CAARRAY 1.6

Technical Guide

Center for Bioinformatics

September 6, 2007

Credits and Resources

caArray Development and Management Teams			
Development	***Quality Assurance***	***Documentation***	***Project and Product Management***
Eric Tavela[2]	Bill Mylander[5]	Eric Tavela[2]	Mervi Heiskanen[1]
Leslie Power[2]	Jenny Glenn[5]	Leslie Power[2]	Anand Basu[1c]
Steve Matyas[2]	Xiaopeng Bian[1]	Rob Daly[2]	Jerry Eads[4]
Krishna Kanchinadam[2]		Krishna Kanchinadam[2]	Rob Daly[2]
		Steve Matyas[2]	Juli Klemm[1]
		Jill Hadfield[1]	
Systems and Application Support		***Training***	
Wei Lu[3]		Don Swan[3]	
Vanessa Caldwell[3]			
Michael Gomes[3]			
[1] National Cancer Institute Center for Bioinformatics (NCICB)		[2] 5AM Solutions	[3] Terrapin Systems
[4] Northern Taiga Ventures, Inc.	[5]NARTech		

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

TABLE OF CONTENTS

Credits and Resources i

Using the caArray Technical Guide 1

Introduction to caArray 1
Purpose of this Manual 1
Recommended Reading 2
Organization of the Manual 2
Document Text Conventions 2

Chapter 1
caCORE and caBIG Overviews 5

NCICB caCORE Infrastructure Overview 5
caBIG Compliance 6
caBIG Architectural Principles 7
caBIG-Compliant Data Standards 7
caArray at caBIG Cancer Centers 7

Chapter 2
Overview of caArray 9

Chapter 3
caArray Architecture 11

caArray Architecture Diagram 11
caArray Technologies 12
caArray Interfaces 14

Chapter 4
caArray Design 17

Web Tier 17
Model View Controller 2 17
Composite View 18
Struts 18

Application Tier 19
Data Transfer Objects 19
EJB Layer 20
MAGE Manager 25
MAGE-stk 26
Object Relational Mapping 28
Database Tier 32
caArray Database 32
Security Database 32
File Share 32

Chapter 5
caArray Data Access Security 33
Access Privileges 33
CRUD Permission Levels 35
Security EJB 35
Enabling Security 36
EJBs 36
Programmatic Security 37

Chapter 6
caArray Download Site 39
caArray Portal 39
caArray MAGE-OM API 40

Chapter 7
MAGE-OM API 41
MAGE-OM API Introduction 41
MAGE-OM Directory Structure 42
Accessing the MAGE-OM Production Server 43
Testing the MAGE-OM Production Server 43
MAGE-OM Architecture 44
Domain Objects 44
SearchCriteria 44
RMI 45
MAGE-OM Security 47
Client-side Security 47
Server-side Security 48
Directable 50
Persistence 50

Chapter 8
caArray APIs 53

caArray EJB API 53
Directory Structure 54
Testing caArray EJB APIs 56

Chapter 9
caAMEL Service API 59

Introduction to the caAMEL API 59
caAMEL Service API Code Example 60
Considerations Before Coding for caAMEL Service API 62

Appendix A
UML Modeling 63

UML Modeling 63
Use-case Documents and Diagrams 64
Class Diagrams 66
Relationships Between Classes 67
Sequence Diagrams 69

Appendix B
caArray References 71

Appendix C
caArray Glossary 73

Index 77

USING THE caARRAY TECHNICAL GUIDE

This chapter contains an overview of the technical guide.

Topics in this chapter include:

- *Introduction to caArray* on this page
- *Purpose of this Manual* on this page
- *Recommended Reading* on page 2
- *Organization of the Manual* on page 2
- *Document Text Conventions* on page 2

Introduction to caArray

The National Cancer Institute (NCI) Center for Bioinformatics (NCICB) Cancer Array Informatics Project (caArray) consists of a microarray database and microarray data analysis and visualization tools (http://caArray.nci.nih.gov). caArray is an open source project, and the source code and Application Programming Interfaces (APIs) are available for local installations at http://ncicb.nci.nih.gov/download under an open source license. The goals of the project are to make microarray data publicly available, and to develop and bring together open source tools to analyze these data. See *Overview of caArray* on page 9 for more information on caArray.

Purpose of this Manual

The caArray Technical Guide is intended for developers wanting to understand the underlying design of the caArray database and architecture to help them better utilize the open source code and APIs. This guide does not contain information on the data management or data analysis tools used by caArray.

Existing caArray documentation can be found on the caArray page of the NCICB website: http://caarray.nci.nih.gov/documentation. This guide does not duplicate documents found independently at that website, but contains ancillary technical documentation contributing to the successful utilization of caArray.

Recommended Reading

Appendix B contains reading materials and resources that can be useful for familiarizing oneself with concepts contained within this guide.

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

Organization of the Manual

The *caArray Technical Guide* contains the following chapters:

- *Using the caArray Technical Guide*
- *Chapter 1 caCORE and caBIG Overviews*
- *Chapter 2 Overview of caArray*
- *Chapter 3 caArray Architecture*
- *Chapter 4 caArray Design*
- *Chapter 5 caArray Data Access Security*
- *Chapter 6 caArray Download Site*
- *Chapter 7 MAGE-OM API*
- *Chapter 8 caArray APIs*
- *Chapter 9 caAMEL Service API*
- *Appendix A UML Modeling*
- *Appendix B caArray References*
- *Appendix C caArray Glossary*

Document Text Conventions

Table *1.1* illustrates how text conventions are represented in this guide. The various typefaces differentiate between regular text and menu commands, keyboard keys, toolbar buttons, dialog box options and text that you type.

Convention	***Description***	***Example***
Bold & Capitalized Command Capitalized command > Capitalized command	Indicates a Menu command Indicates Sequential Menu commands	**New Array Design**
TEXT IN SMALL CAPS	Keyboard key that you press	Press ENTER
TEXT IN SMALL CAPS + TEXT IN SMALL CAPS	Keyboard keys that you press simultaneously	Press SHIFT + CTRL and then release both.

Table 1.1 caArray Guide Text Conventions

Convention	***Description***	***Example***
`Monospace type`	Used for filenames, directory names, commands, file listings, and anything that would appear in a Java program, such as methods, variables, and classes.	`ExperimentData`
Boldface type	Options that you select in dialog boxes or drop-down menus. Buttons or icons that you click.	From the Experiment Details page, click **Generate MAGE-ML**.
Italics	Used to reference other documents, sections, figures, and tables.	*caArray User's Guide*
`Boldface monospace type`	Text that you type	In the New Subset text box, enter **`Array Manufacture Software.`**
Note:	Highlights a concept of particular interest	**Note:** This concept is used throughout the installation manual.
Warning!	Highlights information of which you should be particularly aware.	**Warning!** Deleting an object will permanently delete it from the database.
{ }	Curly brackets are used for replaceable items.	Replace `{root directory}` with its proper value, such as `c:\caarray`

Table 1.1 caArray Guide Text Conventions (Continued)

CHAPTER
1

CACORE AND CABIG OVERVIEWS

This chapter contains overviews of the cancer Common Ontologic Representation Environment (caCORE) and the cancer Biomedical Informatics Grid (caBIG) since the caArray application is engineered to use the caBIG APIs, thus complementing the caCORE infrastructure.

Topics in this chapter include:

- *NCICB caCORE Infrastructure Overview* on this page
- *caBIG Compliance* on page 6

NCICB caCORE Infrastructure Overview

Note: caArray will interface to the caCORE infrastructure in future releases; the caArray application will point to Cancer Bioinformatics Infrastructure Objects (caBIO). Currently, Cancer Data Standards Repository (caDSR) data is loaded into the caArray database.

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

Characteristics of caCORE include:

- Model Driven Architecture
- n-tier architecture with open APIs
- Use of controlled vocabularies, wherever possible
- Registration of metadata

The first two items allow for easy access of data, particularly by other applications. The last two items help resolve what might be called the ‘metadata problem’, that is, identifying in an unambiguous manner the meaning of each object and attribute in an API. In this case, metadata is the definition of an attribute rather than its value. For example, given the attribute ‘zipCode’ its value might be ‘20852’ while its metadata (definition) is ‘a 5 or 9 digit number used by the United States Postal Service to divide geographical regions into delivery zones’. By registering metadata (using terms in an electronically accessible controlled vocabulary) in a repository, caCORE provides a means to select appropriate information resources and to aggregate information from multiple sources.

All of the components of caCORE are designed using these same principles. Systems with these properties are said to be “caCORE-like”.The main components of caCORE, created and deployed by NCICB, include:

- **Cancer Bioinformatics Infrastructure Objects (caBIO):** A set of JavaBeans with open APIs that can be used to directly access bioinformatics data. (http://ncicb.nci.nih.gov/NCICB/infrastructure/cacore_overview/caBIO). Unified Modeling Language™ (UML) models of biomedical objects are implemented in Java as middleware connected to various cancer research databases to facilitate data integration and consistent representation.
- **Cancer Data Standards Repository (caDSR):** A metadata registry based upon the ISO/IEC11179 standard that is used to register the descriptive information needed to render cancer research data reusable and interoperable (http://ncicb.nci.nih.gov/NCICB/infrastructure/cacore_overview/cadsr). The caBIO, EVS and caDSR data classes are registered in the caDSR, as are the data elements on NCI-sponsored clinical trials case report forms.
- **Enterprise Vocabulary Services (EVS):** Controlled vocabulary resources that support the life sciences domain, implemented in a description logics framework (http://ncicb.nci.nih.gov/NCICB/infrastructure/cacore_overview/vocabulary). EVS vocabularies provide the semantic 'raw material' from which data elements, classes, and objects are constructed.

caBIG Compliance

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

The caArray database and analysis tools were developed to be consistent with caBIG compatibility guidelines that highlight use of controlled vocabularies, Common Data Elements (CDEs), well documented APIs and UML models. caBIG is a new initiative coordinated by NCI in partnership with other members of the cancer research community. caBIG seeks to create a network that links organizations, institutions, and individuals to enable the sharing of cancer research infrastructure, data, and interoperable tools. It is an open-access, open-source activity that promises to expedite progress in cancer research. caArray's compatibility with the caBIG design requirements facilitates the cross-silo use of cancer biology information to promote integrated cancer research.

caBIG Architectural Principles

Data and analytic services using uniform APIs and appropriate standard message formats are made available to caBIG. This allows programmatic interface to caArray data via the Enterprise JavaBeans (EJB) Managers, Microarray and Gene Expression Object Model (MAGE-OM) API, and the Microarray and Gene Expression Markup Language (MAGE-ML) document. caArray is built upon the MAGE-OM object model, Minimum Information About a Microarray Experiment (MIAME) and Microarray Gene Expression Data (MGED) Ontology standards. The APIs and messages support the delivery of data and of accompanying metadata in order to ensure that aggregated data sets are comparable. caArray supports and extends MAGE-OM that allows for the deep annotation of microarray experiments according to MIAME.

caBIG-Compliant Data Standards

caArray data is described by metadata elements that conform to the accepted ISO/IEC 11179 standard. The metadata resides in caDSR and is leveraged to achieve data interoperability and comparability. caArray supports MAGE-OM as described in Unified Modeling Language (UML).

Standards for the following international data exchange formats form the basis for caBIG-compliant data exchange in caArray:

- **MIAME** - http://www.mged.org/Workgroups/MIAME/miame.html
- **MAGE-ML** - http://www.mged.org/Workgroups/MAGE/mage-ml.html
- **MGED Ontology** - http://mged.sourceforge.net/ontologies/MGEDontology.php

caArray at caBIG Cancer Centers

caArray design is based on the caBIG compatibility guidelines to allow interoperability with other applications developed under the caBIG program. The design allows the integration with other data for comparative analysis. The caArray database can be deployed locally at NCI-designated cancer centers and other affiliated organizations as shown in *Figure 1.1*. The caArray system design facilitates data exchange between research centers. The data can be easily migrated to the central caArray database at the NCI when the data is published.

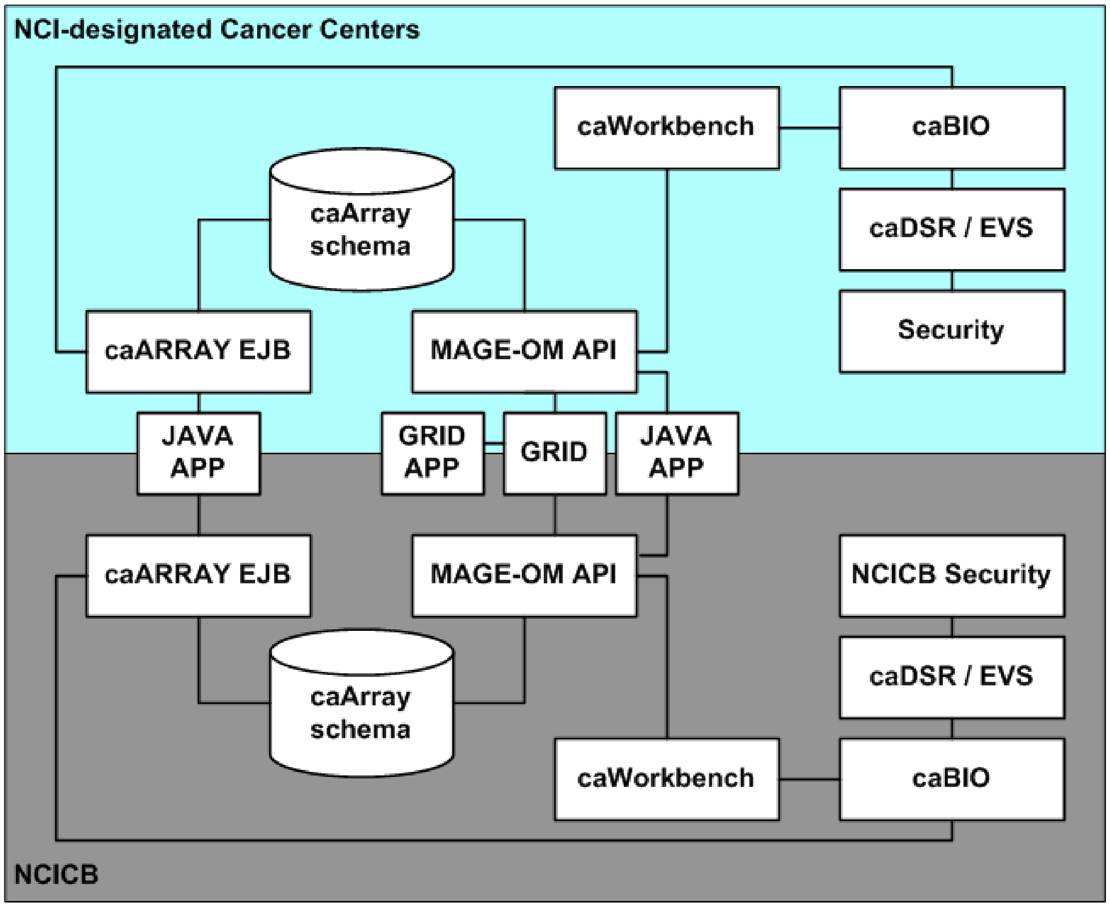

Figure 1.1 caArray database deployment

CHAPTER 2
OVERVIEW OF caARRAY

This chapter contains an overview of caArray and specifies the data management and data analysis tools used with caArray.

The NCICB's caArray consists of a microarray database and microarray data analysis and visualization tools. caArray is an open source project, and the source code and APIs are available for local installations at http://ncicb.nci.nih.gov/download under an open source license. The goals of the project are to make microarray data publicly available, and to develop and bring together open source tools to analyze these data.

The key features of caArray include the following:

- Allows researchers to manage and document their experiment data with full MIAME 1.1 compliant annotation
- Supports the import and export of MAGE-ML, promotes data exchange via an internationally standardized format.
- Supports submission and retrieval of Affymetrix and GenePix native format files
- Supports use of MGED controlled vocabularies (via OntologyEntry)
- Allows access via MAGE-OM API
- Is consistent with caBIG principles in providing an open-source application to NCI-designated cancer centers and other affiliated organizations

The caArray database is a standards-based open source data management system; version 1.0 was released in January 2005. caArray features MIAME 1.1 compliant data annotation forms, controlled vocabularies (MGED ontology), and MAGE-ML import and export. caArray also provides open interfaces for programmatic access to microarray data.

The caArray datasets and open source tools are publicly available and contain the following data management and data analysis tools. These tools are referenced here, but are not described in this guide. For more information about these tools, refer to the websites displayed.

1. **Data Management** - (https://caarraydb.nci.nih.gov/caarray/index.jsp)
   - **caArray Data Portal** - is a MIAME compliant data repository that allows submission of MIAME 1.1 level annotations and microarray data via web-based submission forms.
2. **Data Analysis** -
   - **geWorkbench** - https://gforge.nci.nih.gov/projects/geworkbench/ - A suite of tools for loading, visualizing and analyzing gene expression data that provides access to data from any repository with a MAGE-OM API
   - **GenePattern** - https://gforge.nci.nih.gov/projects/genepattern/ - An analysis platform that supports multidisciplinary genomic analysis and the integration of new technologies
   - **Bioconductor affy package** - https://gforge.nci.nih.gov/projects/bioconductor/ - A package that contains functions for exploratory oligonucleotide array analysis. Functionality includes background correction, normalization, and expression summaries.

CHAPTER

3

caARRAY ARCHITECTURE

This chapter contains the architecture of caArray including its design, implementation and interfaces.

Topics in this chapter include:

- *caArray Architecture Diagram* on this page
- *caArray Technologies* on page 12
- *caArray Interfaces* on page 14

caArray Architecture Diagram

The caArray system uses an architecture that separates the application into a series of tiers. A typical client-server system is a two-tier system (the client and the server that returns the data). This has the advantage of simplicity, but it ties the client very tightly to the details of the implementation model. To isolate the client from the implementation details, a data system can be built with one or more layers of 'middleware', software whose purpose is to act as a bridge between the server and the client. If changes are made to the server, the middleware is modified so that the client sees a consistent interface (API).

Some of the features of the caArray architecture as shown in *Figure 3.1* include:

- Servlet
- Web tier
- EJB/ Session Facade
- Data Transfer Object (DTO
- Data Access Object (DAO)
- MAGE-OM API

- MAGE Software Toolkit (MAGE-stk)
- caArray MAGE-ML Loader (caAMEL)
- Messaging Service

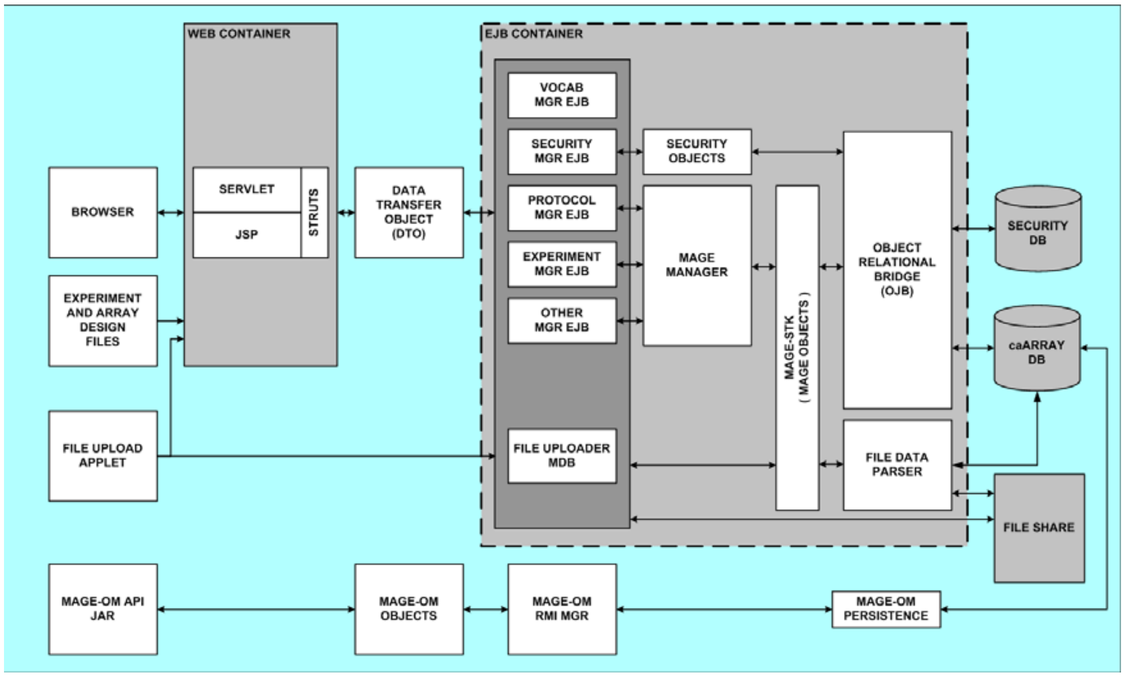

Figure 3.1 caArray architecture diagram

caArray utilizes an *n*-tier architecture as shown in *Figure 3.1.* It utilizes the J2EE framework and provides programmatic interfaces as well as a web portal interface for submission and retrieval of microarray data. The EJB interface provides transactional API capability with the use of DTOs that are used to transfer actual data. The Web portal, built utilizing Struts framework, uses the EJB and corresponding DTOs to perform transactions with the backend. An RMI-based query, API-based on MAGE-OM, provides programmatic query access to fine grain data, for example Quantitation data. Internally, caArray utilizes an Object Relational Mapping (ORM) tool called ObjectRelationalBridge (OJB) to abstract the actual Data source from the application and provide object-based access to data. The application also utilizes Java Messaging Service (JMS) from asynchronous parsing of large data files.

Role based security is implemented with the system, and a common NCICB security schema is used for users access. This configurable security architecture allows for Lightweight Directory Access Protocol (LDAP)- or Relational Database Management System (RDBMS)-based authentication and has concept of Groups, which allows for sharing of data amongst a consortium of researchers. MAGE-OM also utilizes the same common security service to filter objects based on users' roles and permission. This implementation is provided via Aspect Oriented Programming (AOP).

caArray Technologies

To create a scalable and robust distributed caArray architecture, caArray uses the following technologies:

- J2EE web container and J2EE EJB container (JBoss implementation) for data submission
- Java Messaging Service (JMS) for asynchronous processing of large uploaded documents
- Apache's OJB as the ORM tool that allows transparent persistence for "plain old" Java objects (POJOs) in the MAGE-stk toolkit
- MAGE-ML import via caAMEL uses MAGE-stk 1.1 with caArray-specific enhancements

Figure 3.2 represents the implementation of caArray. The top three boxes represent the client machine as noted by the client layer. The presentation, business and data access layers represent the layers on the server.

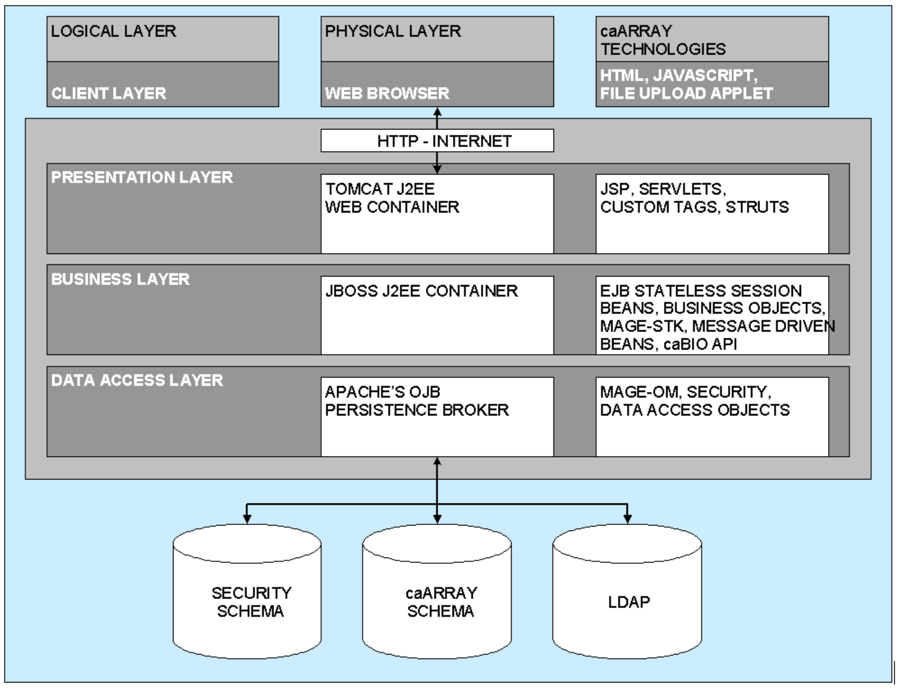
LOGICAL LAYER
CLIENT LAYER
PHYSICAL LAYER
WEB BROWSER
caARRAY TECHNOLOGIES
HTML, JAVASCRIPT, FILE UPLOAD APPLET
HTTP - INTERNET
PRESENTATION LAYER
TOMCAT J2EE WEB CONTAINER
JSP, SERVLETS, CUSTOM TAGS, STRUTS
BUSINESS LAYER
JBOSS J2EE CONTAINER
EJB STATELESS SESSION BEANS, BUSINESS OBJECTS, MAGE-STK, MESSAGE DRIVEN BEANS, caBIO API
DATA ACCESS LAYER
APACHE'S OJB PERSISTENCE BROKER
MAGE-OM, SECURITY, DATA ACCESS OBJECTS
SECURITY SCHEMA
caARRAY SCHEMA
LDAP

Figure 3.2 caArray implementation

The Presentation layer provides the GUI framework (see *Model View Controller 2* on page 17) and flexible web page layout (see *Composite View* on page 18).

Table *3.1* displays the business layer challenges and solutions used by caArray.

Design Challenge	***caArray Solution***
Data encapsulation	Transfer Objects
Common object to locate/lookup services	Service Locator
Abstract and decouple business services	Business Delegate
Encapsulate the DTO to MAGE-OM logic	Transfer Object Assembler

Table 3.1 Business layer design solutions

Design Challenge	***caArray Solution***
Uniform coarse-grained service access layer to clients	Session Facade
Access to vocab/metadata services	Configurable Interface Pattern
Process asynchronous MAGE-ML import & file uploads	Service Activator

Table 3.1 Business layer design solutions (Continued)

Table *3.2* displays the persistence design challenges and solutions used by caArray.

Design Challenge	***caArray Solution***
Decouple object and data source layer	Abstraction and Database Independence
Efficient materialization of objects	Lazy Materialization Pattern

Table 3.2 Persistence design solutions

caArray Interfaces

caArray provides access to its data via the following interfaces.

1. Programmatic
   - **MAGE-OM API** - Provides fine-grain search and retrieval of all caArray data via a RMI-based API. The MAGE-OM API objects are based on MAGE-OM 1.1 and are modified to map the MAGE objects to the new caArray database schema. An RMI security module provides for user/group level data access.
   - **caArray EJB API** - Provides transaction control, asynchronous processes, service location, common security and distributed capabilities for submission and retrieval of microarray experiments, MAGE-ML documents and its associated data files via DTOs.
   - **caArray Submission Portal -** Provides a user friendly GUI. caArray allows users to submit, annotate and retrieve experiments. caArray portal includes MIAME 1.1 compliance, MGED Ontology for annotation, MAGE-ML import, and support for file formats from Affymetrix, Agilent, GenePix, Illumina, and Imagene.
   - **caAMEL Service API** – The caArray MAGE-ML Loader (caAMEL) distribution provides an EJB remote interface for validating and loading MAGE-ML documents into caArray.
2. Document
   - **MAGE-ML import** – caArray supports MAGE-ML import for any well-formed and valid MAGE-ML document. caArray supports DataExternal and DataInternal for import of Quantitation data with MAGE-ML. The Quantitation data is generated in a tab delimited text file. The actual format of the DataExternal/DataInternal is specified within the XML, such as tab delimited and others. For more information, refer to the *caAMEL 1.0 User's Guide*: https://caarraydb.nci.nih.gov/caamel/.

- **Native Affymetrix, Agilent, GenePix, Illumina, and Imagene file formats** - Submission and retrieval of native Affymetrix , Agilent, GenePix, Illumina, and Imagene files

CHAPTER

4

caArray Design

This chapter describes design strategy and solutions employed by the caArray development team.

Topics in this chapter include:

- *Web Tier* on this page
- *Application Tier* on page 19
- *Database Tier* on page 32

Web Tier

Model View Controller 2

caArray's Presentation Layer utilizes the Model View Controller 2 (MVC2) design pattern. As shown in *Figure 4.1*, there is a separation of the application object (model) from the way it is represented to the user (view) from the way in which the user controls it (controller). The benefit is it separates the User Interface from the underlying structure.

MVC2 is implemented via Apache Struts framework. For more information on Struts, see http://struts.apache.org/ and http://struts apache.org/userGuide/index.htm.

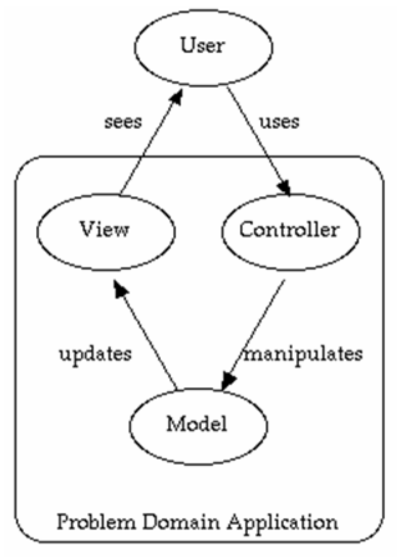

Figure 4.1 MVPC2 design pattern

Composite View

The Composite design pattern lets you treat primitive and composite objects the same as shown in *Figure 4.2*.

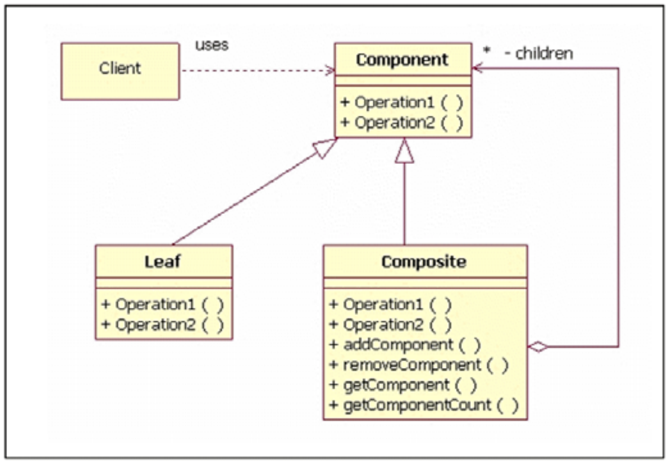

Figure 4.2 Composite view

The Apache Struts framework includes a JSP tag library, known as Tiles (http://struts.apache.org/userGuide/dev_tiles.html), which lets you compose a Webpage from multiple JSPs. Tiles allows you to build a flexible and reusable presentation layer.

Struts

The components utilized for Struts include:

- Model View Controller 2
- Tiles for layout (composite view pattern)

- Validation Framework (form validate method)
- Common functionality for Action classes in `ActionUtils`
- Some complex logic involved forms/objects in session.
- Delegates used for EJB calls
- Exception Handling mechanism

Application Tier

Data Transfer Objects

Data Transfer Objects (DTOs) provide a means to interact with EJB APIs and contain information that allows web tier and other applications to create, read, update or delete caArray data. The DTOs contain a mix of custom DTOs and domain DTOs. DTOs are structured according to the needs of the web tier while representing copies of MAGE-stk (domain objects). They take away the complexity of MAGE-stk objects.

There are three levels of DTO objects:

1. **Desc classes** - Contain minimal identifying data for search results. These classes name ends with Desc (for example, `ExperimentDesc` as shown in *Figure 4.3*).
2. **Data classes** - Contain data intended for update and insert operations of the object and its associated objects which must exist and may contain only the ID. These classes name ends with Data (for example, ExperimentData as shown in *Figure 4.3*).
3. **View classes** - Contain data intended for displaying the full details of an object often with all of its aggregations. For example, ExperimentView contains an array of ExperimentalFactorData while ExperimentData does not. These classes name ends with View (for example, ExperimentView as shown in *Figure 4.3*).

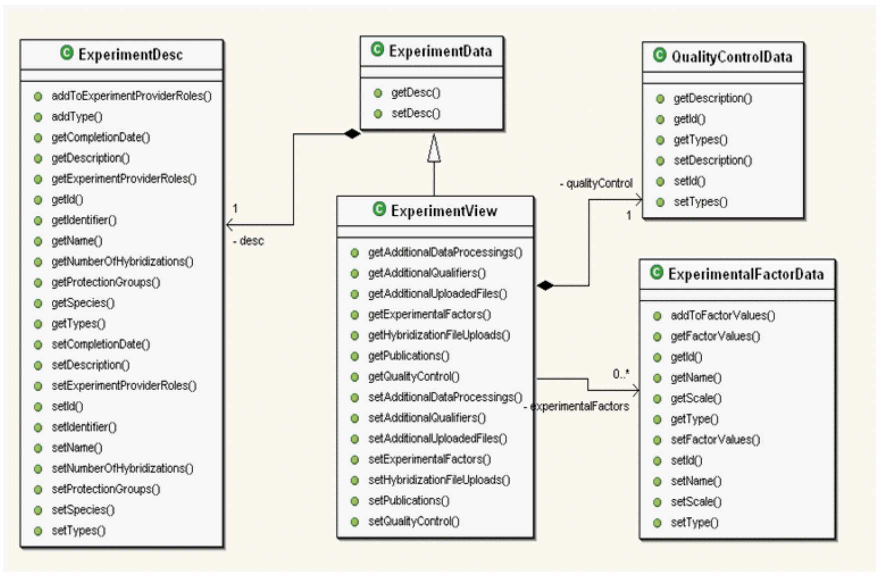

Figure 4.3 DTO class structure

The benefits of DTO include:

- All data needed by client and server-side processes are encapsulated in the object and sent/retrieved in one method call, therefore lessening the network impact.
- Strongly typed DTOs simplify server-side interface, providing easier code maintenance.

Figure 4.4, the DTO sequence diagram, includes the following:

- The application client sets user-entered values in the `ProtocolData` Transfer Object.
- The client application then invokes the EJB method to add protocol, sending the Transfer Object by value.
- The EJB method then retrieves all user-entered values from the Transfer Object, and begins business processing.

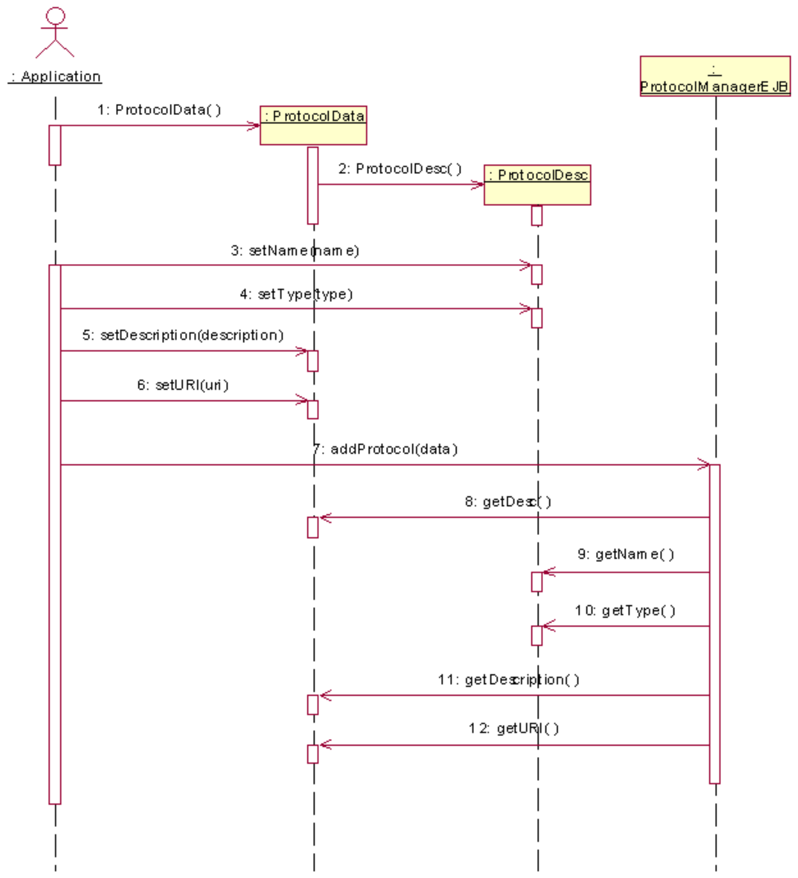

Figure 4.4 DTO sequence diagram

EJB Layer

The EJB layer contains Stateless Session Beans that follow the session façade pattern. They partition the business logic to minimize dependencies between the client and server while forcing use cases to execute in one network call and in one transaction. An

EJB method typically invokes a mirrored method from a `ManagerDB` subclass for a given use case. All EJBs extend from the base `AbstractSessionBean.`

The EJB layer has Bean-managed persistence with explicit use of Java Transaction API (JTA) such as begin, commit, rollback, and so forth. One EJB method typically uses one transaction except for hybridization data file parsing services, which use multiple sequential transactions.

Figure 4.5 depicts the logic flow through the EJB layer.

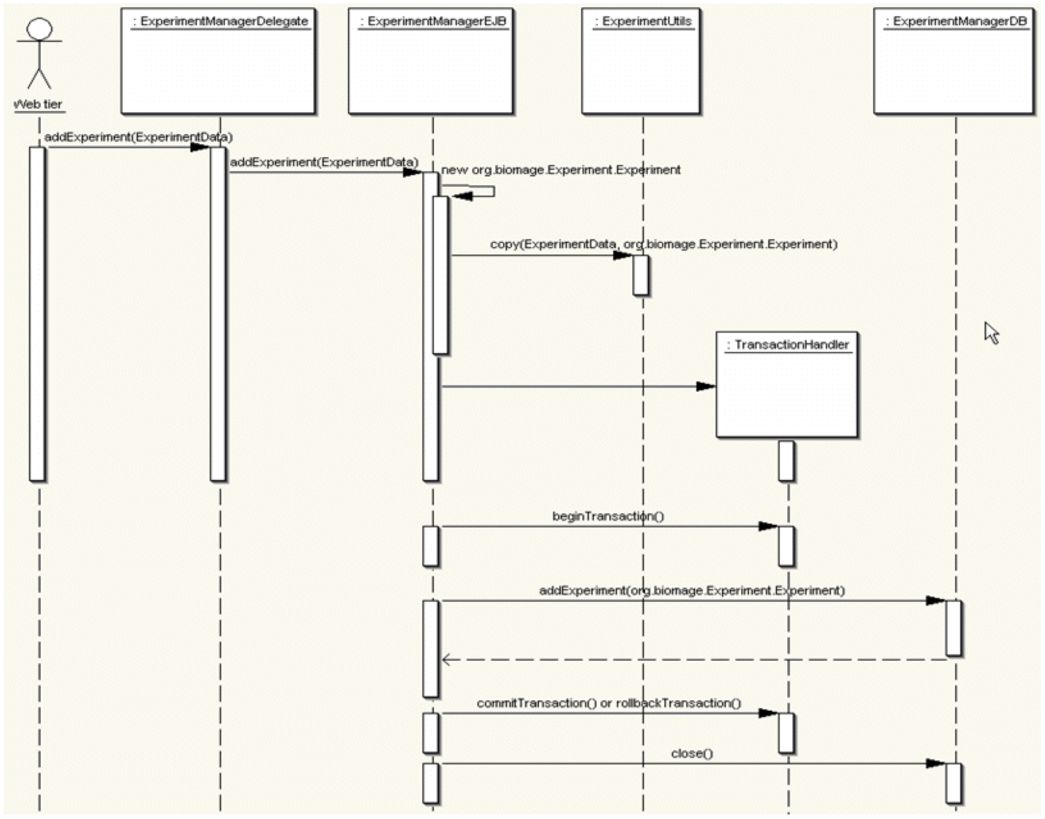
: ExperimentManagerDelegate
: ExperimentManagerEJB
: ExperimentUtils
: ExperimentManagerDB
Web tier
addExperiment(ExperimentData)
addExperiment(ExperimentData)
new org.biomage.Experiment.Experiment
copy(ExperimentData, org.biomage.Experiment.Experiment)
: TransactionHandler
beginTransaction()
addExperiment(org.biomage.Experiment.Experiment)
commitTransaction() or rollbackTransaction()
close()

Figure 4.5 Logic flow of EJB layer

Service Locator Pattern

As shown in *Figure 4.6* and *Figure 4.7*, caArray uses a common object, `ServiceLocator`, to locate/lookup EJB home objects and JMS service components (Connection Factory, Session, Topic, and so forth). The benefits include:

- A single point of control of the complexity of the lookup operation
- Ease of code maintenance for the creation of vendor-dependent initial context

- Increased application performance with cached EJB home objects.

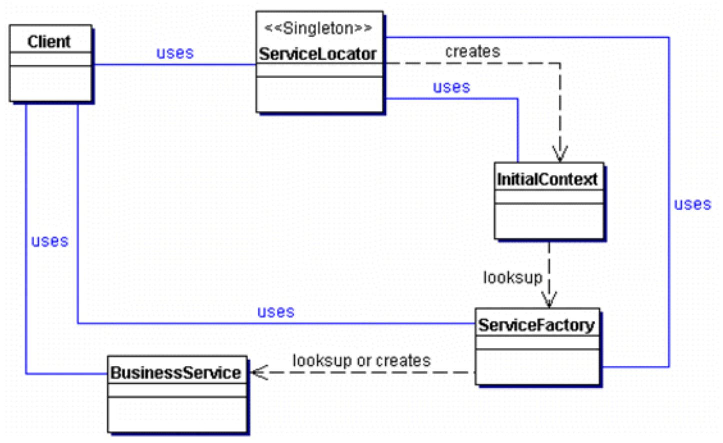

Figure 4.6 ServiceLocator object

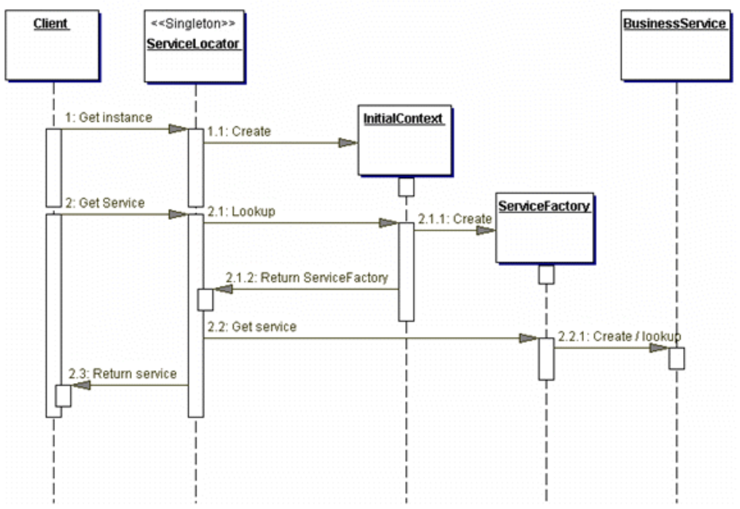

Figure 4.7 ServiceLocator sequence diagram

Business Delegate Pattern

As shown in *Figure 4.8* and *Figure 4.9*, caArray uses a wrapper class for each of the specific EJB Managers such as ExperimentManager, ProtocolManager, and so forth as BusinessDelegate reduces coupling between presentation-tier clients and business services. The benefit is it hides the underlying implementation details of the business service, such as the creation of EJB objects and accessing the details of business operations.

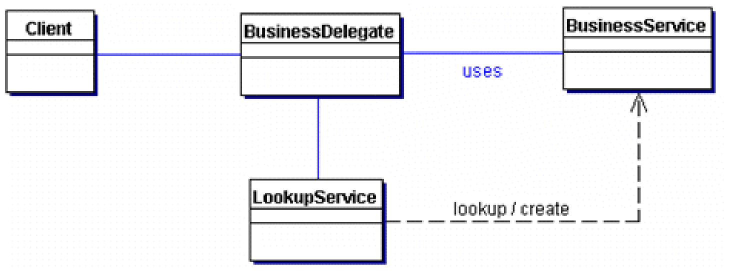

Figure 4.8 BusinessDelegate object

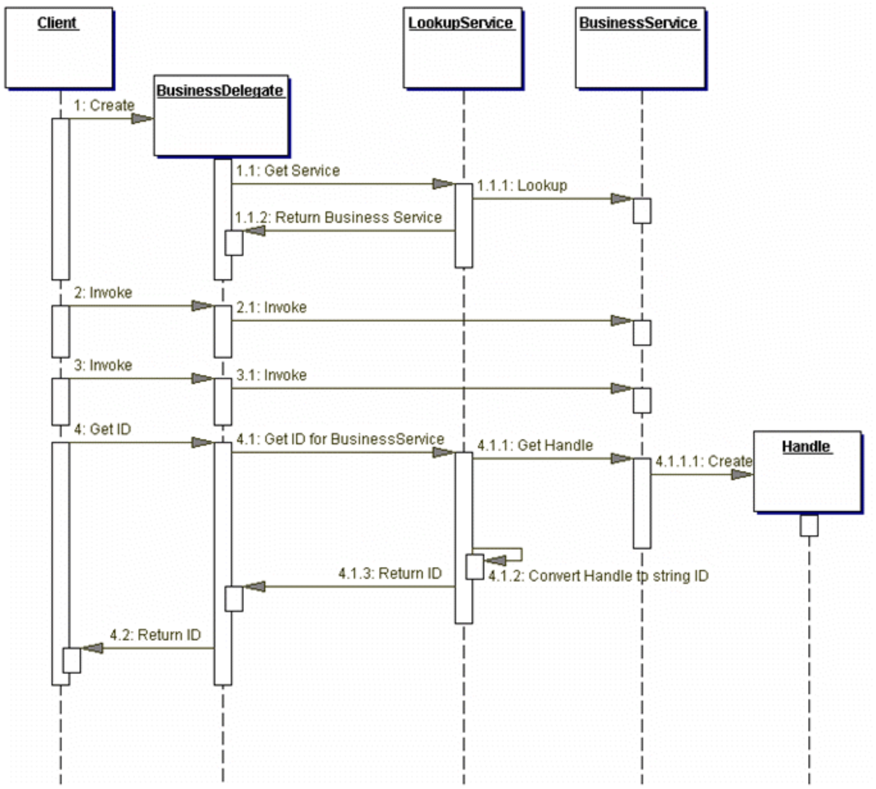

Figure 4.9 BusinessDelegate sequence diagram

Session Façade Pattern

As shown in *Figure 4.10* and *Figure 4.11*, caArray uses EJB managers such as `ProtocolManager`, `ExperimentManager` and so forth as a façade to hide the complexity of interactions between the business objects participating in a workflow. The Session Façade manages the business objects and provides a uniform coarse-grained service access layer to clients. The benefits include:

- Reduces coupling between client and server side code increasing manageability
- Provides common API interface access for multiple client applications
- Provides clean coarse grained access interface

Allows for distributed deployment of client and server distributions.

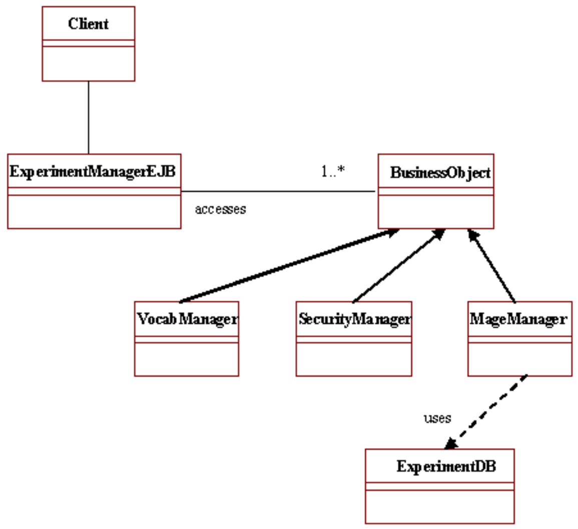

Figure 4.10 SessionFacade

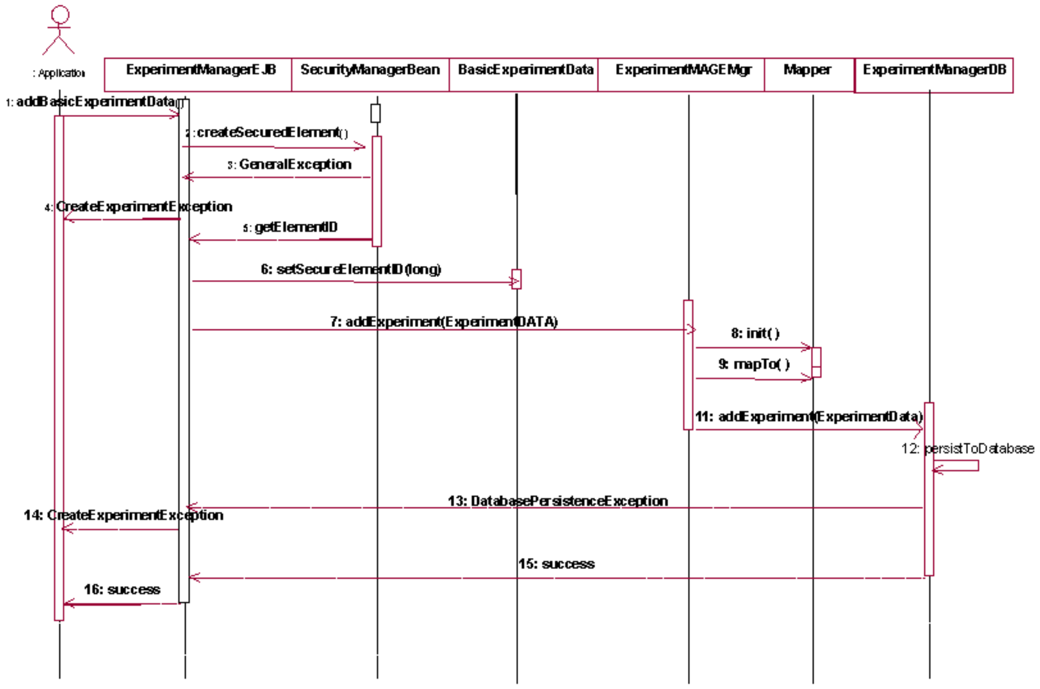
: Application
ExperimentManagerEJB
SecurityManagerBean
BasicExperimentData
ExperimentMAGEMgr
Mapper
ExperimentManagerDB
1: addBasicExperimentData()
2: createSecuredElement()
3: GeneralException
4: CreateExperimentException
5: getElementID
6: setSecureElementID(long)
7: addExperiment(ExperimentDATA)
8: init( )
9: mapTo( )
11: addExperiment(ExperimentData)
12: persistToDatabase
13: DatabasePersistenceException
14: CreateExperimentException
15: success
16: success

Figure 4.11 SessionFacade sequence diagram

VocabManager

The VocabManager retrieves controlled vocabularies and metadata via an interface pattern. The implementations associated with this interface are configurable. This allows you to plug caArray into the caCORE API, a different metadata repository or an XML metadata descriptor file. Through this interface, the VocabManager allows the caArray application to perform attribute type checking, validation and population of enumerated lists and other controlled vocabularies.

-

SecurityManager EJB

SecurityManager EJB provides programmatic APIs to access the security functionality. All the domain elements implement the secure element interface using these methods to get the security ID. Some methods are overloaded to take a collection of objects implementing `SecuredElementItf` for scalability. SecurityManager EJB does not know about the actual objects, it only deals with `securedElementId` passed in as an argument or from `SecuredElementItf`.

It is configurable to use LDAP or RDBMS for authentication. Authorization roles and so forth are accessed from RDBMS. `SecurityCommon.properties` and `LDAP.properties` contain configurable properties. `LDAP.properties` specifies LDAP server properties. `SecurityCommon.properties` specifies LDAP properties file names as well as authentication and authorization class names.

• LDAPAuthentication	LdapAuthenticationDAObj
• RdbmsAuthentication	RdbmsAuthentication
• Authorization	RdbmsAuthorizationDAObj

Two commonly used SecurityManager methods are listed here but there are many more.

1. `Role[] getUserRoleData(String userName, String password)`
    a. Returns a collection of Role objects if you have authenticated correctly
2. `public SecuredElementItf canUserAccessElement(SecuredElementItf object, String[] roleIds)`
    a. Meant to be called from EJBs, not from web tier
    b. Returns the object if user can access the element
    c. User is passed in via the session and is not passed explicitly as an argument
    d. Roles which are given declarative access to a method are passed in
    e. Corresponding overloaded method works with a collection

```
public SecuredElementItf[] canUserAccessElement(SecuredElementItf[] objectIds, String[] roleIds)
```

MAGE Manager

caArray uses the MAGE Manager as a Transfer Object Assembler to build the required MAGE model or sub model as shown in *Figure 4.12* and *Figure 4.13*. It uses DTOs to retrieve data from various MAGE-stk business objects that define the model or part of the model. The mapping between the DTOs and MAGE-stk is generated using XDoclet and persists in an XML document. The benefits of the MAGE Manager include:

- It encapsulates the MAGE-OM and hides it from business logic.
- It shields business logic from complexity of assembling transfer objects.

The use of the generic method reduces the code-base and enhances maintenance.

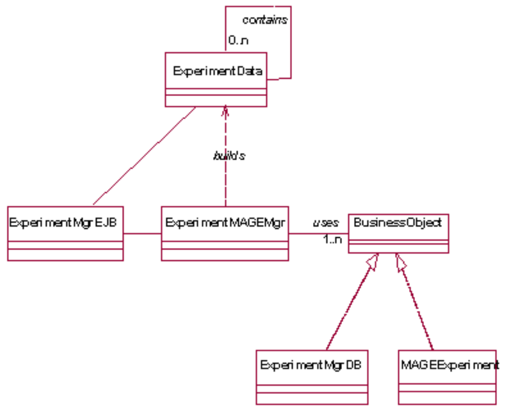

Figure 4.12 MAGE Manager Transfer Object Assembler Pattern

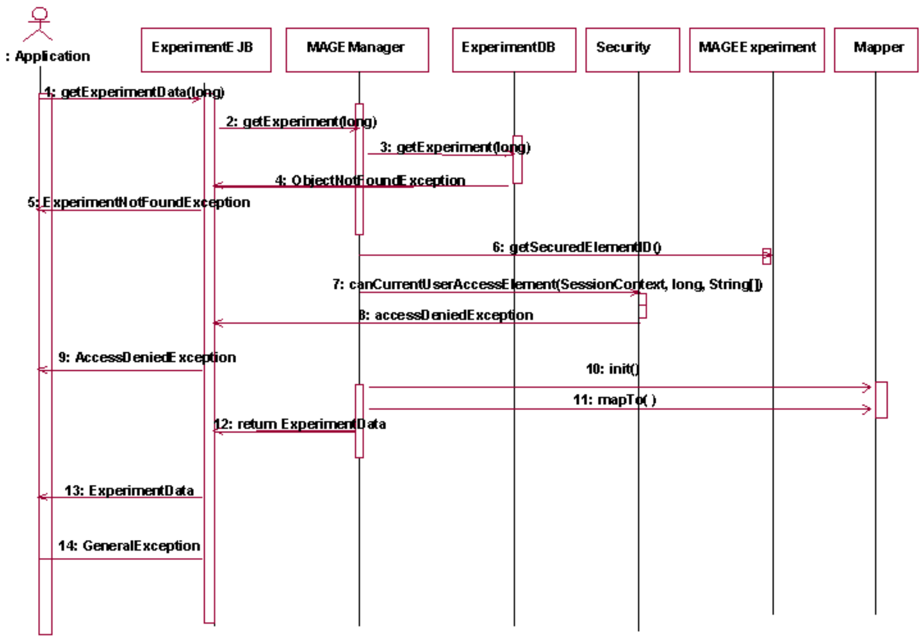

Figure 4.13 MAGE Manager Transfer Object sequence diagram

MAGE-stk

MAGE-stk is a collection of open source packages that implement the MAGE-OM in various programming languages. The MAGE web site http://mged.sourceforge.net has links to the MAGE-OM UML model, MAGE-stk source and javadoc, and user forums. MAGE-stk is used as domain and persistence objects for caArray. MAGE-stk objects are stored and retrieved using OJB.

DTO MAGE-stk Conversion

DTO - MAGE-stk conversion is performed to make copies of data stored as MAGE-stk objects and represent them as DTO objects and vice versa. This is accomplished by generalizing the copying of a source object to a target object based on the mapping of the classes and their attributes. The DTO - MAGE-stk conversion utility uses Apache's commons-beanutils for fast cache of reflected methods, avoiding inefficient method reflection in subsequent calls.

Shown in *Figure 4.14* is an example of a DTO XML mapping.

```
<DTOmapping>
  <DTOClass
name="gov.nih.nci.caarray.common.data.vocab.DatabaseData">
    <TargetClass name="org.biomage.Description.Database">
      <ConvertField id="id"
         from="id_" to="id"
         fromtype="long" totype="long">
      </ConvertField>
      <ConvertField id="name‟
  from="name_" to="name"
         fromtype="java.lang.String"
totype="java.lang.String">
      </ConvertField>
      …
    </TargetClass>
  </DTOClass>
<DTOClass
name="gov.nih.nci.caarray.common.data.vocab.VocabData">
    <TargetClass
name="org.biomage.Description.OntologyEntry">
      <ConvertField id="id"
         from="desc_.id_" to="id"
         fromtype="long" totype="long">
      </ConvertField>
      <ConvertField id="database"
         from="database_" to="ontologyReference.database‟
fromtype="gov.nih.nci.caarray.common.data.vocab.DatabaseData"
         totype="org.biomage.Description.Database">
      </ConvertField>
      …
    </TargetClass>
  </DTOClass>
<DTOClass
name="gov.nih.nci.caarray.common.data.vocab.VocabDesc">
    <TargetClass
name="org.biomage.Description.OntologyEntry">
      <ConvertField id="id"
         from="id_" to="id"
         fromtype="long" totype="long">
      </ConvertField>
      …
    </TargetClass>
  </DTOClass>
  …
<DTOmapping>
```

Figure 4.14 DTO XML mapping

The conversion functionalities include:

- **Copy an object of one object type to an object of another type**: Copy an object of one type by value **to and from** an object of another type using the mapping of their attributes
- **Copy attribute to attribute**: Attributes can be of the same or different types (for example, String to int, int to int, and so forth). This is supported by `commons-utils`.
- **Copy object reference to object reference**: Based on the mapping of the referenced object types. For example, copying VocabData.database attribute to OntologyEntry.ontologyReference.database attribute is done using the mapping of the classes DatabaseData to Database.
- **Copy an array or list of primitives or object references to an array or list of primitives or object references**: For example, String[ ] to String[ ], String[ ] to `java.util.ArrayList`, int[ ] to Integer[ ], DatabaseData[ ] to Database[ ], DatabaseData[ ] to `java.util.List` are all possible

Note: Copying null results in null; Array or list attributes that are set to null are converted to null in the target object. Always initialize/return zero-size arrays and empty lists instead of null.

The package name for the converter is `gov.nih.nci.caarray.services.util.Dataconverter`. The `DataConverter` class contains static methods create and update for performing the conversion. The use of the `DataConverter` class is abstracted from the EJB layer using static classes such as `services.experiment.ExperimentUtils`, and so forth.

Object Relational Mapping

ObjectRelationalBridge

caArray uses Apache's OJB as the ORM tool that allows transparent persistence for POJOs against relational databases. The underlying utilization of OJB is standard. See http://db.apache.org/ojb for more information on using OJB.

The benefits of OJB include:

- Structured Query Language (SQL) generation is based on ORM (You don't have to write SQL)
- Create, Read, Update, Delete (CRUD) operations work with Java objects directly
- A wide range of database platforms including Oracle, MySQL, Sybase, etc. are supported
- Caching is performed; Once objects are retrieved, they stay in memory and are handed out for subsequent retrievals
- Connection pooling, statement cache

caArray utilizes OJB's persistence facility to manage the connection to the data source, store and restore MAGE-stk objects. The mapping of MAGE-stk objects and their relationships to the database entities is described in OJB repository files to allow OJB to know how to persist or restore them.

The caArray schema and OJB mapping repository were originally generated by annotating MAGE-stk with XDoclet tags and using Apache's Torque. A variety of updates/corrections has been incrementally made to the schema and OJB repository such as:

- Key constraints
- Column size
- Java-to-SQL type conversion
- Optimized inheritance hierarchy mappings
- Adjusted settings for cascade-update, -deletion and -retrieval

caARRAY utilizes OJB's table-based high/low sequence generator using the OJB_HL_SEQ table to create globally unique primary keys for the MAGE-stk tables. caArray utilizes OJB HI_LO sequence manager so you can set the max_seq in OJB_HL_SEQ table against SEQ_Extendable to set the right values. Ids are long by default and so are biomaterials. Ids in DTOs are java long.

OJB's persistence mechanism is abstracted from caARRAY by a small set of classes in package `gov.nih.nci.caarray.services.util.db`. Hiding OJB's specifics facilitates replacing it if needed.

`ManagerDB` is a wrapper of OJB's PersistenceBroker (which can be thought of as Java Database Connectivity (JDBC) connection, though there are some differences). All persistence classes extend from `ManagerDB`. Table *4.1* contains important methods in `ManagerDB`. They work hand-in-hand with the settings for auto-update, auto-delete and auto-retrieve for the attribute-descriptor mappings in the OJB repository.

Method	***Description***
`storeObject(Object object)`	Insert or update an object in the database along with its associated objects if `auto-update=true`
`deleteObject(Object object)`	Delete an object from the database along with its associated objects if `auto-delete=true`
`updateObject(Object object)`	Update an object in database and insert or update its associated values if `auto-update=true`. Cascade update (`auto-update=true`) for 1-n does not mean the removal of elements on the n-side association effect the deletion of them, rather just their relationship with the parent object.

Table 4.1 ManagerDB Methods

Method	Description
`link(Object instance, String attributeName, boolean insert)`	Manually associate the parent object with the associated object(s) for persistence based on their relationship when the attribute has `auto-update=false` • **For 1-1 relationship**: Not needed. The parent object is automatically set with the foreign key identifying the associated child object • **For 1-n relationship**: Set the primary key of the parent object as the foreign key in the associated list of child objects. **Note:** This method is called after the parent object is stored and before the child objects is stored • **For m-n relationship**: Insert or remove the links in the many-to-many association table as specified by the mapping repository. **Note:** This method is called after both parent and child objects are stored.
`close()`	Close connection to database, similar to JDBC `Statement.close()`

Table 4.1 ManagerDB Methods

`TransactionHandler` as shown in *Figure 4.15* was added to allow for the same transaction operations used in both managed and non-managed (local) environments:

- Local accessed database uses the JDBC’s autoCommit flag while J2EE uses JTA, i.e. UserTransaction
- Abstract UserTransaction for ease of exception handling

`PersistenceBrokerManager` as shown in *Figure 4.15* manages OJB’s brokers used by `ManagerDB` subclasses. The `ManagerDB` subclasses are expected to provide specific persistence operations for domain objects and often contain business logic as well. For example, `ProtocolManagerDB` has methods `addProtocol`, `updateProtocol`, `deleteProtocol`, and so forth. Therefore, many methods such as `storeObject`, `updateObject` and `deleteObject` in `ManagerDB` are protected.

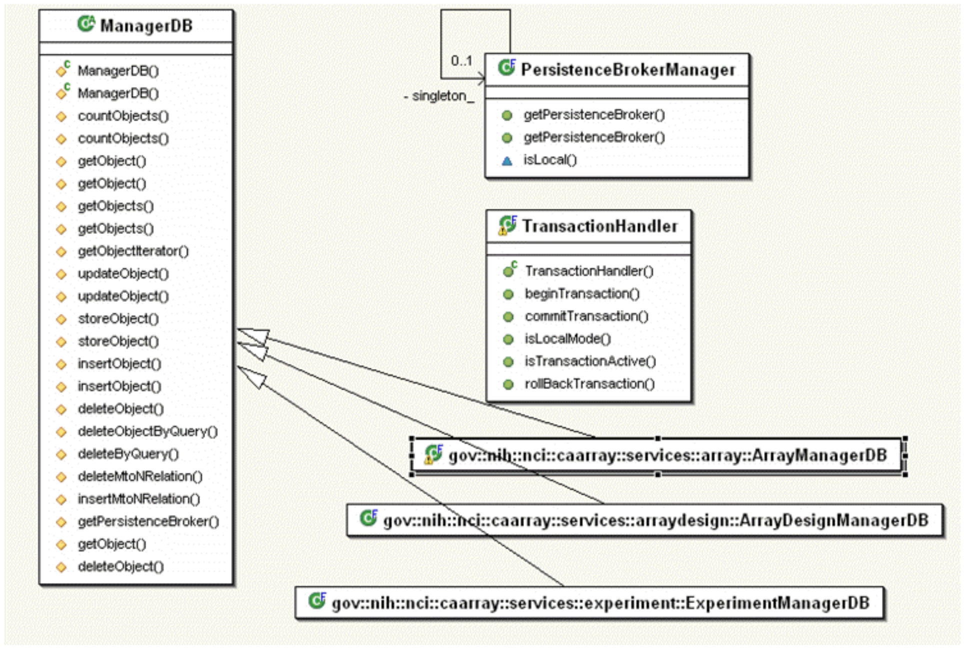

Figure 4.15 OJB Abstraction layer and DAO classes

Reading of incorrect repository files should not be an issue as the OJB object names in MAGE-OM are different from that in caArray even though they are mapped to the same tables. The class `contact` is an abstract class and a `person` or `organization` is a concrete class of `contact`. OJB needs to know in which table to look for a given `contact` object. For this task, OJB goes to the `contact` table and reads in the value of the `OjbConcreteClass` column. Dependent on this value, OJB goes to the `person` or `organization` table. However, if the value in this class is incorrect, empty or OJB cannot read it, then an error occurs.

Lazy Materialization Pattern

Lazy materialization refers to loading data collections only when they are actually required. In caArray, lazy loading (also known as lazy materialization) is a capability that is implemented via OJB collection proxies. Lazy materialization is implemented using a proxy to make the calls to manipulate the collection. This can help you in reducing unnecessary database lookups and object materialization.

For example, you load an Array Design object from the database, which contains a collection of 15000 feature objects. Without proxies, all 15000-feature objects are immediately loaded from the database, even if you are not interested in them but just want to lookup the description-attribute of the array design object.

With a proxied class, the collection is implemented via a proxy that implements the same interface as the real collection but only materializes the objects in the collection when necessary. Once you access such a proxied collection, it loads its collection objects by OJB and executes the method call.

Since the actual Java class uses an interface for the collection, the OJB proxy can be utilized without changing, or creating dependencies within the classes code. The benefits include allowing materialization of objects efficiently in terms of both time and memory usage.

Database Tier

The database tier of caArray contains caArray data sources that consist of the following:

- caArray Database
- Security Database
- File Share

caArray Database

The caArray database schema is originally generated from the MAGE-OM then modifications are made for performance and caArray functionality. The caArray data model was derived by annotating the MAGE-OM API with XDoclet tags. An XDoclet module (Apache's Torque) processed the Java files. The process generated the OJB `repository.xml` file and the SQL Data Definition Language (DDL) schema. The schema was then optimized for performance. Tables prefixed with `as_` are security tables, `ca_` are caArray specific tables and no prefix indicates a MAGE-OM table (even though there still could be changes). Reference the caArray database DDL (http://caarray.nci.nih.gov/documentation/) for more information.

Security Database

The security database has its own schema and is physically located in the same database as the caArray database, but it does not have to be.

File Share

The file share stores the uploaded hybridization files, MAGE-ML, and other files. The file share location is selected during caArray installation (see the *caArray 1.6 Local Installation Guide* for more information). The property file `caarray.properties` contains the location of the various subdirectories in the file share used for the various types of uploaded artifacts.

CHAPTER

5

caARRAY DATA ACCESS SECURITY

This chapter describes declarative as well as programmatic security as it pertains to caArray.

Topics in this chapter include:

- *Access Privileges* on this page
- *CRUD Permission Levels* on page 35
- *Security EJB* on page 35
- *Enabling Security* on page 36
- *EJBs* on page 36
- *Programmatic Security* on page 37

Access Privileges

Table *5.1* displays the Roles that can be assigned in caArray. The **Constants Class Reference** column defines constants for the roles that can be referenced in the code.

Access Privilege	***Description***	***Role in Database***	***Constants Class Reference***
User	General access to data that has been identified as public or is not protected; will use the system to submit data or search through and utilize existing data	User	USER = "User"

Table 5.1 caArray Access Privileges

Access Privilege	***Description***	***Role in Database***	***Constants Class Reference***
User Manager	Creates and manages users, roles, protections groups, etc.	UserManager	USER_MANAGER = "UserManager"
Protocol Manager	Manages protocol, hardware and software definitions	ProtocolManager	PROTOCOL_MANAGER = "ProtocolManager"
Array Design Manager	Manages caArray design definitions	ArrayDesignManager	ARRAY_DESIGN_MANAGER = "ArrayDesignManager"
Experiment Manager	Manages experiment data; is responsible for annotating biomaterials and maintaining the data associated with experiments	ExperimentManager	EXP_MANAGER = "ExperimentManager"
Curator	Has full capability to modify data for specified protection groups and is responsible for the integrity of the data for a particular group or a set of groups	Curator	CURATOR = "Curator"
BioMaterial Manager	Manages biomaterial definitions and annotations	BioMaterialManager	BIO_MANAGER = "BiomaterialsManager"
Vocabulary (“Ontology”) Manager	Manages dynamic vocabulary elements	VocabularyManager	VOCABULARY_MANAGER = “VocabularyManager”
Data Owner	Any user that is currently listed as the owner of any protected data element in the system		
Unauthenticated user		N/A	UNAUTHENTICATED_USER = "Public"

Table 5.1 caArray Access Privileges (Continued)

Notes:

- Unauthenticated public users are assigned a principal with name **Public** and role of **User**. In the security database, there must be a user with name **Public** which has role **User** on protection group **Public**. This data is essential security data that is tied to the code and must be populated in the security database.
- All the users of the system always have a **User** role on the protection Group **Public**. This allows caArray to search secured elements with visibility **Public**.

CRUD Permission Levels

Declarative security at EJB level controls the CRUD authorization for a given role. The programmatic security determines the given users access to the particular secured element.

- **Create—**User must have the appropriate role to call the method in the corresponding EJB. For example, only a user with ArrayDesignManager role can create an array design. The container blocks all other calls to create method if the user does not have the ArrayDesignManager role. The creator of the secured element becomes its owner.
- **Read—**Any user with role **User** can read any secured element. The secured element has to be part of a protection group on which the user has the **User** role. An unauthenticated user is given a username of **Public** and assigned the role **User** to the protection Group **Public**.
- **Update/Delete—**A user needs to be owner of a secured element to be able to edit or delete the secured element. In order for a user to update/delete an array design, the user not only needs to have ArrayDesignManager role, but also be the owner of the array design. By transferring ownership to a group account, the group is able to update/delete the secured element.

Security EJB

`canUserAccessElement`method has a new implementation. This is necessary to check for ownership in case of update/delete permission. In case of read only access, additional checks are performed for a user having the appropriate role in the Protection Group which has the secured element.

The `canUserAccessElement` method takes in objects implementing `securedElement` Interface. This interface has following methods:

```
public String getSecuredElementId() ;
public void setSecuredElementId(String securedObjectId) ;
public boolean isEditable() ;
public void setIsEditable(boolean editable) ;
```

The security checks the secured element for access. If the caller is owner, security sets the `isEditable` flag to **true**. The calling EJB makes the determination based on the `isEditable` flag whether to delete/update the secured element. If `isEditable` is **false**, the caller can only read it. If `isEditable` is **true**, the caller can edit it. The search method also makes use of this interface. The `isEditable` flag is transferred to DTO and sent back to GUI. The GUI makes the determination based on the `isEditable` flag whether to display a **Modify** button to the user.

There are two versions of the `caUserAccessElement` method.

1. One takes a single object implementing `securedElementInterface`. It returns that object with the appropriate setting for the `isEditable` flag. If the user cannot access it, it returns no object.

2. The second version takes a collection of `securedElementInterface` and returns a subset that the user can access. Each object is appropriately set with the appropriate `isEditable` flag.

Enabling Security

Complete the following steps to implement and enable security properly:

1. Use the latest `login-config.xml` that has a definition for "caarray" as the authentication configuration. This makes use of the custom login model for authentication as well as authorization. Login code makes the appropriate call on the client and sets appropriate Java Authentication and Authorization Service (JAAS) principals/subjects. The client login should also put appropriate credentials in `sessionContext` that each EJB could use to authenticate.

   Note: The EJBs need to authenticate caller credentials on each call to each method, as a dubious client can fake the principal and credentials gaining access to the system/data. This could be a potential security hole if the deployment is outside a firewall.
2. If using Eclipse, set XDoclet setting for JBoss by setting the security domain to `java:/jaas/caarray`. If using the ant scripts, uncomment the security domain tagline.

EJBs

Security EJB has both local as well as remote interfaces. EJBs should call `localhome` as it makes a call by reference and avoids expensive marshalling/unmarshalling of data for calling remote methods. If using Eclipse, set `localhomeinterface` and `localinterface` tags for generation.

In respective EJBs, set appropriate XDoclets to enable and enforce security. Based on a caller's JAAS role, the container allows or disallows calls to EJB methods.

Implementation at class level:

```
* @ejb.security-identity use-caller-identity="true"
* @ejb.security-role-ref role-name="User" role-link="User" etc…
```
for all the roles.

```
* @jboss.container-configuration name="Standard Stateless Ses-
sionBean"
```

For each of the secured methods set:

```
* @ejb:permission role-name="User" etc.
```
The following is added to the relevant `ejb-jar.xml` file via XDoclet:

```
<method-permission >
<role-name>User</role-name>
<method >
```

```
<ejb-name>gov/nih/nci/caarray/services/protocol/ejb/
HardwareManager</ejb-name>
<method-intf>Remote</method-intf>
<method-name>search</method-name>
```

Programmatic Security

Programmatic security allows for more specific security check for a protection element. It is performed in the EJB with the help of helper methods. `SecuredElementItf` is used as shown in *Figure 5.1*. All MAGE objects implement this.

```
public interface SecuredElementItf
{
   public String getSecuredElementId() ;
   public void setSecuredElementId(String securedObjectId) ;
   public boolean isEditable() ;
   public void setIsEditable(boolean editable) ;
}
```

Figure 5.1 SecuredElementItf code

Chapter
6

caArray Download Site

This chapter describes the caArray and MAGE-OM files that can be downloaded.

Topics in this chapter include:

- *caArray Portal* on this page
- *caArray MAGE-OM API* on page 40

caArray Portal

The caArray download web site https://ncicb.nci.nih.gov/download/index.jsp contains the following caArray software for distribution. The caArray Portal distribution contains files in Table *6.1*. See the *caArray 1.4 Local Installation Instructions* for detailed steps to install the caArray source code and create a local caArray database.

File	***File Description***
Database Dump file	Contains a caArrayop.dmp.zip file for caArray seed data
caArray DDL Schema file	Contains the caArray_DDL.sql schema file
Update Database Scripts	Contains a SQL files to update the database schema to the specified version
caArray Portal Source Code	Contains the caArray_SourceCode_{version}.zip file with the source code
caArray Local Installation Instructions	Contains instructions to install the caArray portal source code and create a local caArray database
caArray Java documents	Describes the caArray APIs
caArray Microarray Files	Contains seed data for caArray Microarray files

Table 6.1 caArray download files

File	File Description
UCSF Spot to GenePix Utility	Is a Perl script utility that converts a UCSF Spot clone list file or a UCSF Spot CGH results file to GenePix .gal files and .gpr files: ftp://ftp1.nci.nih.gov/pub/caARRAY/Utilities/spot2genepix.pl. Detailed instructions are included in the Perl file.

Table 6.1 caArray download files (Continued)

caArray MAGE-OM API

The MAGE-OM API download site (http://ncicb.nci.nih.gov/download/index.jsp) contains the files in *Table 7-2*. See the *MAGE-OM API Installation Instructions* for detailed steps to build the source code. For more information on caArray MAGE-OM API, see *Chapter 8*.

File	File Description
MAGE-OM API Source Code	Contains the source code, client jar file, and Java documents
MAGE-OM API Installation Instructions	Contains the instructions to build the source code which builds a client and a server
MAGE-OM API Java Documents	Describes the MAGE-OM APIs
caArray-MAGE-OM-CLIENT.jar	Client jar file set up to access the NCICB server

Table 6.2 MAGE-OM download files

The caAMEL download site (http://ncicb.nci.nih.gov/download/index.jsp) contains the files in Table *6.3*. See the caAMEL Installation Instructions for detailed steps to install caAMEL. For more information on caAMEL, see *Chapter 9 MAGE-OM API*.

File	File Description
`caamel-1.0.zip` (and `tar.gz`)	Binary distribution of the caAMEL application. This is the recommended distribution for users who wish to install and use caAMEL locally.
`caamel-1.0-src.zip` (and `.tar.gz`)	Source code distribution of the caAMEL application. This is intended for developers who want access to the source code for informational purposes or who wish to modify the code base to suit their local needs.
caAMEL Installation Guide	Describes how to install and configure caAMEL from either the binary distribution or source distribution.

Table 6.3 caArray MAGE-ML Loader

CHAPTER

7

MAGE-OM API

This chapter contains the architecture, design and implementation of the MAGE-OM API.

Topics in this chapter include:

- *MAGE-OM API Introduction* on this page
- *MAGE-OM Directory Structure* on page 42
- *Accessing the MAGE-OM Production Server* on page 43
- *Testing the MAGE-OM Production Server* on page 43
- *MAGE-OM Architecture* on page 44
- *MAGE-OM Security* on page 47
- *Directable* on page 50
- *Persistence* on page 50

MAGE-OM API Introduction

The caArray MAGE-OM API is a set of Java objects that adhere to the object model defined by the OMG Gene Expression Specification, http://www.omg.org/technology/documents/index.htm. The caArray MAGE-OM API objects provide access to data in the caArray database via a Remote Method Invocation (RMI) call issued to a dedicated MAGE server at NCI or any other site with an accessible MAGE-OM server installation. There are two primary types of objects defined in the API as shown in *Figure 7.1*:

1. MAGE-OM-compliant interfaces
2. Custom MAGE-OM Impl (implementation) objects

The MAGE-compliant objects are defined as Java interfaces, which the custom MAGE-OM Impl Java classes implement. This ensures the custom MAGE-OM Impl provides a MAGE-OM compliant API. The MGED Society web site, http://www.mged.org/

Workgroups/MAGE/mage.html, is an excellent source for supplemental material on the MAGE object model.

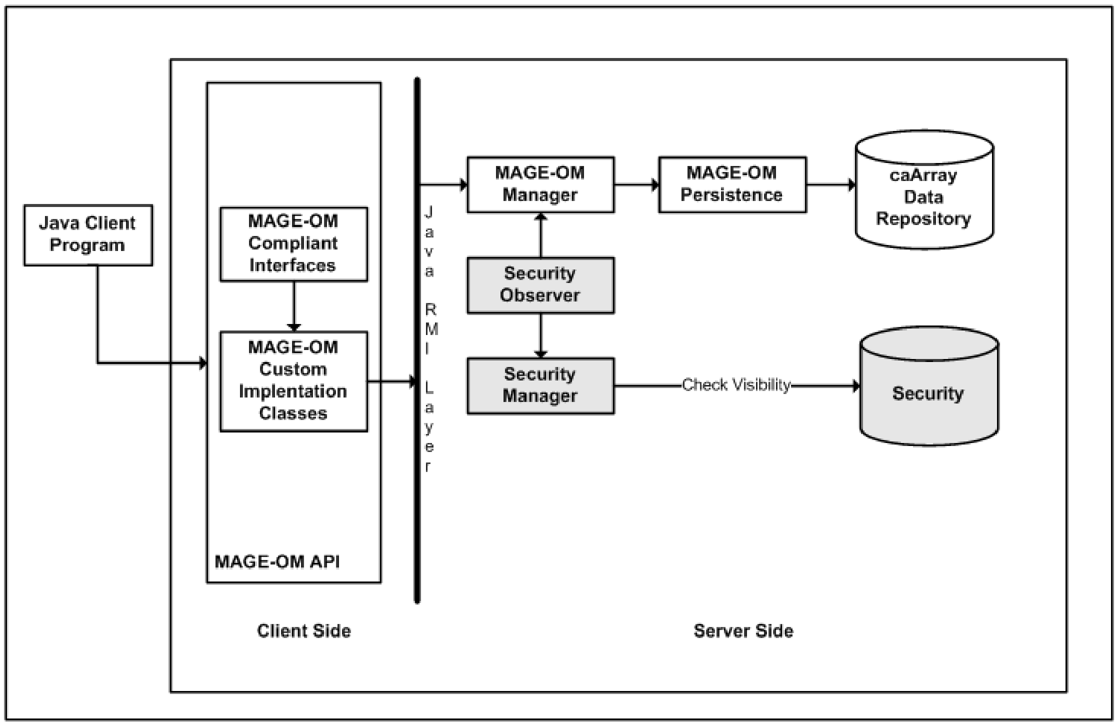

Figure 7.1 MAGE-OM architecture

The MAGE-OM API provides the following:

- MAGE-OM API objects are based on MAGE-OM 1.1 and are modified to map the MAGE objects to the new caArray database schema.
- RMI security module is incorporated and tested for user/group level data access.

MAGE-OM Directory Structure

After installing MAGE-OM API, the top-level directories are listed in Table *7.1*.

Directory	***Description***
conf	Contains configuration files
docs	Contains MAGE-OM Javadocs
download	Contains demonstration code
lib	Contains all JAR files
models	Contains UML models
src	Contains source code
test	Contains test source code

Table 7.1 MAGE-OM directory structure

Accessing the MAGE-OM Production Server

Following is the RMIconnection information to access the production server. The server URL and port should be defined as:

- `rmi.public.host =caarray-mageom-server.nci.nih.gov`
- `rmi.public.port =8080`

The following steps summarize how RMI works.

1. The RMI service is started after reading `rmi-server.properties` that defines RMI service host name, data port, registry port, and shutdown port.
2. `RMIBinder` creates those using `caCORERMISocketFacotry` then starts the RMI registry.
3. Two service objects, `RMISecureSessionManagerRmote` and `RMISearch-CriteriaHandlerRmote`, are registered on this registry.
4. They are proxy in server site, waiting calls from clients. The implementation to handle the real call is `DefaultSearchCriteriaHandler` and `SecureSes-sionManagerImpl` objects.
5. A client, based on `rmi-client.properties`, has access to those service points by looking up the RMI service registry remotely. Registry returns a directable object that can be used to enforce session management and access remote objects using session ID once logged in.

Testing the MAGE-OM Production Server

The code for a sample test case displays in *Figure 7.2*. More test cases can be found in the `test/src/java/gov/nih/nci/mageom/test/`package.

```
public void testRetrieveExperimentDescriptions() {
  try {
  String serverLoc = System
  .getProperty("mageom.test.directable.serverLoc1");
     if (serverLoc == null) {
      fail("No server location provided");
     }
   SecureSession sess =
SecureSessionFactory.defaultSecureSession();
  ((Directable) sess).direct(serverLoc + "SecureSessionManager");
  sess.start(CAARRAY_USER, CAARRAY_USER_PWD);
  String sessionId = sess.getSessionId();
   ExperimentSearchCriteria sc = SearchCriteriaFactory
     .new_EXPERIMENT_EXPERIMENT_SC();

  // sc.setId(new Long("1015897536498154")) ;
   sc.setMaxRecordset(new Integer(20));

   ((SessionObject) sc).setSessionId(sessionId);
   ((Directable) sc).direct(serverLoc + "SearchCriteriaHandler");
   SearchResult sr = sc.search();

   Experiment[] result = (Experiment[]) sr.getResultSet();
    if(result.length > 0){
                 for(int x=0;x<result.length;x++){
                     System.out.println("Experiement : "+x) ;
                     Description[] descs =
result[x].getDescriptions();
                     for (int i = 0; i < descs.length; i++) {
                         System.out.println("Experiemental Desc: "
+ result[x].getIdentifier() + " - " + descs[i].getText());
                     }
                 }
             }
             else
                 System.out.println("No Experiemental Description
Found!");
    sess.end() ;
  } catch (Exception ex) {
   ex.printStackTrace();
   fail("Got " + ex.getClass().getName() + ": " + ex.getMessage());
  }
 }
```

Figure 7.2 MAGE-OM test case

MAGE-OM Architecture

Domain Objects

The domain objects comprise the NCICB’s Java implementation of the MAGE-OM model, which is defined at http://www.omg.org/cgi-bin/doc?dtc/02-09-06. You can use instances of these classes to navigate through the caArray data and each class defines a toXML method.

An interface for each UML Class is defined underneath the `gov.nih.nci.mageom.domain` package. The implementation for each interface is defined in the corresponding `impl` package. For example, the interface for the MAGE-OM’s Experiment UML Class is defined in the `gov.nih.nci.mageom.domain.Experiment` package, and the default implementation of that class is defined in the `gov.nih.nci.mageom.domain.Experiment.impl` package.

SearchCriteria

The MAGE-OM client locates domain objects by constructing queries using SearchCriteria objects. There is one SearchCriteria class defined for each UML Class in the MAGE-OM model. For example, the `ExperimentSearchCriteria` interface is defined in the `gov.nih.nci.search.Experiment` package. Its implementation is defined in the `gov.nih.nci.search.Experiment.impl` package.

RMI

The MAGE-OM middleware uses RMI. The RMI server does not have to run in JBoss; it needs to run in its Java Virtual Machine(JVM). However, it uses the caArray SecurityEJB service that is hosted in JBoss and, therefore, needs to look that up through Java Naming and Directory Interface (JNDI).

The UML diagram in *Figure 7.3* shows this middleware architecture.

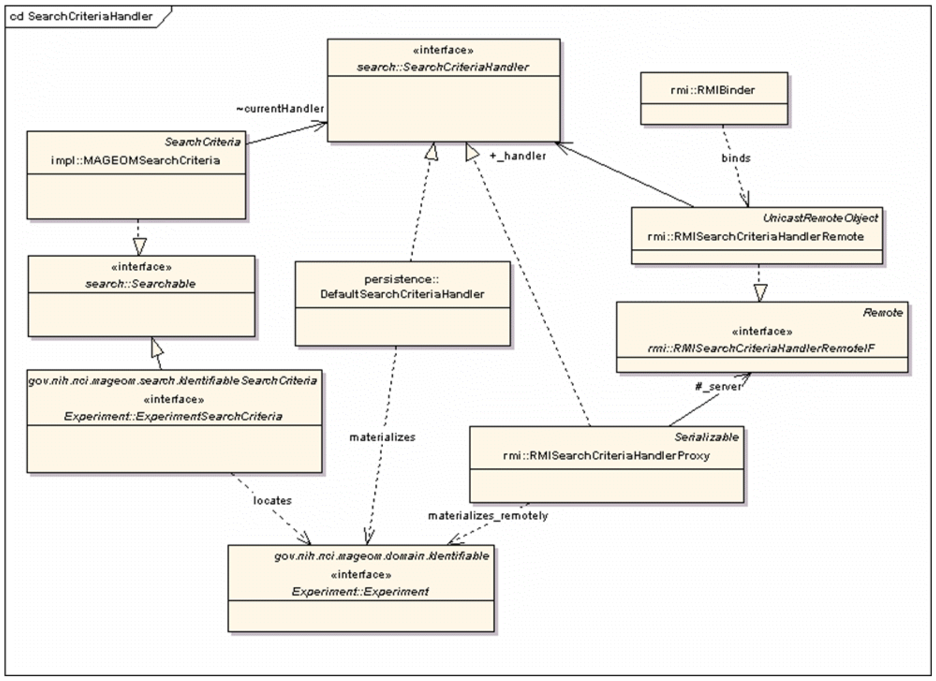
cd SearchCriteriaHandler
«interface»
search::SearchCriteriaHandler
rmi::RMIBinder
~currentHandler
SearchCriteria
impl::MAGEOMSearchCriteria
+_handler
binds
UnicastRemoteObject
rmi::RMISearchCriteriaHandlerRemote
«interface»
search::Searchable
persistence::
DefaultSearchCriteriaHandler
Remote
«interface»
rmi::RMISearchCriteriaHandlerRemoteIF
gov.nih.nci.mageom.search.IdentifiableSearchCriteria
«interface»
Experiment::ExperimentSearchCriteria
#_server
materializes
Serializable
rmi::RMISearchCriteriaHandlerProxy
locates
materializes_remotely
gov.nih.nci.mageom.domain.Identifiable
«interface»
Experiment::Experiment

Figure 7.3 Middleware architecture

SearchCriteria objects are used to locate domain objects. Therefore, instances of classes that implement `ExperimentSearchCriteria` locate instances of classes that implement Experiment. In this implementation of the MAGE-OM, all SearchCriteria implementations extend `MAGEOMSearchCriteria`. When the client executes a search, by call of the search method that is defined on the Searchable interface, `MAGEOMSearchCriteria` delegates the request to an instance of `SearchCriteriaHandler`.

Client code can be running locally (within the same JVM as the MAGE-OM server) or it can be running remotely (not within the same JVM as the MAGEOM server). In either case, queries are ultimately handled my some implementation of `SearchCriteriaHandler` that is running in the MAGE-OM server's JVM.

If the client is running remotely, then the SearchCriteria must be sent to the MAGE-OM server's JVM using RMI. `MAGEOMSearchCriteria` instantiates `RMISearchCriteriaHandlerProxy` which sends the SearchCriteria over RMI and is ultimately processed by `DefaultSearchCriteriaHandler`.

If the client is running locally, then RMI is not needed. In this case, `MAGEOMSearchCriteria` instantiates `DefaultSearchCriteriaHandler`, which materializes the selected domain objects.

MAGEOMSearchCriteria determines what implementation of SearchCriteriaHandler by checking the value of the following two system properties:

- DefaultSearchCriteriaHandlerClass - the classname of the local SearchCriteriaHandler; defaults to gov.nih.nci.common.persistence.DefaultSearchCriteriaHandler
- DefaultProxySearchCriteriaHandlerClass - the classname of the remote SearchCriteriaHandler; defaults to gov.nih.nci.common.remote.rmi.RMISearchCriteriaHandlerProxy

Upon instantiation, MAGEOMSearchCriteria first tries to instantiate a local SearchCriteriaHandler (i.e., an instance of the class defined by the value of DefaultSearchCriteriaHandler). If that fails, it tries to instantiate a remote SearchCriteriaHandler. The client can also set the SearchCriteriaHandler programmatically at any time.

RMISearchCriteriaProxy looks for the RMIServerURL property in the rmi-client.properties file, which must be on the client classpath. The value of this property is the RMI URL of the SearchCriteriaHandler.

The class that binds RMI server classes to the RMI registry is RMIBinder. This is the MAGE-OM server. It defines a main method and is started from the command line by the startmgrs.bat or startmgrs.sh script.

RMIBinder reads the file remote-objects.properties, which contains a mapping of RMI bind names to remote class names. RMIBinder iterates through these mappings, instantiating each remote class and binding it to the RMI registry using the specified name.

RMIBinder takes a number of command-line arguments that affect how the RMI registry is setup and how remote objects are bound. Table *7.2* provides how the values of these arguments are specified in startmgrs.bat or startmgrs.sh.

Argument	***Values***
rmipublichost	This is the hostname part of the RMI URL that clients must use when they look up a remote object. It also becomes the value of the java.rmi.server.hostname system property within the MAGE-OM server's JVM.
rmiserverhost	This is the hostname part of the RMI URL that is used to bind remote objects to the RMI registry. The rmipublichost and rmiserverhost values may have to be different to allow MAGE-OM clients to run outside of a firewall.
rmiserverport	This becomes the value of the java.rmi.registry.port system property.
rmiserverdataport	This becomes the value of the java.rmi.data.port system property.

Table 7.2 RMIBinder command-line arguments

MAGE-OM Security

The MAGE-OM API is currently read-only. Therefore, MAGE-OM clients cannot manipulate the persistent state of the caArray database. However, MAGE-OM requires that a security mechanism restrict read access to data in the caArray database based on a user's affiliation with caBIG consortia. To provide for this requirement, the concept of a secured session was introduced into the MAGE-OM design. You can obtain read access to caArray data that is restricted to particular consortia by creating a secured session using your username and password.

Client-side Security

The client must start a secure session by executing the following code:

```
SecureSession sess =
SecureSessionFactory.defaultSecureSession();
sess.start("johndoe", "secretpassword");
String sessionId = sess.getSessionId();
```

`SecureSession.start(String,String)` returns a `boolean` indicating whether you authenticated successfully. If so, you have access to data that is accessible to the consortia to which you belong. The session lasts until the client code calls `sess.end()`, the MAGE-OM RMI server exits or the default session length is exceeded.

The default life of a session is 24 hours that is defined in `gov.nih.nci.common.search.SecureSessionManager.DEFAULT_SECURE_SESSION_LENGTH`, which is in units of milliseconds. The default session length can be changed by updating the `MaxSecureSessionLengthMins` system property, which is in units of minutes, that is defined in the MAGE-OM server's JVM. The client can extend the session life by calling `SecureSession.extend()`, which extends the session by the default session length. The client can learn the time remaining in the session by calling `SecureSession.getTimeToLive()`, which returns the time remaining in milliseconds.

The value returned by `SecureSession.getSessionId()` is the client's session ID. This ID must be set on the SearchCriteria object that should retrieve restricted data. All MAGE-OM SearchCriteria objects implement `gov.nih.nci.common.search.SessionObject`, which declares the `setSessionId(String)` method.

To associate a query with an authenticated session, the client must call `SearchCriteria.setSessionId(String)`, passing in the value returned from `SecureSession.getSessionId()`.

For example:

```
  ArrayDesignSearchCriteria sc = new
ArrayDesignSearchCriteria();
  sc.setSessionId(sess.getSessionId());
```

Since the MAGE-OM objects use lazy-fetching by default, calls to get associated objects result in new queries. Those queries will be automatically associated with the current session. The implication of using lazy-fetching in this way is that it is possible that the session will expire before a lazy-fetch is executed. In such an event, no associated objects are retrieved.

MAGE-OM objects retrieved within a secure session are automatically associated with that session by the `gov.nih.nci.common.search.session.SecureSessionInitializer` aspect.

If the client does not associate a SearchCriteria object with a valid session ID, then only public data is retrieved.

Server-side Security

Query Result Filtering

MAGE-OM queries are handled by `gov.nih.nci.common.persistence.PersistenceManagerQueryHandler`, which delegates the actual query to OJB. The result of a successful query request is intercepted by the `gov.nih.nci.common.persistence.SearchIntercepter` aspect. This aspect then delegates to `gov.nih.nci.common.persistence.SecurityFilter`, which filters out objects to which the client does not have access. `SecurityFilter` actually delegates the filtering task to `gov.nih.nci.caarray.services.security.ejb.SecurityManager`.

The logic that `SecurityFilter` uses to filter objects is as follows (though the program flow is different):

```
If the client is not authenticated,
  Then return an empty result set.
Else,
  For each object in the original result set,
    If the object has no securedElementId,
      Then it is public.
    Else,
      Ask SecurityManager if the client has access to the object.
  If the object is public or the client has access to it,
```

When asking the `SecurityManager` if the client has access, `SecurityFilter` simply passes all objects and all roles to `SecurityManager.canUserAccessElements(SecuredElementItf[],String[])`. SecurityFilter constructs this array of roles as follows:

```
_defaultRoles = new String[9];
_defaultRoles[0] = SecurityConstants.CURATOR_ROLE;
_defaultRoles[1] = SecurityConstants.EXP_MANAGER_ROLE;
_defaultRoles[2] = SecurityConstants.BIO_MANAGER_ROLE;
_defaultRoles[3] = SecurityConstants.USER_ROLE;
```

```
  _defaultRoles[4] =
SecurityConstants.ARRAY_DESIGN_MANAGER_ROLE;
  _defaultRoles[5] = SecurityConstants.USER_MANAGER_ROLE;
  _defaultRoles[6] = SecurityConstants.PROTOCOL_MANAGER_ROLE;
  _defaultRoles[7] =
SecurityConstants.VOCABULARY_MANAGER_ROLE;
  _defaultRoles[8] = SecurityConstants.DEVELOPER_ROLE;
```

The JNDI lookup information for SecurityManager is located in `security-jndi.properties` file that must be located on the server classpath.

Secure Session Classes

The UML diagram in *Figure 7.4* shows the classes involved in managing secure sessions.

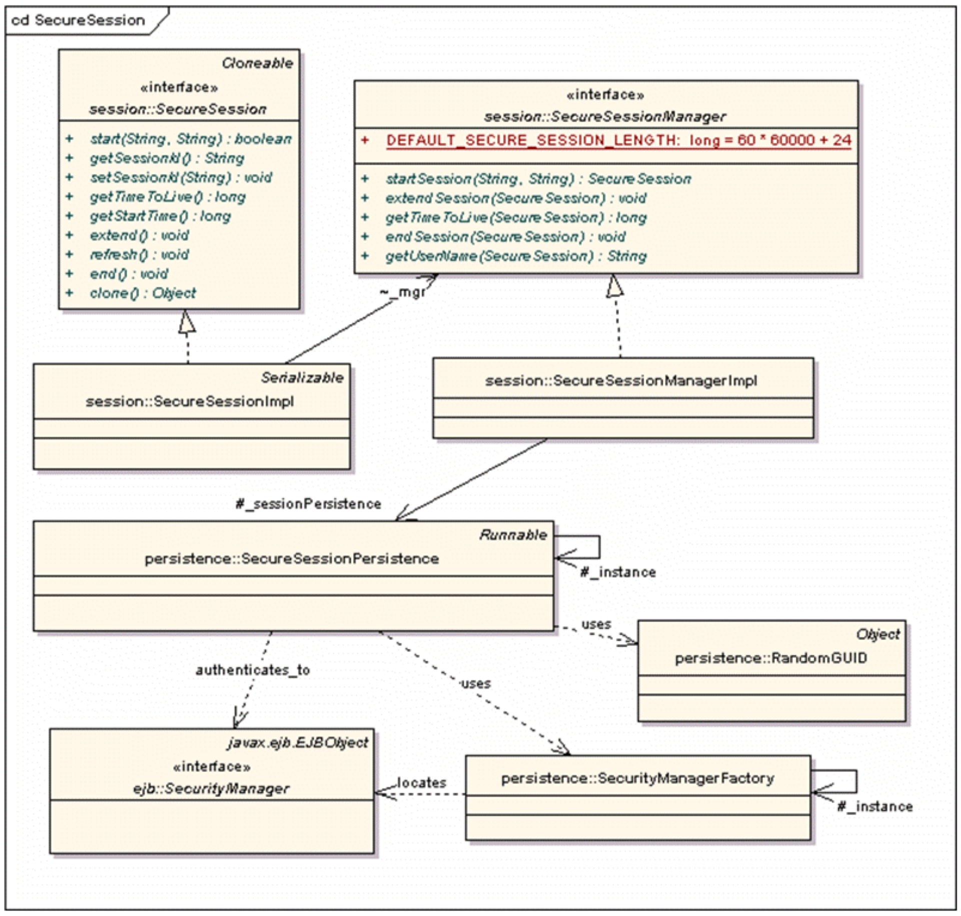
cd SecureSession
Cloneable
«interface»
session::SecureSession
+ start(String, String) : boolean
+ getSessionId() : String
+ setSessionId(String) : void
+ getTimeToLive() : long
+ getStartTime() : long
+ extend() : void
+ refresh() : void
+ end() : void
+ clone() : Object
«interface»
session::SecureSessionManager
+ DEFAULT_SECURE_SESSION_LENGTH: long = 60 * 60000 + 24
+ startSession(String, String) : SecureSession
+ extendSession(SecureSession) : void
+ getTimeToLive(SecureSession) : long
+ endSession(SecureSession) : void
+ getUserName(SecureSession) : String
~_mgr
Serializable
session::SecureSessionImpl
session::SecureSessionManagerImpl
#_sessionPersistence
Runnable
persistence::SecureSessionPersistence
#_instance
uses
Object
persistence::RandomGUID
authenticates_to
uses
javax.ejb.EJBObject
«interface»
ejb::SecurityManager
locates
persistence::SecurityManagerFactory
#_instance

Figure 7.4 Secure Session UML diagram

`SecureSessionPersistence` is the class that manages user authentication and session ID persistence. User authentication is handled within caArray, which provides the SecurityManager EJB. `SecureSessionPersistence` uses `SecurityManagerFactory` to locate (JNDI lookup) an instance of `SecurityManager`, `SecurityManagerFactory` uses the file `security-jndi.properties` to populate the InitialContex object that does the JNDI lookup. This file must be on the classpath. If it is not found, then default values are used.

`SecureSessionPersistence` calls `SecurityManager.isUserAuthenticated(String,String)` passing in the username and password, to verify that the user is authenticated. If so, then a new session ID is created. Session IDs are generated by `gov.nih.nci.common.persistence.RandomGUID` and maintained in memory by `gov.nih.nci.common.persistence.SessionPersistence.`

A session ID is generated in the following way:

1. Construct String from:
    a. IP address
    b. System time
    c. Result of `java.security.SecureRandom.nextLong()`
2. Run this String through a MD5 hash.
3. Format String as a GUID.

Directable

The interface `gov.nih.nci.common.search.Directable` represents a locally running client that can be directed to communicate with some remote server. The method, `direct(String`, takes the URL of some server location. In MAGE-OM, the important classes that implement this interface are:

- `gov.nih.nci.common.search.session.SecureSessionImpl`
- `gov.nih.nci.mageom.search.impl.MAGEOMSearchCriteria`

Since MAGE-OM uses RMI as its remote communication mechanism, the String arguments to the direct method are RMI URLs of the form `‘//hostname:port/bindname`. The default values used by `MAGEOMSearchCriteria` and `SecureSessionImpl` are located in `rmi-client.properties` file that is on the client classpath at runtime and is located in the source project in the `conf/rmi` directory.

Persistence

MAGE-OM uses OJB to perform ORM. The MAGE-OM objects are populated from data from the same database schema that the caArray application uses, and the OJB repository used by MAGE-OM is generated from the OJB repository used by caArray. This creates some difficulties because the MAGE-OM conforms strictly to the official MAGE UML, while the caArray database schema and ORM have been tuned for performance reasons. Furthermore, the OJB inheritance mechanism is not flexible enough (out of the box) to allow two different code bases to be populated from the same schema (this has to do with the `OJB_CONCRETE_CLASS` feature).

When the client invokes the search method on some MAGE-OM search criteria (as introduced in the *RMI* section on page 50), the `SearchCriteria` object is eventually passed to `DefaultSearchCriteriaHandler.` `DefaultSearchCriteriaHandler` uses `SC2query` to convert the SearchCriteria object into a `gov.nih.nci.common.query.Query` object. Then it delegates to `OJBQueryHandler`, which uses `Query2PBQ` to convert the `gov.nih.nci.common.query.Query` object into an `org.apache.ojb.broker.query.Query` object. The reason all of this conversion is taking place is that `OJBQueryHandler` was written in another project and works well with OJB and complex object models, like MAGE. The quickest way to integrate it into the MAGE-OM was to write `SC2Query` to do the conversion.

`Query2PBQ` compensates for some problems that OJB has with inheritance when one uses `ReportQuery` objects. You can see this in the `processAssociationCriterion` method.

CHAPTER 8 CAARRAY APIS

This chapter describes the caArray APIs.

Topics in this chapter include:

- *caArray EJB API* on this page
- *Directory Structure* on page 54
- *Testing caArray EJB APIs* on page 56

caArray EJB API

caArray EJB API provides transaction control, asynchronous processes, service location, common security and distributed capabilities for submission and retrieval of Microarray Experiments, MAGE-ML documents and its associated data files. The caArray EJB API provides the above functionality via the caArray presentation layer.

Notes:

- If you are troubleshooting, and your portal works with a deployed caArray in JBoss, then your JNDI server is fine, as the portal uses same service/EJBs to perform transactions. This assumes that the MAGE server is running on the same server as caArray. If not, you need to use the fully qualified `address.servername:1099.`
- The current caArray EJB API is being deprecated, and we recommend against its use by external applications. The next major release of caArray (2.0) will not support the current API; it will instead expose a re-engineered public API that will not be backward compatible.

Directory Structure

During installation, the `caArray_SourceCode_{version}.zip` file was unjarred and placed in the `{JBOSS_HOME}/caarray` directory. *Figure 8.1* displays an example directory structure.

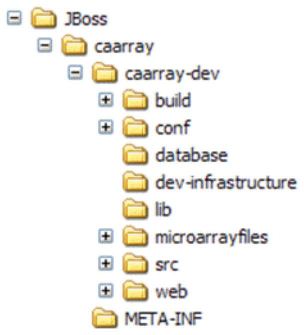

Figure 8.1 caArray directory structure

Table *8.1* lists a description of each directory in `{JBOSS_HOME}/caarray / caarray-dev`.

Directory	***Description***
`build`	Contains a `bin` directory that includes template, xml and property files (see *Table 9-2* for more information)
`conf`	Contains configuration files for `jboss`, `jndi`, `local`, `mapper`, `ojb`, `scm`, `security` and `sql` directories (see *Table 9-3* for more information)
`database`	Contains SQL files/scripts with updates to the schema. It should not be necessary to use these in most cases.
`dev-infrastructure`	Contains zip files for software products used
`lib`	Contains required JAR files needed to build and run the application
`microarrayfiles`	Contains a sample directory structure for files stored on the file system
`src`	Contains Java source code
`web`	Contains JSP files and configuration for the web tier

Table 8.1 caArray directory structure

Table *8.2* contains some of caArray properties files located in `{JBOSS_HOME}/caarray/ caarray-dev/build/bin`. When you deploy your system, the information from the `deploy.properties` file is used to create the other properties files listed in Table *8.2* from template files. The directory also contains the XML files to build the application including `build.xml`, `application.xml`, `jboss-service.xml.template` and `ear-build.xml`.

File Name	***Description***
`deploy.properties`	Contains installation specific properties that need to be populated before the caArray application can be deployed correctly. See the *caArray Local Installation Instructions* for a description of each parameter.
`caarray.properties`	Is the OJB repository file used for the MAGE-ML importer and exporter profiles. caARRAY application uses the default `repository_user_MAGE.xml`.
`db.properties`	Defines the database properties including the `LPG.driver, LPG.url, LPG.user, LPG.password` and `LPG.maximum`. It also contains the maximum records in the database (for example, gov.nih.nci.caarray.services.security.db.ROWSPAN=1000).
`ldap.properties`	Allows you to configure system parameters, paths and roles used for access control, specifically including settings for the LDAP server
`securityCommon.properties`	Is the property file for the security framework

Table 8.2 caArray properties files

Table *8.3* lists a description of each directory in `{JBOSS_HOME}/caarray` /`caarray-dev/conf`

Directory	***Description***
`jboss`	Contains JMS, JBoss and logging XML files including `jbossmq-destinations-service.xml`, `jboss-service.xml` `log4j.xml` and `login-config.xml`

`jndi`	Contains the `jndi.properties` file. Following is an example file:
	java.naming.factory.initial=org.jnp.interfaces.NamingContextFactory
	java.naming.factory.url.pkgs=org.jboss.naming:org.jnp.interfaces
	java.naming.provider.url=localhost

`mapper`	Contains `META-INF\mapper-repository.xml`, which includes DTO to MAGE-stk mappings, and `mapper-repository-template.xdt`.

Table 8.3 caArray conf directory

Directory	Description
ojb	Contains many XML files for the database schema and OJB including repository.dtd repository.xml repository_database.xml repository_database_local.xml repository_EXPORT_MAGE.xml repository_IMPORT_MAGE.xml repository_internal.xml, repository_MAGE.xml repository_MAGE_local.xml repository_user_CAARRAY.xml repository_user_EXPORT_MAGE.xml repository_user_IMPORT_MAGE.xml repository_user_MAGE.xml and spy.properties. This directory contains the files that you need to update your database schema. For example, use repository_database.xml file to configure the database connection. If a class-descriptor does not specify its own JDBC Connection, the connection specified in this file is used.
security	Contains secured-elements.xml which is used to define and mark security for fine grain MAGE-OM objects
sql	Contains SQL files

Table 8.3 caArray conf directory (Continued)

Testing caArray EJB APIs

Perform the following steps to call EJB methods.

1. Include the `jndi.properties` file on your classpath, which is included in caArray source distribution under `conf/jndi`.
2. Modify and run one of the JUnit test classes in the test packages (for example, `services.biomaterial.test.Get*` is a good test case).
3. The default `jndi.properties` file locates caArray running on localhost Edit the `jndi.properties` file for caArray production to be located at `caar-raydb.nci.nih.gov`. The port should be the same.

   Note: Since caArray production is behind a firewall, the RMI ports for MAGE-OM are opened up (the ports/IP on the server are not the same as they are used to call from outside firewall as the server is sitting behind a BIG IP and there is mapping of ports/IP done at that point). You would need to do the same for EJBs, for both JNDI as well as the underlying RMI protocol used by EJB. For now, it is best to test one of the JUnit test cases with local install. If declarative security has been enforced then you need to update `auth.conf` in the `jboss` directory.
4. Set the authentication service. If you are running within the container, the authentication service setting is automatically handled by the Tomcat servlet container embedded in JBoss when you run the webapp. If you're not running within the container, you must set the following to tell the Java client what authentication service to use:

```
-Djava.security.auth.login.config={JBOSS_HOME}/client/auth.conf
```

Note: If you are running the test case on the same server, then you can point right to your {JBOSS_HOME} directory. If you are not on a JBoss server, then you must copy the file then point to it.

CHAPTER 9

caAMEL SERVICE API

This chapter describes the caAMEL Service EJB API that can be used to programmatically validate and import MAGE-ML documents into caArray.

Topics in this chapter include:

- *Introduction to the caAMEL API* on this page
- *caAMEL Service API Code Example* on page 60
- *Considerations Before Coding for caAMEL Service API* on page 62

Introduction to the caAMEL API

Prior to using the API , the caAMEL application (`caamel.ear`) must be deployed and available. Generally caAMEL will be deployed to the same application server running caArray. See the *caAMEL Installation Guide* (downloadable from http://ncicb.nci.nih.gov/download/downloadcaarray.jsp) for details on deploying and configuring caAMEL. The caAMEL application includes the implementation that is invoked by the service client API.

The caAMEL Service may be invoked remotely by using the classes and interfaces that are included in `caamel_client.jar` which is included in the caAMEL 1.0 distribution. The contents of the `caamel_client.jar` are documented in the docs/javadocs subdirectory of the caAMEL distribution. The main elements are pictured in the class diagram shown in *Figure 9.1*.

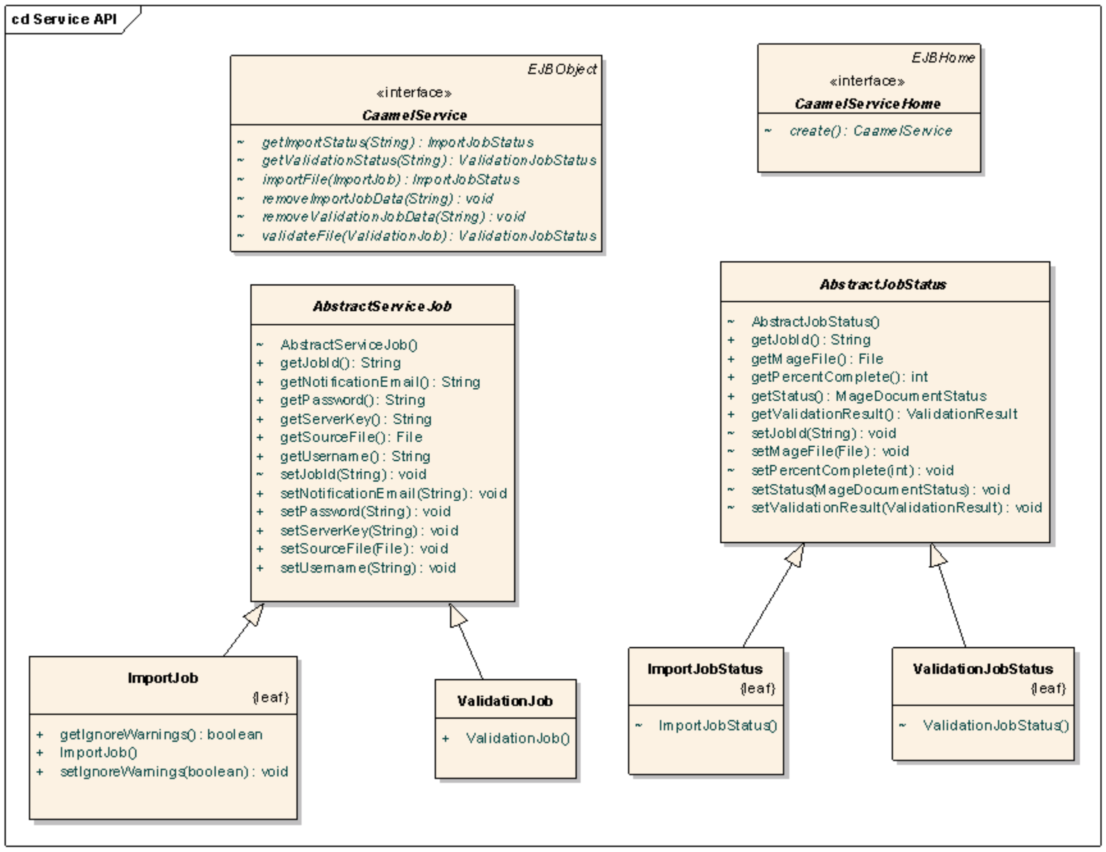
cd Service API
EJBObject
«interface»
CaamelService
~ getImportStatus(String) : ImportJobStatus
~ getValidationStatus(String) : ValidationJobStatus
~ importFile(ImportJob) : ImportJobStatus
~ removeImportJobData(String) : void
~ removeValidationJobData(String) : void
~ validateFile(ValidationJob) : ValidationJobStatus
EJBHome
«interface»
CaamelServiceHome
~ create() : CaamelService
AbstractServiceJob
~ AbstractServiceJob()
+ getJobId() : String
+ getNotificationEmail() : String
+ getPassword() : String
+ getServerKey() : String
+ getSourceFile() : File
+ getUsername() : String
~ setJobId(String) : void
+ setNotificationEmail(String) : void
+ setPassword(String) : void
+ setServerKey(String) : void
+ setSourceFile(File) : void
+ setUsername(String) : void
AbstractJobStatus
~ AbstractJobStatus()
+ getJobId() : String
+ getMageFile() : File
+ getPercentComplete() : int
+ getStatus() : MageDocumentStatus
+ getValidationResult() : ValidationResult
~ setJobId(String) : void
~ setMageFile(File) : void
~ setPercentComplete(int) : void
~ setStatus(MageDocumentStatus) : void
~ setValidationResult(ValidationResult) : void
ImportJob
{leaf}
+ getIgnoreWarnings() : boolean
+ ImportJob()
+ setIgnoreWarnings(boolean) : void
ValidationJob
+ ValidationJob()
ImportJobStatus
{leaf}
~ ImportJobStatus()
ValidationJobStatus
{leaf}
~ ValidationJobStatus()

Figure 9.1 cd Service API class diagram

The caAMEL Service is a stateless, remote session EJB, so the standard methods of looking up the remote home interface and creating the bean interface apply. The general workflow for using the API can be summarized as follows:

1. Copy the MAGE-ML files to a directory visible to the server running caAMEL.
2. Instantiate the CaamelService interface.
3. Create a new ImportJob or ValidationJob instance.
4. Submit the job instance to the interface.
5. Check the job status periodically until completion (note that these can be very long running jobs – import of very large MAGE-ML documents can take several hours)
6. Once the job has completed, remove the job data.

caAMEL Service API Code Example

The following code provides a simple example that shows submitting an import job, waiting for completion, and then writing to standard output either any validation errors or a success message. Note that this is a simple, illustrative example. There are additional document statuses that should be accounted for (e.g. validation warnings). Also, due to length of time that these jobs may take to complete, code would not typically wait in a loop for completion. (A more reasonable implementation might check job status at a user's request).

```
public void caamelServiceExample () {
  try {
    // Set the JNDI EJB lookup properties
    System.setProperty("java.naming.factory.initial",
        "org.jnp.interfaces.NamingContextFactory");
    System.setProperty("java.naming.factory.url.pkgs",
        "org.jboss.naming:org.jnp.interfaces");
    System.setProperty("java.naming.provider.url",
        "jnp://mycaamel.org:1099");

    // Get the Home interface
    final InitialContext context = new InitialContext();
    Object homeRef = context.lookup("ejb/CaamelService");
    CaamelServiceHome home = (CaamelServiceHome)
      PortableRemoteObject.narrow(homeRef, CaamelServiceHome.class);

    // Create the CaamelService interface
    CaamelService caamelService = home.create();

    // Create an import job
    ImportJob job = new ImportJob();
    job.setUsername("myusername");
    job.setPassword("mypassword");
    job.setServerKey("caarray");
    job.setSourceFile(new File("/usr/share/mage/upload/my_mage_file.xml"));

    // start the job
    ImportJobStatus jobStatus = caamelService.importFile(job);

    // wait for the job to complete
    while (MageDocumentStatus.IMPORTING.equals(jobStatus.getStatus())
        || MageDocumentStatus.VALIDATING.equals(jobStatus.getStatus())) {
      Thread.sleep(10000);
      jobStatus = caamelService.getImportStatus(jobStatus.getJobId());
    }

    // Print out the result
    if (MageDocumentStatus.IMPORTED.equals(jobStatus.getStatus())) {
      System.out.println("Import completed successfully.");
    } else if (MageDocumentStatus.INVALID.equals(jobStatus.getStatus())) {
      // MAGE-ML failed validation -- print out the validation messages
      ValidationResult result = jobStatus.getValidationResult();
      AbstractValidationMessage[] messages = result.getMessages();
      for (int i = 0; i < messages.length; i++) {
        System.out.println(
            messages[i].getMageElementInfo().getMageElementName()
```

```
                + ": "
                + messages[i].getDescription());
          }
        }

        // Remove the job from the system
        caamelService.removeImportJobData(jobStatus.getJobId());

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

Creating and running a validation-only job would look very similar to the code above, except that the job object created would be an instance of ValidationJob and the CaamelSession method invoked would be validateFile().

Considerations Before Coding for caAMEL Service API

Having covered the basics of using the caAMEL Service API, some additional considerations may be helpful to understand prior to developing code that employs the API. Reviewing the API's javadoc is also strongly recommended.

- The serverKey property of the ImportJob and ValidationJob indicates which caArray server caAMEL should use for validation and import. Generally, caAMEL will only be integrated with a single caArray server. The default server key created in the caAMEL installation is "caarray". If your instance of caAMEL is configured to integrate with multiple caArray instances, you may find the server keys at the bottom of the caamel.properties file on your caAMEL server. The server key is the prefix to the various caArray integration properties (e.g. `"caarray.name", "caarray.hostname", "caarray.url"`, etc.).
- It's possible to validate the internal structure of a MAGE-ML document without validating the data against a particular caArray server. To perform this more basic validation only, do not provide a serverKey, username, or password in the `ValidationJob` object.
- If the username or password provided in a job object is invalid, an `InvalidLoginException` will be thrown.
- If a duplicate job is submitted (that is, there is already a validation or import job in progress for a file), an `InProgressException` will be thrown.

APPENDIX

A

UML MODELING

The caArray team bases its software development primarily on UML. This chapter is designed to familiarize the reader who has not worked with UML with its background and notation. Topics in this chapter include:

- *UML Modeling* on this page
- *Use-case Documents and Diagrams* on page 64
- *Class Diagrams* on page 66
- *Relationships Between Classes* on page 67
- *Sequence Diagrams* on page 69

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://dn.codegear.com/article/31863.

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

Only a subset of the diagrams, those used in caArray development, is described in this chapter.

- Use-case diagrams
- Class diagrams
- Sequence diagrams

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

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

While many good development tools provide support for generating UML diagrams, the Enterprise Architect (EA) software is used by the caArray development team.

Use-case Documents and Diagrams

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

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.

Associate Labeled Extract with Hybridization file

Description:

This Use Case describes automatic creation of additional labeled Extract (and associated BioSample/BioSource) from a template Labeled Extract when annotating hybridization files.

Actors: caArray User.

Pre-Conditions:

- [2]User is logged into the caArray application portal and is annotating the upload of Hybridization Data with Labeled Extracts.

Basic Course:

1. Actor checks the Copy button for the Labeled Extract.
2. Actor Selects the (template) Labeled Extract to associate with the hybridization file.
3. System make a copy (by value) of the labeled extract as well as all the parent BioSamples as well as the BioSource. The copied BioMaterial are nabes based on the uploaded file name for easy identification.
4. System associates the copied labeled extract with the hybridization file/data.
5. Actor searches biomaterial based on the uploaded file name as a token for the file name.
6. Actor modifies the details of the biomaterial to reflect the actual biomaterial.

Alternative flows:

Flow A

1. The actor does not check the copy button with the labeled extract.
2. System associates the selected labeled extract with the uploaded hybridization.

Flow B

1. Actor searches biomaterial based on the uploaded file name as a token for the file name.
2. Actor modifies the details of the biomaterial to reflect the actual biomaterial.

Figure A.1 Use-case document

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

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 the OJB abstraction layer and DAO classes.

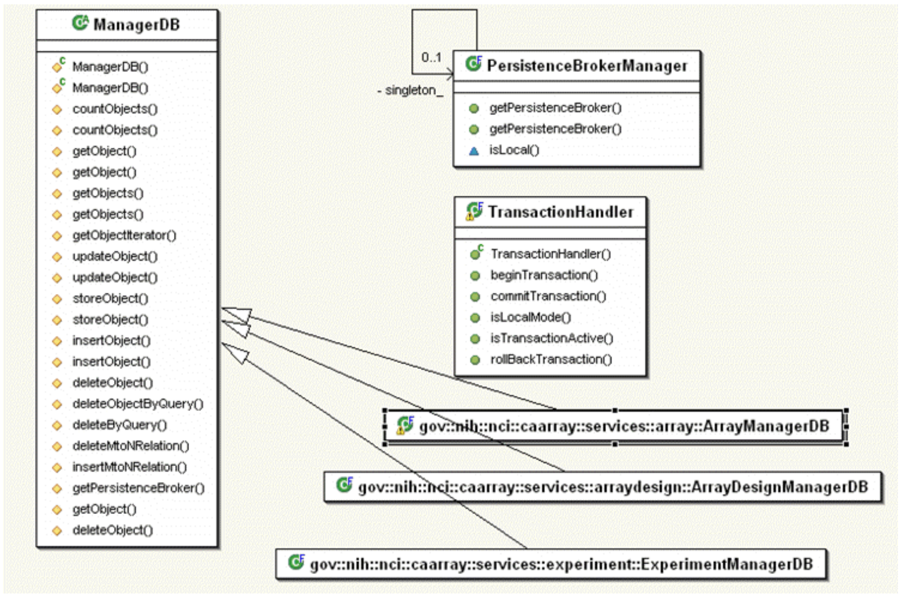

Figure A.2 OJB Abstraction Layer and DAO Classes

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

Figure A.3 (a) is a schematic for a UML class representation, the fundamental element of a class diagram. *Figure A.3* (b) is an example of how a simple class might be represented in this scheme. The enclosing box is divided into three sections: The topmost section provides the name of the class, and is often used as the identifier for the class; the middle section contains a list of attributes (structures) for the class. (The attribute in the class diagram maps into a column name in the data model and an attribute within the Java class.); the bottom section lists the object’s operations (methods). In the example below, (b) specifies the Gene class as having a single attribute called *sequence* and a single operation called *getSequence()*:

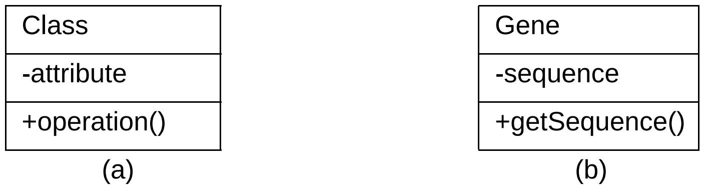

Figure A.3 (a) Schematic for a UML class (b) A simple class called Gene

Naming conventions are very important when creating class diagrams. caArray 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 classes represented 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 the diagram is intended to convey. Most UML design tools provide means for selectively suppressing either or both attributes and operation compartments of the class without removing the information from the underlying design model.

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

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

In *Figure B-3* for example, the Gene object’s *sequence* attribute is private and can only be accessed using the public *getSequence ()* method.

Relationships Between Classes

A quick glance at *Figure A.4* demonstrates relationships between some of the classes. Generally, the relationships occurring are of the following types: association, aggregation, generalization, and multiplicity, described as follows:

- **•Association** —The most primitive of these relationships is association, which represents the ability of one instance to send a message to another instance. Association is depicted by a simple solid line connecting the two classes.
- **Directionality**—UML relations can have directionality (sometimes called navigability), as in *Figure B-4*. Here, a *Gene* object is uniquely associated with a *Taxon* object, with an arrow denoting bi-directional navigability. Specifically, the Gene object has access to the Taxon object (i.e., there is a *getTaxon()* method), and the Taxon object has access to the Gene object. (There is a corresponding *getGeneCollection()* method.) Role names also display in *Figure B-4*, clarifying the nature of the association between the two classes. For example, a taxon

(rolename identified in *Figure A.4*) is a line item of each Gene object. The (+) indicates public accessibility.

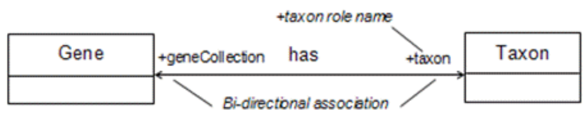

Figure A.4 one to one association

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

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

Table A.1 Commonly used multiplicities

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

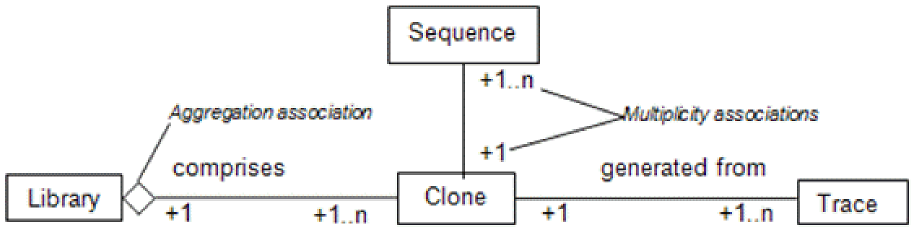

Figure A.5 Aggregation and multiplicity

Additionally, *Figure A.5* shows a more complex network of relations. This diagram indicates that:

a. One or more Sequences is associated with a Clone;

b. The Clone is contained in a Library, which comprises one or more Clones; and

c. The Clone may have one or more Traces.

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

- **Generalization** – Generalization is an inheritance link indicating that one class is a subclass of another. *Figure A.6* 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*.

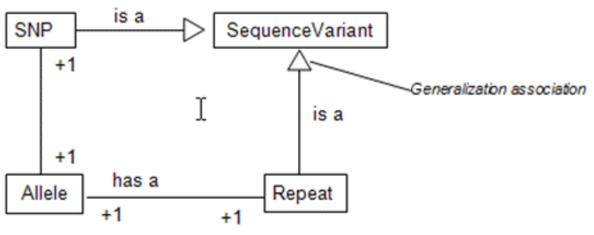

Figure A.6 Generalization relationship

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.

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.7 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.

The DTO sequence diagram (*Figure A.7*) includes the following:

- The application client sets user-entered values in the `ProtocolData` Transfer Object.
- The client application then invokes the EJB method to add protocol, sending the Transfer Object by value.

The EJB method then retrieves all user-entered values from the Transfer Object, and begins business processing.

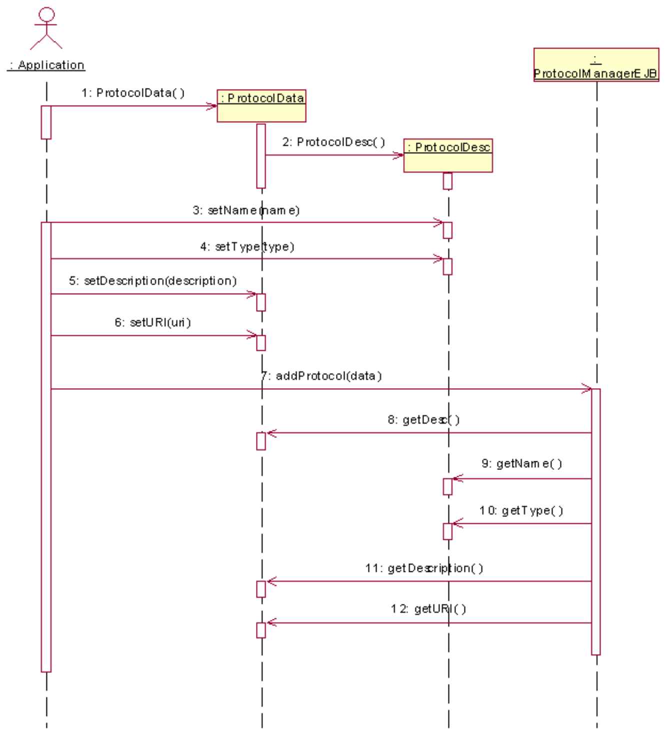

Figure A.7 DTO sequence diagram

APPENDIX
B

CAARRAY REFERENCES

Background Information

MAGE: http://mged.sourceforge.net

MIAME: http://www.mged.org/Workgroups/MIAME/miame.html

MAGE-ML: http://www.mged.org/Workgroups/MAGE/mage-ml.html

MGED Ontology: http://mged.sourceforge.net/ontologies/MGEDontology.php

MAGE-OM: http://www.mged.org/Workgroups/MAGE/mage-om.html

MAGE-OM model: http://www.omg.org/cgi-bin/doc?dtc/02-09-06

caArray Tools

Data Management tools: https://caarraydb.nci.nih.gov/caarray/index.jsp

Data Analysis tools: http://caarray.nci.nih.gov/caARRAY/data_analysis

caAMEL Material

caAMEL documentation: https://caarraydb.nci.nih.gov/caamel/

caBIG Material

caBIG: http://cabig.nci.nih.gov/

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

caCORE Material

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

Software Products

Struts: http://struts.apache.org/ and http://struts.apache.org/userGuide/index.html l

Tiles: http://struts.apache.org/userGuide/dev_tiles.html

OJB: http://db.apache.org/ojb

R: http://www.r-project.org/

Modeling Concepts

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/

UML tutorial: http://dn.codegear.com/article/31863

APPENDIX
C

CAARRAY GLOSSARY

Term	*Definition*
{jboss-home}	The base directory where JBoss is installed on the server
AOP	Aspect Oriented Programming
API	Application Programming Interface
caAMEL	caArray MAGE-ML Loader
caArray	cancer Array Informatics
CRUD	Create, Read, Update, Delete
caBIG	cancer Biomedical Informatics Grid
caBIO	Cancer Bioinformatics Infrastructure Objects
caDSR	Cancer Data Standards Repository
caCORE	cancer Common Ontologic Representation Environment
CDE	Common Data Elements
CGH	Comparative Genomic Hybridization
CSM	Common Security Model
CVS	Concurrent Versions System
DAO	Data Access Object
DDL	Data Definition Language
DTO	Data Transfer Object
EA	Enterprise Architect
Eclipse	Eclipse is a universal tool platform - an open extensible IDE http://www.eclipse.org/

Table C.1 Terms related to caArray or microarray technology

Term	**Definition**
EJB	Enterprise JavaBeans
GenePix	GenePix AutoProcessor (GPAP) is an automated and customizable application which is used to correct, filter and normalize raw microarray data and identify differentially expressed genes. Primary microarray data are captured using GenePix to generate a GenePix Results File (GPR). GPR files obtained from analyzing biological or technical replicates of the same treatment or time point can be processed using GPAP.
GPR	GenePix Results File
GUI	Graphical User Interface
JAR	Java Archive
JAAS	Java Authentication and Authorization Service
Javadoc	Tool for generating API documentation in HTML format from doc comments in source code (http://java.sun.com/j2se/javadoc/)
JDBC	Java Database Connectivity
JMS	Java Messaging Service
JNDI	Java Naming and Directory Interface
JSP	JavaServer Pages
JUnit	A simple framework to write repeatable tests (http://junit.sourceforge.net/)
JVM	Java Virtual Machine
LDAP	Lightweight Directory Access Protocol
MAGE	Microarray and Gene Expression
MAGE-ML	Microarray and Gene Expression Markup Language
MAGE-OM	Microarray Gene Expression - Object Model
MAGE-stk	MAGE Software Toolkit is a collection of Open Source packages that implement the MAGE Object Model in various programming languages (http://www.mged.org/Workgroups/MAGE/magestk.html)
MDA	Model Driven Architecture
MDB	Message-Driven Bean
MGED	Microarray Gene Expression Data (http://www.mged.org)
MIAME	Minimum Information About a Microarray Experiment

Table C.1 Terms related to caArray or microarray technology (Continued)

Term	Definition
MO	MGED Ontology - MGED has generated a set of guidelines called the MAGE-OM and has created the MO to provide the semantics for MIAME and MAGE. MO provides terms for the annotation of microarray experiments through classes, properties, and instances to describe the design, biological materials, and technical elements of a microarray experiment. MO also provides a framework to reference terms from external ontologies to take advantage of existing ontologies. In principle, MO can be extended to describe additional types of functional genomics experiments.
MVC2	Model View Controller 2
NCI	National Cancer Institute
NCICB	National Cancer Institute Center for Bioinformatics
OJB	ObjectRelationalBridge is an Object/Relational mapping tool that allows transparent persistence for Java Objects against relational databases (http://db.apache.org/ojb/)
OMG	Object Management Group
ORM	Object Relational Mapping
POJO	Plain Old Java objects
R	R is a language and environment for statistical computing and graphics
RDBMS	Relational Database Management System
RMI	Remote Method Invocation
RUP	Rational Unified Process
SQL	Structured Query Language
UCSF	University of California San Francisco
UI	User Interface
UML	Unified Modeling Language
URL	Uniform Resource Locators
WAR	Web Application Archive
WSDL	Web Services Description Language
XMI	XML Metadata Interchange (http://www.omg.org/technology/documents/formal/xmi.htm) - The main purpose of XMI is to enable easy interchange of metadata between modeling tools (based on the OMG-UML) and metadata repositories (OMG-MOF) in distributed heterogeneous environments
XML	Extensible Markup Language (http://www.w3.org/TR/REC-xml/) - XML is a subset of Standard Generalized Markup Language (SGML). Its goal is to enable generic SGML to be served, received, and processed on the Web in the way that is now possible with HTML. XML has been designed for ease of implementation and for interoperability with both SGML and HTML

Table C.1 Terms related to caArray or microarray technology (Continued)

Term	***Definition***
XP	Extreme Programming
Tiles	Tiles builds on the "include" feature provided by the JavaServer Pages specification to provide a full-featured, robust framework for assembling presentation pages from component parts. Each part ("Tile") can be reused as often as needed throughout your application. This reduces the amount of markup that needs to be maintained and makes it easier to change the look and feel of a website.

Table C.1 Terms related to caArray or microarray technology (Continued)

INDEX

A

access privileges 33
Affymetrix 15
Agilent 15
AOP, description 12
API
  caAMEL service interface 14
  caArray EJB 53
  example code, caAMEL 61
  introduction to caAMEL 59
  MAGE-OM 41
  MAGE-OM directory structure 42
  service API, caAMEL 60
architecture
  caArray 11
  design 11
  MAGE-OM 44
Array Design Manager access 34
Aspect Oriented Programming See AOP 12
association, described 67
authorization
  create 35
  create CRUD 35
  read 35
  update/delete 35

B

bin directory 54
BioMaterial Manager access 34
business layer 13
BusinessDelegate 22

C

caAMEL
  before coding for API 62
  elements of service API 60
  example serviceAPI code 61
  introduction to service API 59
  Service API description 14
caArray
  APIs 53
  architecture 11
  composite design pattern 18
  data sources 32
  database 5
  database schema 32
  database tier 32
  description 1
  design 17
  download site 39
  EJB APIs 53
  implementation 13
  introduction 1
  key features 9
  lazy materialization 31
  local database deployment 7
  overview 9
  portal 10
  portal description 39
  Presentation layer 13, 17
  programmatic Interfaces 14
  properties files 55
  roles 33
  Submission Portal 14
  submission portal 14
  Technical Guide description 1
  technologies used in 12
  testing EJB APIs 56
  User's Guide text conventions 2
caBIG
  compatibility guidelines 6
  compliance 6
  defined 6
  programmatic interface 7
caBIO, description 6
caCORE
  description 5
  plugging caArray into API 24
caDSR, description 6
Cancer Array Informatics Project See caArray 1

Cancer Bioinformatics Infrastructure Objects See caBIO 5
cancer Biomedical Informatics Grid See caBIG 5
cancer Common Ontologic Representation Environment See caCORE 5
Cancer Data Standards Repository See caDSR 5
capturing system requirements 64
CGH 40
class diagrams
  described 66, 67
  fundamental elements 66
  naming conventions 67
  private feature 67
  public feature 67
composite design pattern 18
conf directory 55
constants, for roles 33
conversion, DTO - MAGE-stk 26
converting, SearchCriteria 50
CRUD
  authorization 35
  description 35
curator access 34

D

data classes 19
Data Definition Language See DDL 32
data management tools 9
Data model 32
Data Owner access 34
Data Transfer Objects See DTOs 19
database
  DDL 32
  dump file 39
  schema 32
  security 32
  tier 32
DataConverter 28
DDL 32, 39
declarative security 35
delete authorization 35
deploying, caArray database 7
Desc classes 19
design challenges 14
directionality See navigability 67
directionality, described 67
directory
  bin 54
  conf 55
directory structure
  caArray 54
  MAGE-OM API 42
document
  conventions 2
  interfaces 14
downloading caArray files 39
DTO
  MAGE-stk conversion 26
  XML mapping 27
DTOs
  benefits 20
  description 12
  design 19
  levels 19
  retrieving data 25
  sequence diagram 20

E

EJB
  API description 14
  description 12
  layer 20
  security 35, 36
  SecurityManager 25
EJB APIs
  description 53
  interaction with DTOs 19
Enterprise JavaBeans See EJB 7
Enterprise Vocabulary Services See EVS 6
EVS, description 6
Experiment Manager access 34

G

GenePix 15, 40
generalization, inheritance link 69
generating, session ID 50

H

hybridization files, storage 32

I

Illumina 15
Imagene 15
implementation of caArray 13
import, MAGE-ML 14
initializing arrays 28
interfaces
  caArray 14
  document 14

J

J2EE 13
JAAS 36
JAR 54
Java Authentication and Authorization Service See JAAS 36
Java Database Connectivity See JDBC 29
Java documents 39
Java Messaging Service See JMS 12
Java Naming and Directory Interface See JNDI 45
Java Transaction API See JTA 21
Java Virtual Machine See JVM 45
JDBC 29, 30
JMS
- description 12
- purpose in caArray 13
- service components 21

JNDI 45
JSP 54
JSP tag library 18
JTA 21
JVM 45

L

lazy loading See lazy materialization 31
lazy materialization 31
LDAP, description 12
Lightweight Directory Access Protocol See LDAP 12

M

MAGE Manager 25
MAGE-ML 7, 9
- file storage 32
- import 14
- import/export description 14
- importing into caArray 59
- purpose in caArray 13

MAGE-OM 9, 44
- API description 41
- directable 50
- domain objects 44
- OJB repository 50
- production server 43
- queries 48
- RMI connection 43
- RMI ports 56
- search criteria 44
- security 47
- testing production server 43

MAGE-OM API 14
- annotation 32
- architecture 42
- directory structure 42
- download site 40
- implementation objects 41
- installation instructions reference 40
- MAGE-Compliant objects 41
- middleware 45

MAGE-stk
- business objects 25
- description 26
- DTO conversion 26
- purpose in caArray 13

ManagerDB 29
mapping DTO to XML 27
metadata
- description 6
- for interoperability 7

MGED 7, 9
MIAME 7, 9
Microarray and Gene Expression Markup Language See MAGE-ML 7
Microarray and Gene Expression Object Model See MAGE-OM 7
microarray files 39
Microarray Gene Expression Data See MGED 7
Model View Controller 2 See MVC2 17
multiplicity 68
MVC2 17

N

naming conventions, class diagrams 67
navigability See directionality 67
NCICB, description 5
null lists 28

O

Object Relational Mapping See ORM 12
ObjectRelationalBridge See OJB 12
OJB
- as ORM tool 28
- description 12
- in MAGE-stk 26
- performing ORM 50
- purpose in caArray 13
- repository in MAGE-OM 50

ORM
- description 12
- OJB performing for MAGE-OM 50
- tool for OJB 28
- use in MAGE-stk 13

P

performing ORM 50
PersistenceBrokerManager 30
presentation layer 17
private feature 67
problems, OJB inheritance 51
programmatic
- interface 7, 14
- security 35

properties files
- caarray.properties 32
- LDAP 25
- SecurityCommon.properties 25

protection element 37
Protocol Manager access 34
public feature 67
public group, description 34
public user, description 34

Q

queries, MAGE-OM 48

R

RDBMS, description 12
read authorization 35
reading materials 2
Relational Database Management System See RDBMS 12
Relationships in class diagrams
- aggregation 68
- association 67
- directionality 67
- generalization 69
- multiplicity 68

Remote Method Invocation See RMI 41
Resources, caArray 2
RMI
- connection information, production server 43
- description 41
- use as MAGE-OM middleware 45

role names, defined 67
roles in caArray 33
RUP 75

S

SearchCriteria 50
SearchCriteriaHandler 45
SecuredElementItf 37
security
- architecture 12
- declarative 33, 35
- EJB 35, 36
- filtering objects 48
- implementing 36
- MAGE-OM 47
- programmatic 33, 37
- secure session classes 49

security database 32
SecurityManager EJB 25
semantic interoperability 6
Sequence diagrams
- described 69
- DTOs 20
- example 69

ServiceLocator 21
SessionFaçade 23
SQL 39, 54
starting, secure session 47
Struts 12
- components 18
- framework 17

T

test cases, MAGE-OM 43
testing, caArray EJB APIs 56
text conventions in user guide 2
tiles in Struts framework 18
toXML 44
TransactionHandler 30
troubleshooting
- JBoss 53
- JNDI 53
- null 28
- OJB classes 31

U

UCSF Spot to GenePix utility 40
UML
- caBIO 6
- class diagrams 66
- introduction 63
- sequence diagrams 69
- tutorial 63
- types of diagrams 63
- use-case diagram 65
- use-case document 64

unauthenticated user access 34
Unified Modeling Language See
UML 6
update, authorization for 35
Use-case
- diagram 65
- document 64

user access 33
User Manager access 34

V

view classes 19
VocabManager 24
Vocabulary Manager access 34

X

XML 24