

AbstractAttribute: An entity can either have zero or more primitive attributes, or have zero or more associated classes. This is represented by the AbstractAttribute class. It is the base class for Association and Attribute classes.

Attribute: The class represents a primitive attribute. For example, name is an attribute of the user entity. Attribute can be of following types:

- String attribute
- Double attribute
- Short attribute
- Long attribute
- Boolean Attribute
- Date attribute
- ByteArray (for BLOB/CLOB)

Following diagram shows how attribute type is defined or changed in attribute.

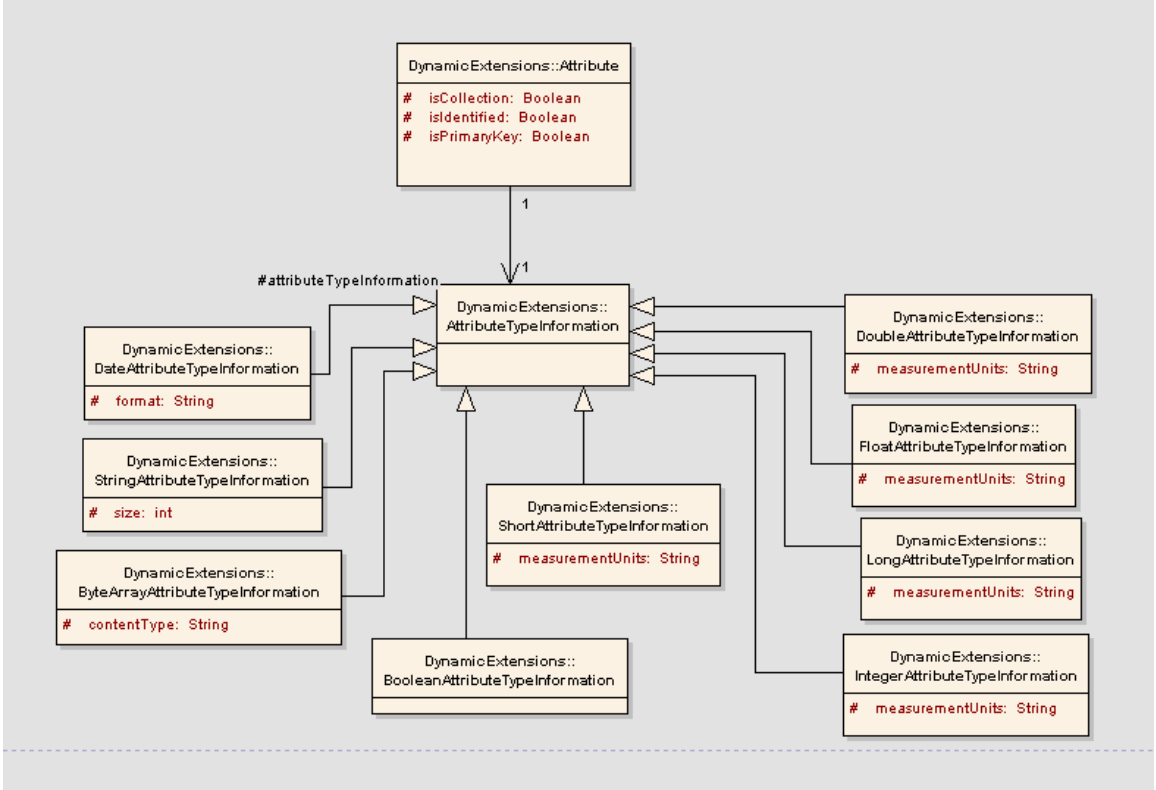

Figure 8 Attribute Type Metadata

Attribute class is associated with the class “AttributeTypeInformation” that specifies the type of the attribute.

AttributeTypeInformation: This class represents the type of the attribute. Attribute type can be any of the above mentioned types. This class is an abstract class which is extended by all the specific primitive attribute types like DoubleAttributeTypeInformation or StringAttributeTypeInformation.

Role: This class describes an association’s cardinality and the association type. The class has the following attributes

associationType: This could be two types of association: containment or linking.

Containment association type is one of Person and Address where the Person entity will contain Address entity within it. The Address object does not exist on its own. Linking

association type is one of User and Study. Here, both the objects can be created independently. The user can be part of multiple studies and a study can contain multiple users.

maxCardinality: Maximum cardinality of association (for example, 1 or many)

minCardinality: Maximum cardinality of association (for example, 0, 1 or many)

name: The role name of the association.

Association: This class represents the associations that an entity can have with other entities. E.g. a User entity is associated with Institute entity.

sourceRole: This represents the role of the association from the source context.

targetRole: This represents the role of the association from the target context.

Inheritance Metadata support

One of the main aspects of any application is the inheritance between its entities. So when any object model is loaded into DE database, this hierarchy of objects should be preserved. This section explains how inheritance is preserved in DE using the required metadata objects of DE. Following diagram explains the required objects and relationships for inheritance.

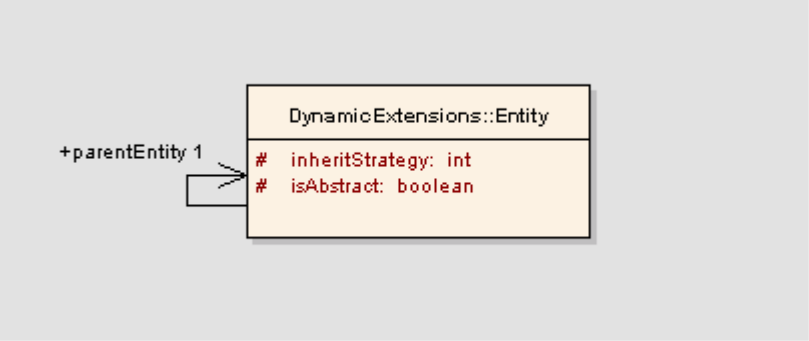

Figure 9 Inheritance Metadata

Entity: Entity object represents the java class in any object model. So to maintain the hierarchy of classes, following attributes and associations are maintained.

isAbstract: This flag maintains whether the entity is abstract or not.

inheritStrategy: This attribute stores the Hibernate's strategy to store the actual data in the actual database. Allowed values for this attribute are:

1. Joined subclass
2. Subclass
3. Table per concrete class.

Attribute data elements and default values

An attribute can have values that are derived from some fixed source or some user defined set of allowable values. For example, gender attribute can have only fixed values like male

and female. Additionally, the attribute can have one of them as a default value. This information is saved in following way. The diagram shows the way in which the allowable and default values are stored in DE

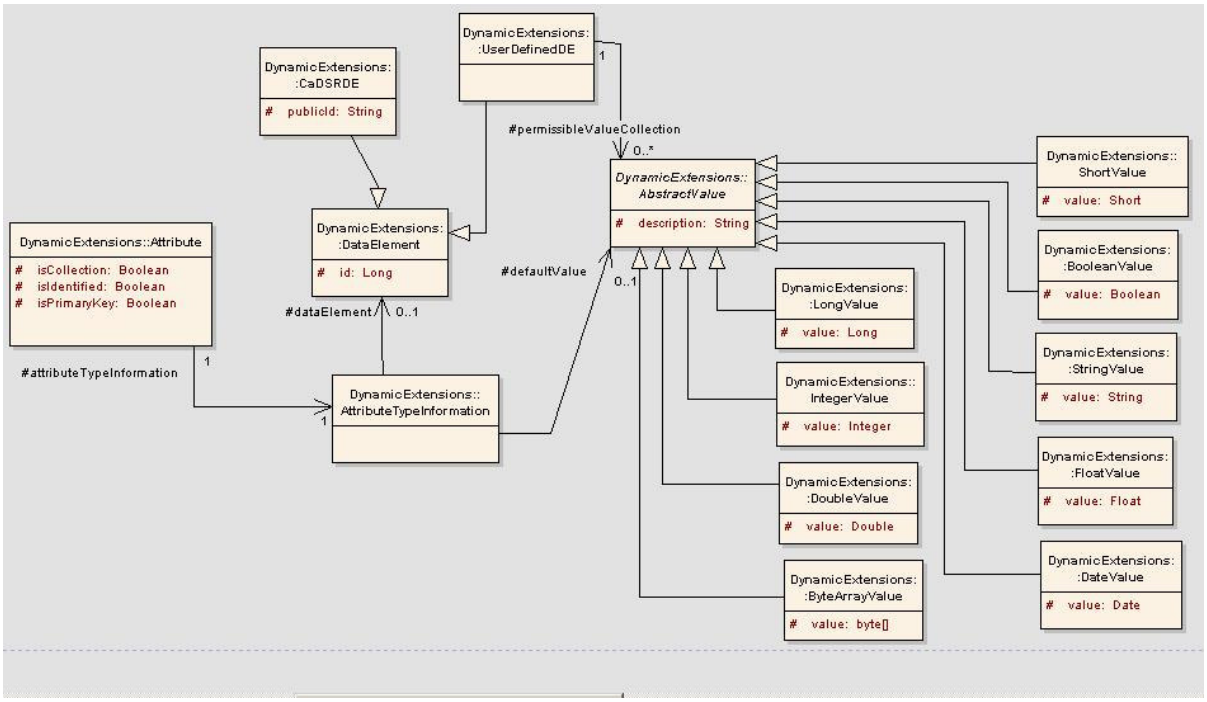

Figure 10 Attribute Data Elements

caDSRDE holds all the common information for all the types of data elements. Some of the associations of this class are:

- *AttributeTypeInformation*: Source of the allowable values is specific to the attribute type. So to represent this information correctly, AttributeTypeInformation class is associated with the DataElement so that it represents the type of source for the attribute.
- *AbstractValue*: This class represents a value, an attribute can have. This value can be used as a default value or as one of the allowable values. The class acts as a base class for the entire attribute type specific value