caArray 2.0

Technical Guide

NATIONAL® CANCER INSTITUTE

Center for Bioinformatics

October 31, 2007

CREDITS AND RESOURCES

caArray Development and Management Teams			
Development	***Quality Assurance***	***Documentation***	***Project and Product Management***
Eric Tavela[2]	Tom Boal[5]	Eric Tavela[2]	Mervi Heiskanen[1]
Bill Mason[2]	Ron Keene[5]	Brent Gendleman[2]	Anand Basu[1c]
Todd Parnell[2]	Xiaopeng Bian[1]	Todd Parnell[2]	Brent Gendleman[2]
Dan Kokotov[2]		Paul Duvall[4]	Juli Klemm[1]
Rashmi Srinivasa[2]		Levent Gurses[4]	
Scott Miller[2]		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] Stelligent	[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
Definitions and Acronyms 2
References 2
Organization of the Manual 3
Document Text Conventions 3

Chapter 1
Architectural Representation of caArray 5

Architectural Representation 5
Architectural Goals and Constraints 5

Chapter 2
Use Case View 7

caArray Use Cases 8

Chapter 3
Logical View 11

Overview 11
Architecturally Significant Design Elements 13
caarraydb 30
Use-Case Realizations 31

Chapter 4
Implementation View 33

Overview 33
Artifacts 34

Chapter 5
Deployment View 35

Appendix A
Glossary 37

Index 39

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 3
- *Document Text Conventions* on page 3

Introduction to caArray

The *caArray Technical Guide* describes the aspects of caArray's design that are considered to be architecturally significant; that is, those elements and behaviors that are most fundamental for guiding the construction of caArray and for understanding caArray as a whole. Stakeholders who require a technical understanding of caArray are encouraged to start by reading this document, then reviewing the caArray UML model [CAARRAY_UML], and then by reviewing the source code. Please note that all diagrams represented in this document are taken from the caArray UML model [CAARRAY_UML]; for more detail about the elements in these diagrams, please consult the source model.

~~remove or revise this paragraph? 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://array.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* provides a comprehensive architectural overview of the caArray system, using a number of different architectural views to depict different aspects of the system. It is intended to capture and convey the significant architectural decisions which have been made on the system.

remove or revise this paragraph? 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.

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

Definitions and Acronyms

- Move these to the glossary? Include them both places?
- **DAO –** Data Access Object
- **EJB –** Enterprise JavaBeans
- **J2EE –** Java 2 Enterprise Edition
- **Java SE –** Java Standard Edition
- **JDK –** Java Development Kit
- **JPA –** Java Persistence API
- **JSP –** JavaServer Pages
- **MAGE-OM –** Microarray Gene Expression Object Model
- **POJO –** Plain Old Java Object
- **RUP –** Rational Unified Process
- **UML –** Unified Modeling Language

~~References~~

~~Do you have URLs for these? Ordinarily we have a references appendix. Because there are so few, leave these here?~~

~~This list contains reading materials and resources that can be useful for familiarizing yourself with concepts contained within this guide.~~

- ~~[CAARRAY_UML] caArray UML Models //cm/caarray2/docs/analysis_and_design/models/caaray.eap~~
- ~~[CAARRAY_UCS] caArray Use Case Summary //cm/caarray2/docs/analysis_and_design/models/caaray.eap~~
- ~~[KRUCHTEN] Philippe Kruchten 1995, "The 4+1 view model of architecture," IEEE Software. 12(6), November 1995. //cm/caarray2/docs/analysis_and_design/references/architecture/Kruchten4+1.pdf~~
- ~~[RUP] The Rational Unified Process, Version 2003.06.13~~

Organization of the Manual

The *caArray Technical Guide* contains the following chapters:I'm leaving this as is until I know what the final chapters are:

- *Using the caArray Technical Guide*
- *caCORE and caBIG OverviewsChapter* caCORE and caBIG Overviews
- *Overview of caArrayChapter Overview of caArray*
- *caArray ArchitectureChapter caArray Architecture*
- *caArray DesignChapter caArray Design*
- *caArray Data Access SecurityChapter caArray Data Access Security*
- *caArray Download SiteChapter caArray Download Site*
- *MAGE-OM APIChapter MAGE-OM API*
- *caArray APIsChapter* caArray APIs
- *caAMEL Service APIChapter 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.
`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*

Table 1.1 caArray Guide Text Conventions

Convention	*Description*	*Example*
`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

ARCHITECTURAL REPRESENTATION OF CAARRAY

Architectural Representation

The caArray architecture is represented in the *caArray Technical Guide* and in the UML design models as a set of views of the system from different but complementary perspectives. These views are: add X refs

- The **Use-Case View** – Describes the functional requirements of the system.
- The **Logical View** – Describes the organization of the system design into subsystems, interfaces, and classes and how these elements collaborate to provide the functionality described in the use-case view.
- The **Process View** - Illustrates the process decomposition of the system, including the mapping of classes and subsystems on to processes and threads.
- The **Deployment View** – Describes how the processes are allocated to hardware and execution environments and the communication paths between hardware nodes.
- The **Implementation View** – Describes the software components that realize the elements from the logical view and the dependencies between these components.

This style of describing software architecture is the approach recommended by the Rational Unified Process and is based on Philippe Kruchten's work, "*The 4+1 view model of architecture*" [KRUCHTEN] and is refined in the Rational Unified Process [RUP].

Architectural Goals and Constraints

The following factors are key considerations beyond the functional requirements that

are guiding the design of caArray 2.0.

caBIG Silver Compliance

caArray must be implemented in such a way that it may be certified caBIG Silver compliant. While Silver compliance is the requirement, caArray will provide a grid interface in anticipation of a possible move to Gold level compliance when the criteria for Gold compliance are established.

.Remote API Usability

One of the major flaws in releases of caArray prior to 2.0 has been the requirement to use the MAGE-OM to access annotation and data. Navigation between key classes in the MAGE is inefficient, difficult to understand, and difficult t0 implement. The object API exposed by the new evolution of caArray must be designed can we replace this with “is designed”? to be easily-understandable and navigable by remote clients, whether they access the API via the grid or a Java programmatic interface.

High Performance Data Parsing, Storage and Retrieval

Given that data storage and retrieval is the principal functionality of caArray, array data parsing, storage, and retrieval performance is key to a successful design.

CHAPTER 2
USE CASE VIEW

The use cases represented in **Figure 1** below contain the functionality that have the greatest impact on the design of the caArray architecture. In brief, the use cases described below require implementation of mechanisms for security, validation, file management, data storage and retrieval, and API design. Brief descriptions of each of these use cases are provided below as extracted from the model. For information on the complete use-case model see the caArray Use-Case Summary

[CAARRAY_UCS].Can we add the use case summary in this chapter? I think that's preferable than referring them to an outside document.

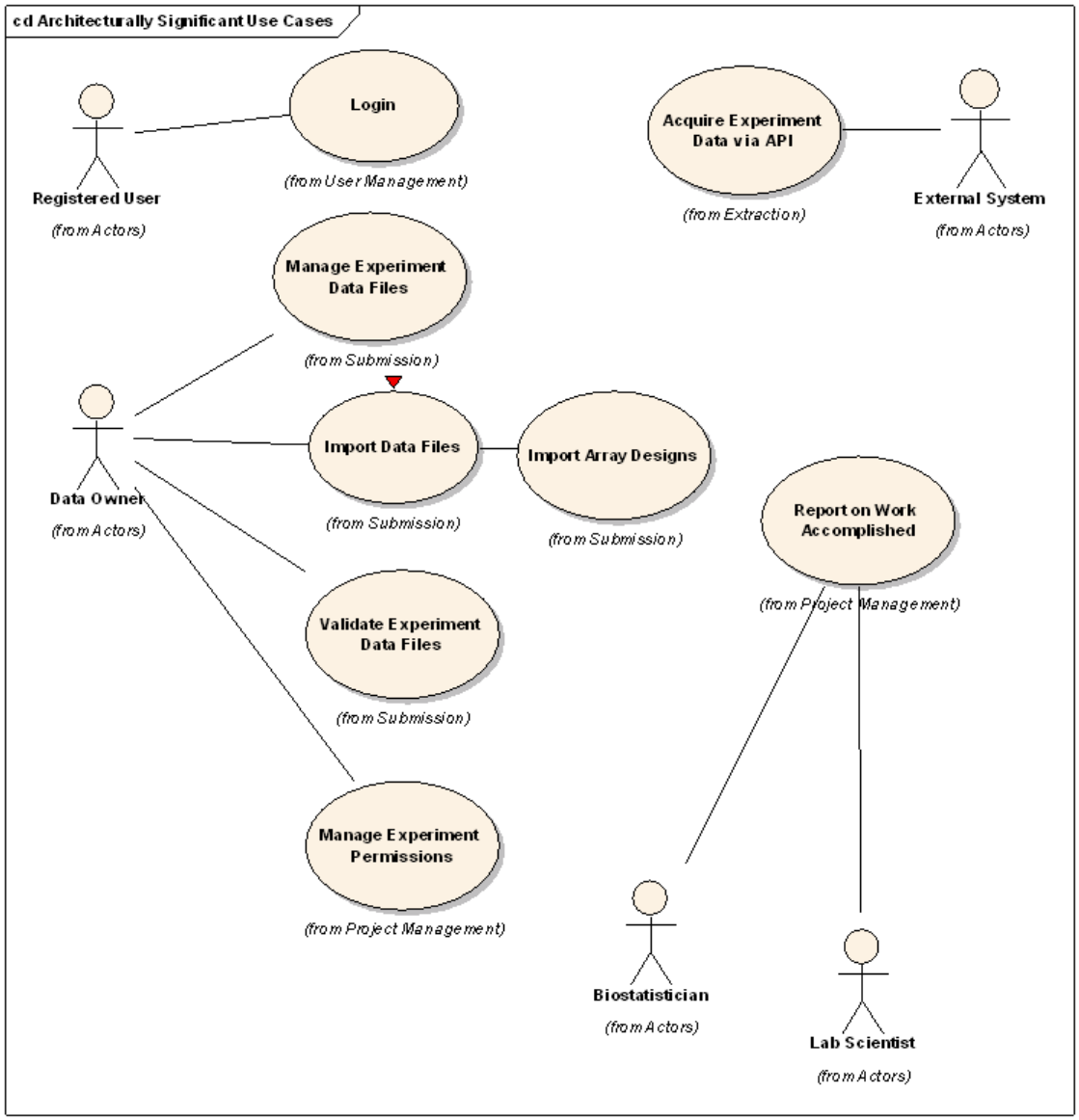

Figure 2.1 Use case summary

caArray Use Cases

Login

Initiated by any Registered User, the login use case allows for the validation of the authenticity and authority of the given user either against a networked (LDAP) set of users or a local set (database). As a result of a successful login, the registered user is presented with their home space and the set of operations they have been granted privileges to perform.

Manage Experiment Data Files

Initiated by a Data Owner, this use case enables the uploading of annotation and array content collectively and independently into a caArray project and then provides the ability to validate, import and/or delete the uploaded files and records each action taken. Due to the large file size of array data and to a lesser extent – the annotation, the transfer of the file from the client to the server may take minutes or even hours to complete; so the ability to offer to run the upload in the background (allowing the user to perform other functions inside caArray) is desirable.

Validate Experiment Data Files

Initiated by a Data Owner, this use case allows for the validation of file structure and content for annotation or array data, with a future intent to import the data into the project. The content validation is not to determine the accuracy of the information from a scientific viewpoint; rather it is purely to ensure that the data files loaded are sound for importing into the system. This use case will also be invoked when a Data Owner chooses to import non-validated data.

Import Data Files

Initiated by a Data Owner, this use case enables the import of annotation and array data from previously-uploaded files.

Import Array Designs

Initiated by a Curator, this use case allows for the uploading of array designs on an as needed basis. A significant number of array designs will come pre-loaded from the NCICB but when necessary, a particular institution may need to upload array designs that are dated (when pushing in legacy data) or be in front of the curve (prior to official support of new array design formats).

Manage Experiment Permissions

Initiated by a Data Owner, this use case allows for the promotion of a project's visibility. This action can apply to an entire project or to specific samples within the project. The basic visibility states of the project are: restricted, institution, group, public and collaborator. There is no restriction on changing a project's visibility, though it is not advisable once a publication has been published against the project.

Acquire Experiment Data via API

Initiated by an External System - though primarily by caBIG Analysis Services - this use case allows for the extraction of project data that is public and protected data is the appropriate authorization is used. This is not intended to be executed through a user interface, rather this will be accomplished through an API and will likely invoke other use cases (particularly search) to find and then acquire the data desired to be pulled into the other application

Chapter 3
Logical View

Overview

The design model (from which the logical view is taken) is the most significant model, requiring the most effort and containing the majority of the content. Accordingly, the description of the logical view of caArray's architecture will receive the most attention here. This chapter of the *caArray Technical Guide* first describes the structural hierarchy of the system in layers, packages, and subsystems and then describes how these elements collaborate to provide the most architecturally significant functionality. *Figure 3.1* illustrates the top-level structural organization of caArray. The major dependencies between subsystems are represented as well, though it should be noted that some supporting dependencies have been elided to enhance readability of the diagram.

The only subsystems that are accessible to external systems are the subsystems represented in the Grid API layer and the Remote Java API layer. All subsystems implemented in the Application Logic and Business Logic layer are internal to the application and do not expose remote interfaces. The User Interface layer is accessible to web clients via HTTPS.

caArray is implemented as a J2EE 1.4 application built on top of Java SE 5 (JDK version 1.5.0_10) employing the following core J2EE technologies:

- SP 2.0
- Servlet 2.4
- JMS 1.1
- EJB 3.0.

Only EJB session and message-driven beans will be employed; persistence is being managed directly with Hibernate 3.2 rather than EJB's persistence API (JPA). JPA provides no significant advantages over Hibernate at this point, and Hibernate provides additional extended functionality not included in JPA.

Clients of caArray can be characterized as either web UI clients or API clients.

Each of the subsystems shown in *Figure 3.1* is described briefly following the diagram. The functionality of each of these subsystems is described in the following section, *Architecturally Significant Design Elements*, and the context of their use is given in *caarraydb* on page 30 This 5.3 section (caarrayu db) is one very brief paragraph. Are you sure this is the correct reference? which documents the use-case realizations that employ these subsystems.

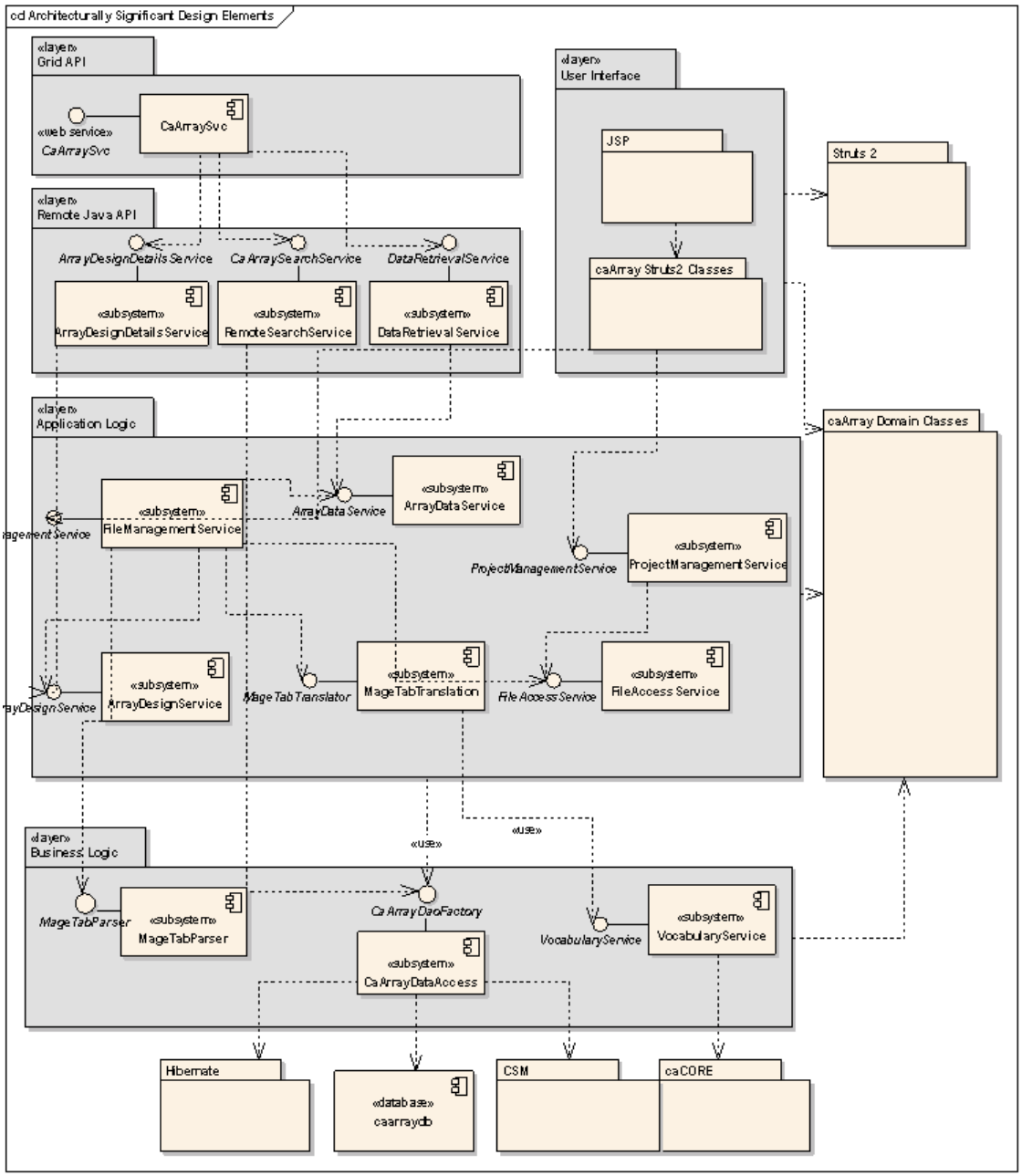

Figure 3.1 Architecturally Significant Design Elements

Architecturally Significant Design Elements

User Interface Layer

The caArray user interface is accessed as a standard web application via HTTPS. It is implemented as a J2EE web application employing Struts 2 as the Model-View-Controller implementation. This layer provides presentation, navigation and validation functionality only. Validation logic at this level is limited to standard form based validation (for example, checking for appropriate field formats) and is implemented using Strut2 2 validation. All application logic is implemented in the lower layers of caArray. The pages presented to the web client use HTML and JavaScript only; no applets or other client-side component technologies are used.

The User Interface layer also includes the login authentication class, CaArrayLoginModule. This class is used to integrate CSM authentication into the J2EE standard security model and allows for both database and LDAP based authentication. This class and its relationship to authentication classes from CSM and the Java security API are shown in *Figure 3.2*.

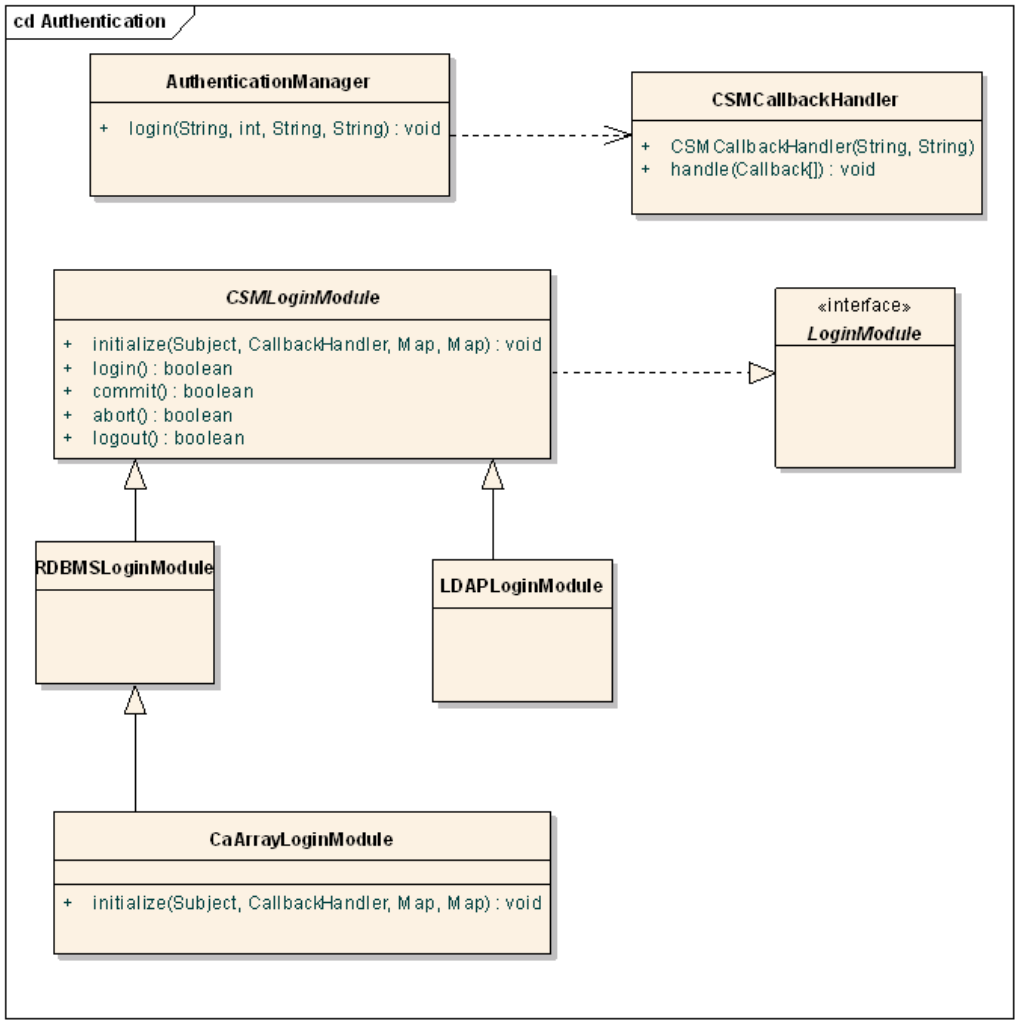

Figure 3.2 Authentication classes

Grid API

The caArray Grid API is a Grid 1.1 compliant data service with several analytical services. The service connects, via JNDI and RMI to a running instance of the caArray Remote Java API. The service connects to the web app at startup and uses the remote EJB API to service all requests received from the grid.

The grid service provides both the standard data query(CQLQuery) method and several analytic services. To retrieve all data stored in caArray, client applications must use a combination of CQL queries (the data service) and analytic service methods. All data stored in caArray is available when utilizing the data service in conjunction with the analytic services. This was a deliberate design choice based upon the team's experience with caArray 1.0.

To perform CQL searches, the service uses the API method `List<AbstractCaAraryObject> search(CQLQuery)` exposed by the CaArraySearchService EJB. After passing the CQLQuery to the EJB API, additional transformations are applied to generate a CQLQueryResults object for the grid client.

The EJB search API performs the bulk of the work for grid clients. The search method accepts the CQLQuery object and returns matching objects from the domain model, ignoring any query modifiers in the original CQLQuery. caArray uses the CQL2HQL class provided by the caCORE SDK, which is immediately runnable in Hibernate.

The grid service receives the list of matching domain objects from the search API and transforms those results into the CQLQueryResults expected by the grid client. To assist in this translation, caArray utilizes the CQLResultsCreationUtil from the SDK. Depending on query modifiers, the system either (1) translates whole objects, (2) translates [unique] specific properties, or (3) returns the count of objects in the list.

The Introduce generated components include all of the classes in *Figure 3.3*, with the exception of CaArraySvcImpl and provide the standard marshalling and query functionality of a standard caGrid data service. Delegation to the Java Remote API is handled

by the CaArraySvcImpl class, which wraps access to the EJB remote session beans that expose array annotation and data retrieval functionality.

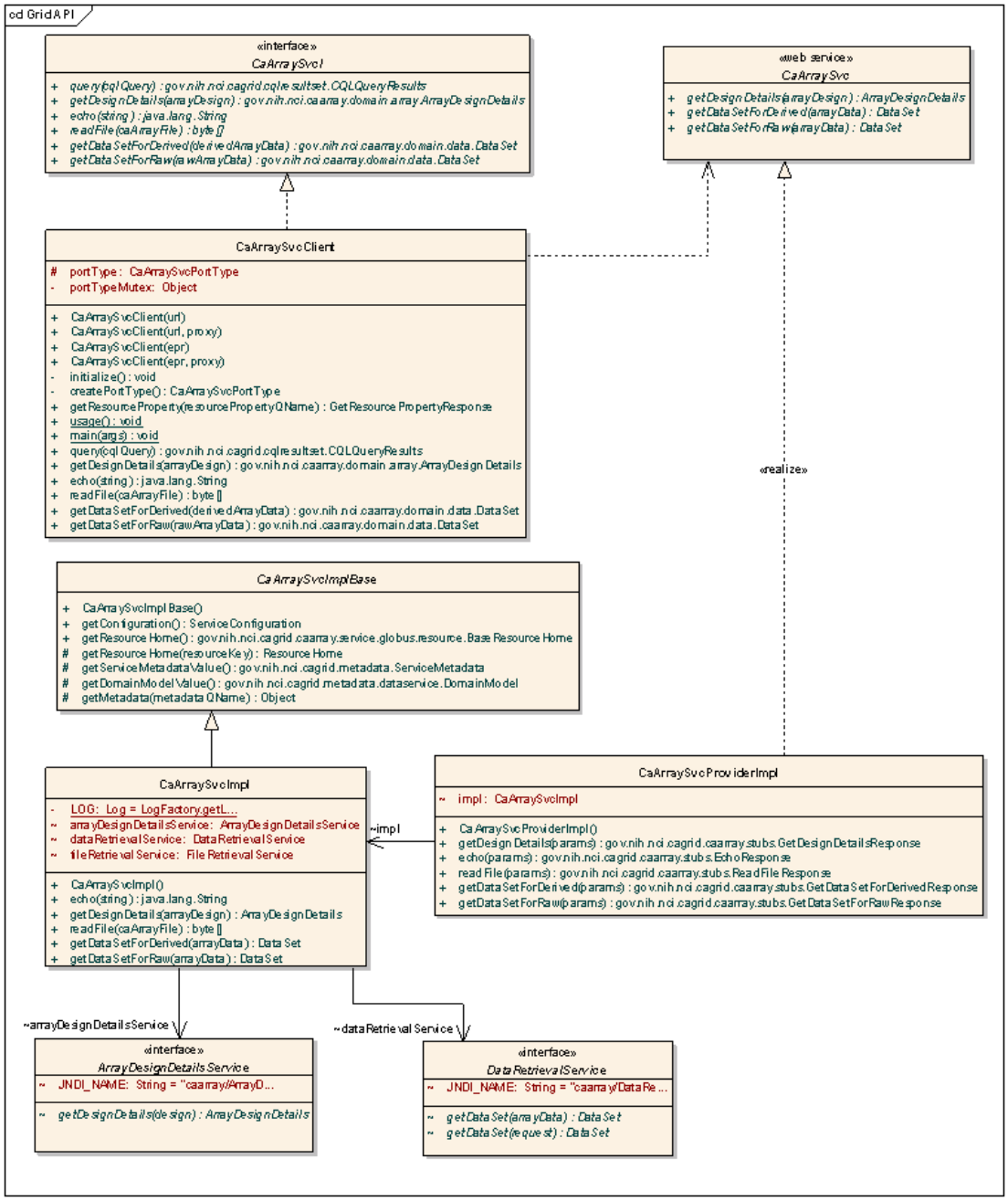

Figure 3.3 caGrid API

Remote Java API

The Remote Java API is implemented as a façade (the CaArrayServer class) representing a connection to caArray and a set of several stateless session EJBs with remote visibility. Clients instantiate a CaArrayServer instance, call the connect method and can then access the session EJB interfaces through accessor methods exposed by the CaArrayServer. These EJBs provide simplified, efficient access to caArray entities and data. Special consideration was given to the DataRetrievalService API to enable

clients to retrieve only the data they require. Clients may select data for specific QuantitationTypes, Hybridizations, and AbstractDesignElements by configuring a DataRetrievalRequest object and passing it as an argument to the getDataSet() method. The remote interfaces and their exposed operations are shown in the class diagram provided in *Figure 3.4*.

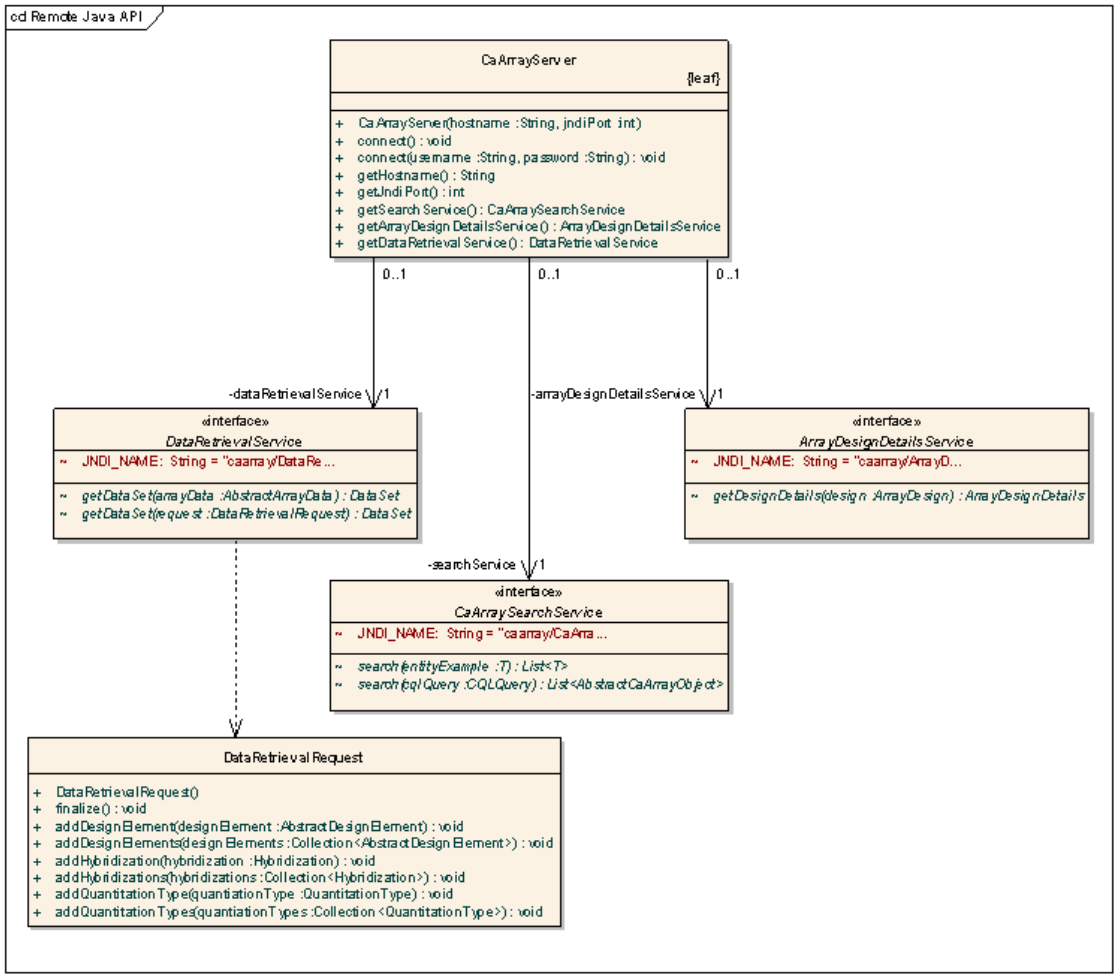

Figure 3.4 caArray Remote Java API

caArray Domain Classes

This section describes the classes used to model the microarray experiment and data that caArray is designed to manage. Since these classes are employed by all of the caArray subsystems and also must be understood by remote caArray clients (both Java RMI and caGrid) classes that represent important data constructs will be given detailed description here.

The underlying business object model is implemented as a set of POJOs that model the domain of microarray experiments and data. Whereas earlier versions of caArray used MAGE-OM 1.1 as the basis for the underlying object and data model, the new caArray implementation will be based on a completely revised, simplified object model. Although MAGE-OM is a published standard, there are significant disadvantages in using it as an underlying object model; it is complicated to understand, its complexity makes data storage inefficient, its structure does not permit useful object graph navigations, and many common relationships can not be stored when complete experiment

annotation is not available. For these reasons we have chosen to produce a new, simplified object model for domain data representation. caCORE was used for the initial generation of these POJOs.

The domain classes are principally designed to support the entity model described by the MAGE-TAB 1.0 specification. The underlying object model described by MAGE-TAB is considerably more understandable than MAGE-OM while still providing a complete enough model to support MIAME compliance. The most central entities in the domain model are pictured in *Figure 3.5*.

Special attention in the domain class design was given to representing array annotation and harmonizing this annotation with caBIO's domain model. caBIO provides a complete, peer-reviewed object model describing array reporter annotation and thus was imported directly into the caArray domain object model. While the caBIO classes provide full annotation details, they do not describe important relationships between array design elements (for example, from features to reporters) and therefore we've introduced a set of classes shown in *Figure 3.6* that record the physical layout of an array design and the mapping of physical probes (feature level) to logical probes (gene level). These physical and abstract probes are then related to the reporter annotation recorded in the caBIO annotation classes. This is shown in *Figure 3.6* by the relationship from `AbstractProbe` to the caBIO annotation class `ArrayReporter`. The diagram included as *Figure 3.7* shows the array annotation classes imported from the caBIO object model.

As has been noted earlier, array data needs to be represented in way that allows for efficient storage and transport when required by remote clients. The classes used to represent array data are shown in *Figure 3.8*. caArray is designed to represent array data at two levels:

- The `AbstractArrayData` hierarchy represents individual data files that have been imported into caArray, describing their type and relationships to hybridizations. These are high level representations that do not contain the actual data values.
- The `DataSet` class and the classes it is related to by composition (`HybridizationData` and the `AbstractDataColumn` hierarchy). These classes ultimately contain the array data values, specifically as arrays of primitive or String values within the `AbstractDataColumn` subclasses.

The DataSet classes are used both to persist the data contained in array data files and as a container for custom data sets requested by clients. As an example, a given Affymetrix CEL file imported into the system will have a single persistent DataSet containing a single persistent HybridizationData instance that contains several AbstractDataColumn instances (IntegerColumns for CELX and CELY, FloatColumn for CELIntensity, etc.). If a remote API client requests the data for all CEL files within an experiment, a transient, compound DataSet is created that contains multiple HybridizationDatas where each HybridizationData is retrieved from persistent storage.

A columnar approach to data representation allows for efficient retrieval and storage when compared with a row-based representation. This represents a significant change from caArray 1.x where an entire BioDataCube must be retrieved in its entirety allowing for a significant reduction of network transfer overhead. This columnar approach is preferable for two reasons:

1. Array data files typically contain relatively few columns but a large number of rows, typically in the tens of thousands or larger. When returning data to remote

clients, it is far more efficient to serialize a large array of primitives when compared to returning a large object graph.

2. Clients typically require only a small subset of the columns represented by an array data file, so organizing data by column allows for much more efficient custom DataSet assembly. Clients may indicate which columns to select by specifying QuantitationTypes to retrieve. The semantics of the various QuantitationTypes will be registered in caDSR in order to make them meaningful and comparable to clients that don't have advance knowledge of the context of specific QuantitationTypes.

In addition to efficient storage and transfer, this approach is also intended to meet the needs of caB2B and other tools that need everything navigable in the model (i.e. require the domain model semantics -- aren't aware of the data retrieval API). Making the columns themselves persistent with their data will allow these clients to navigate to the raw data values while we still retain an efficient mechanism for storage and retrieval (the columns' compressed, serialized value arrays are transparently expanded on request).

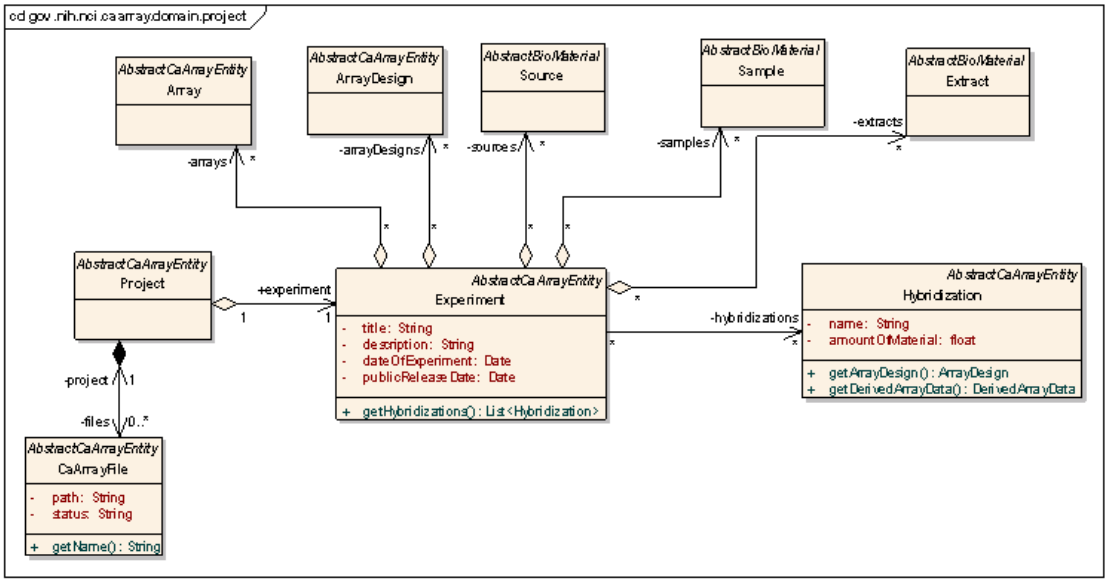

Figure 3.5 Experiment Overview

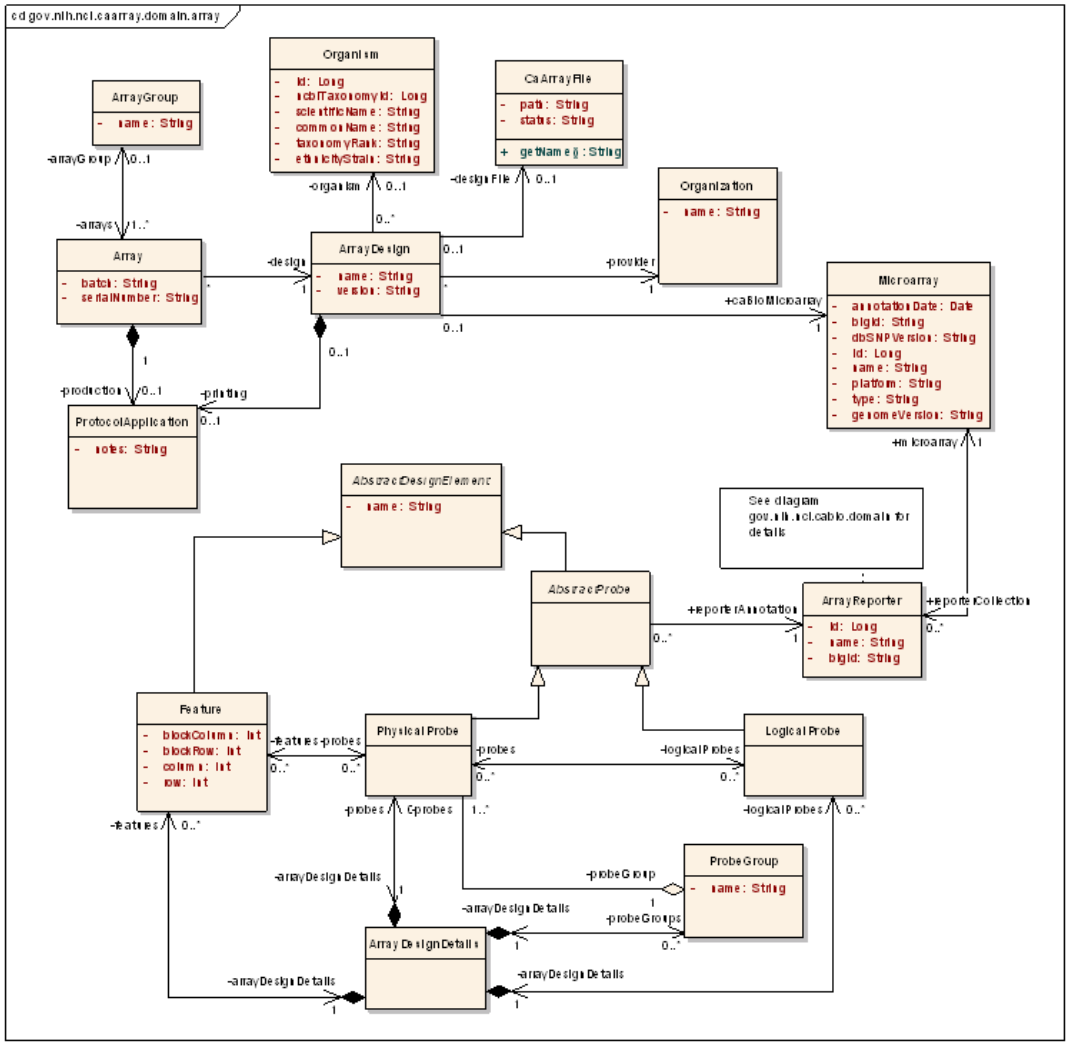

Figure 3.6 caArray Array Design Classes

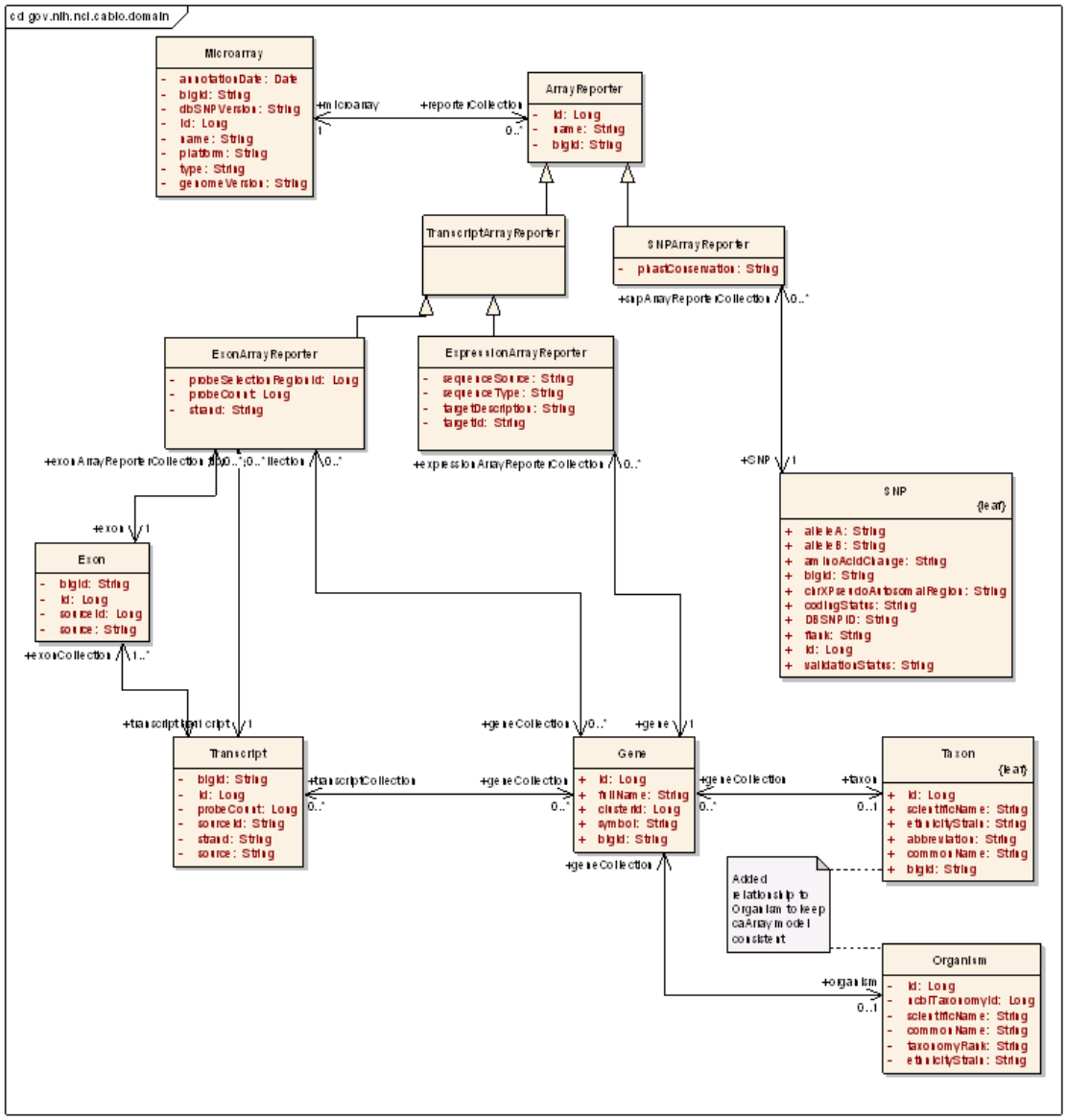

Figure 3.7 caBIO Array Reporter Annotation Object Model

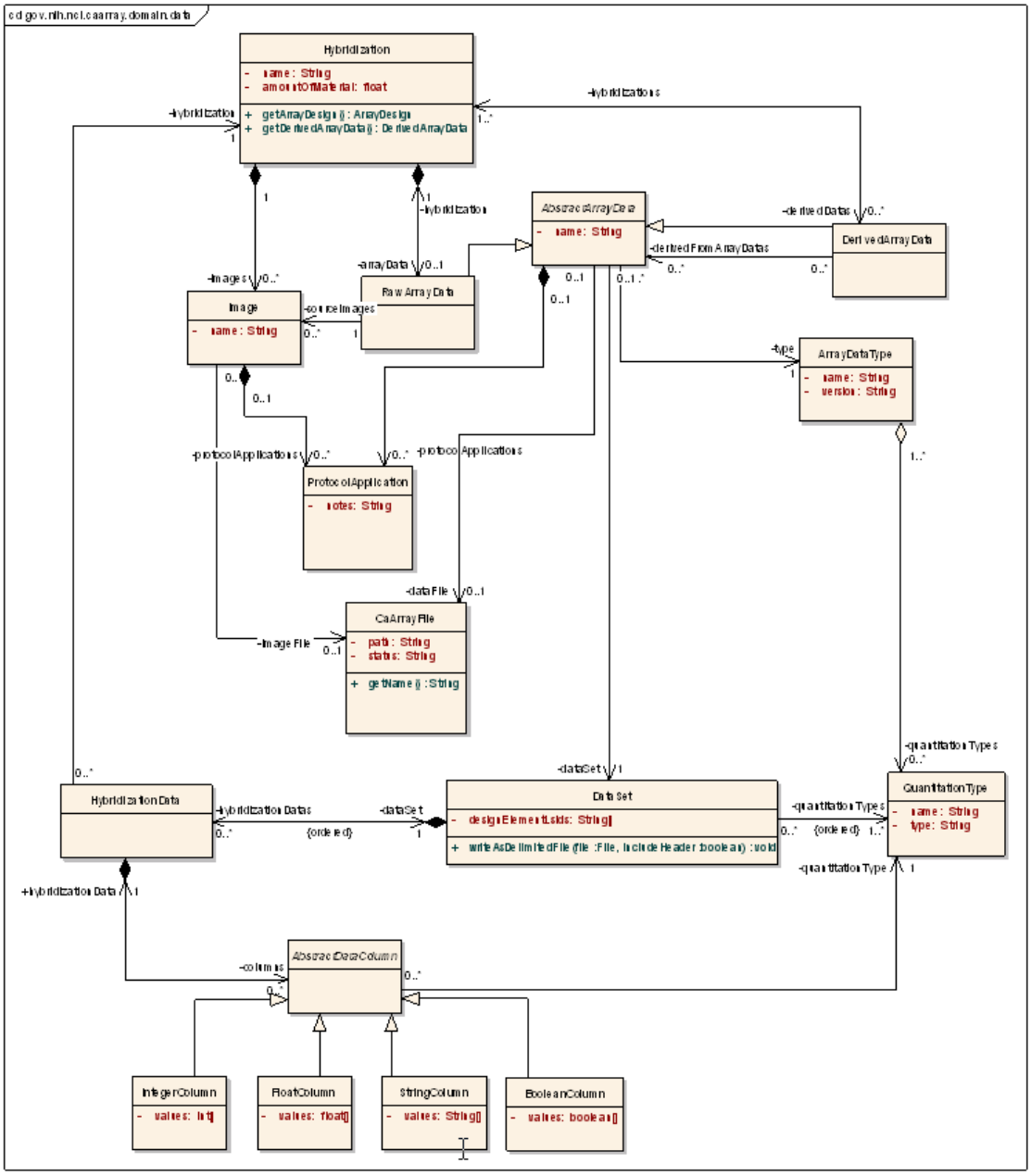

Figure 3.8 Array Data Classes

ProjectManagementService

The ProjectManagementService subsystem is implemented as a façade to allow the user interface to create and retrieve experiments. The implementation of this sub-system delegates directly to the CaArrayDataAccess service for entity management

and to the FileAccessService for file management. The subsystem contents are shown in *Figure 3.9*.

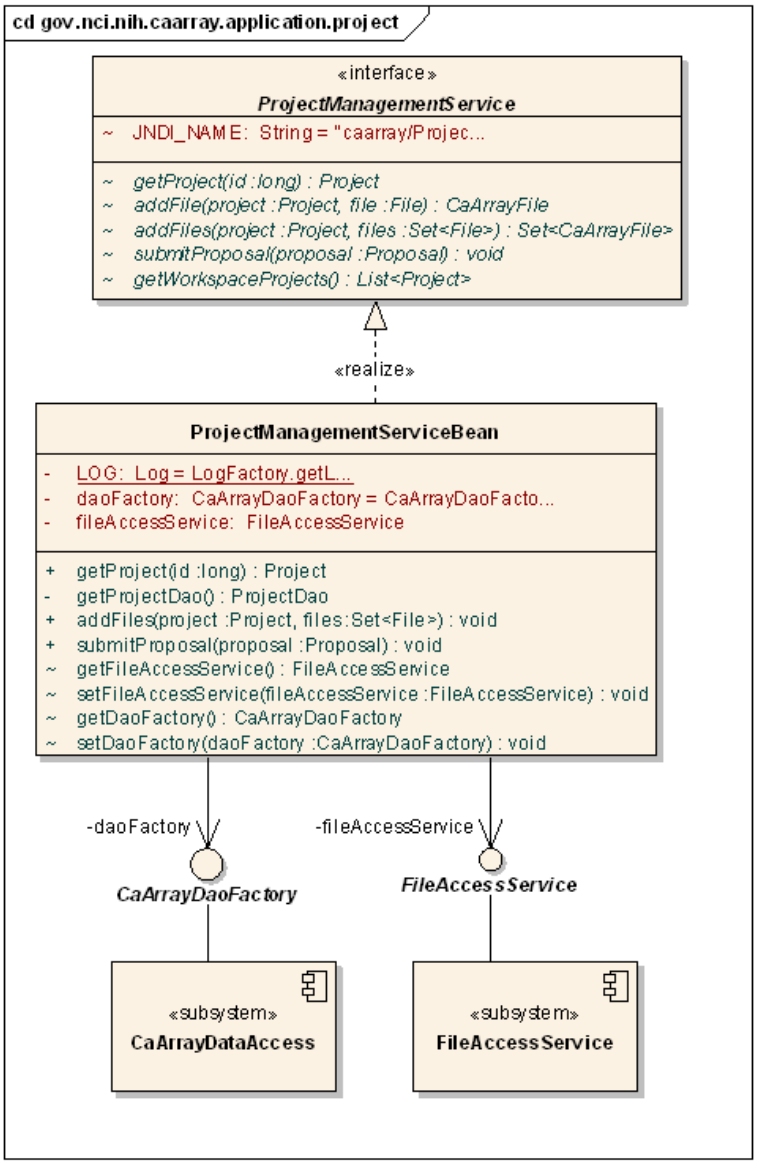

Figure 3.9 ProjectManagementService implementation class diagram

FileAccessService

The FileAccessService subsystem is implemented as a stateful session bean and is responsible for storage and retrieval of all files managed within caArray (annotation, array design and data). Files that are uploaded to caArray are registered with the FileAccessService which reads the files, compresses the contents and stores the contents as a BLOB (in the database) associated with a CaArrayFile instance. The static structure of this subsystem is shown in *Figure 3.10* and the act of reading and storing file contents is shown in *Figure 3.11*. Client subsystems that require access to file contents call the `getFile(caArrayFile : CaArrayFile) : File` method which performs the inverse operation; reading the contents from the BLOB, decompressing the contents and writing them to temporary file reading area. Clients are expected to call

close(file : File) when done using files so that the subsystem may remove the files from temporary file system storage.

This FileAccessService extracts each requested file to a different temporary location. This does mean potentially having duplicates of temporarily uncompressed files, but this should be the exception as files are only needed on download and when parsed. After weighing the potential approaches, the minor overhead of temporary duplicates was definitely preferable to the overhead of maintaining file reference counters across multiple sessions. The session does maintain its own internal map of files opened by the session's client which we use to clean up on session removal in case the client doesn't call close() (for example, if an exception interrupts the client's flow).

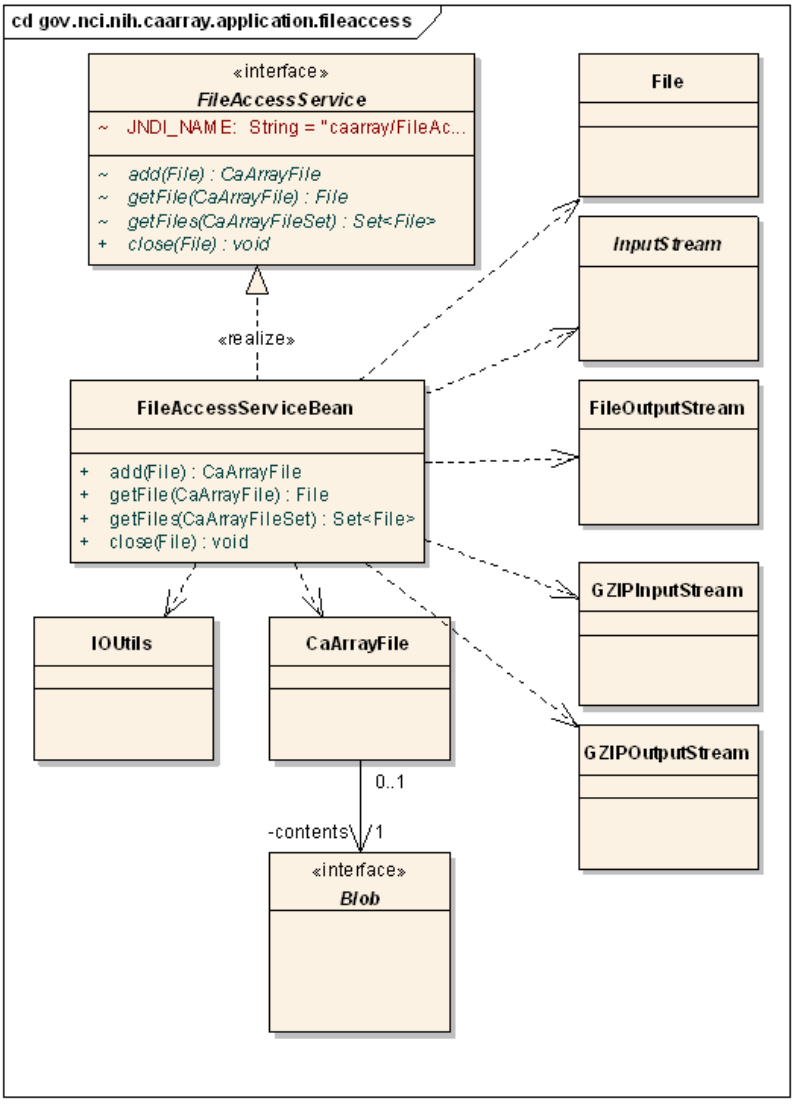

Figure 3.10 FileAccessService implementation class diagram

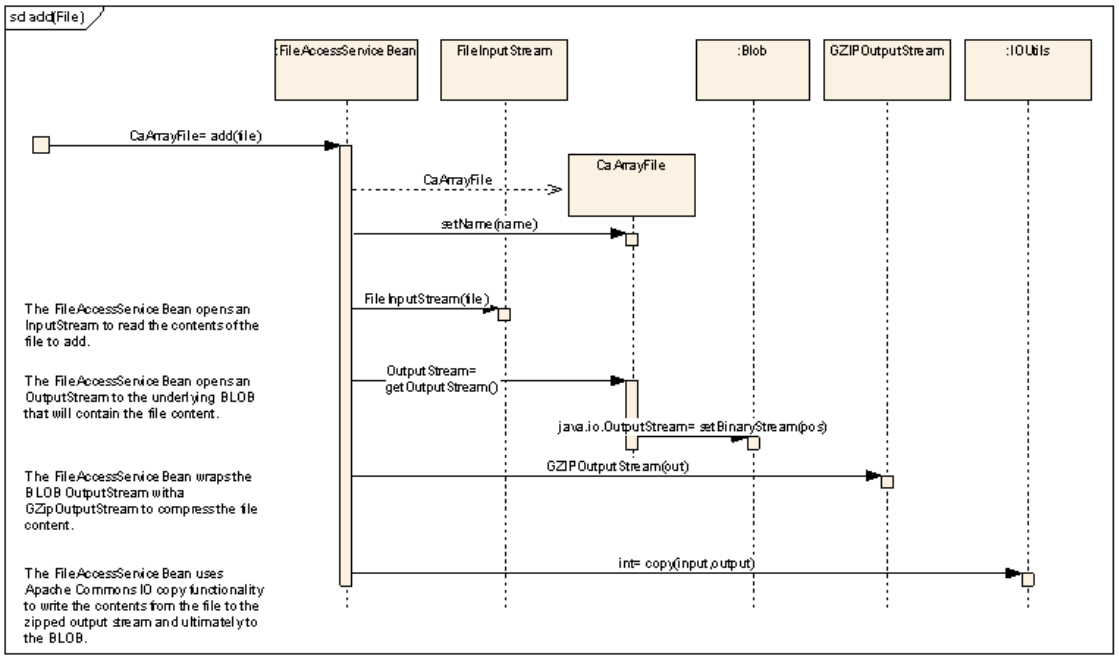

Figure 3.11 FileAccessService operation AddFile(file : File)

FileManagementService

Whereas the FileAccessService handles the lower level functionality of file storage and retrieval, the FileManagementService subsystem is responsible for performing higher level logical file operations, specifically, the validation and import of MAGE-TAB annotation, array design files and array data files. The implementation of the subsystem does this through delegation to subsystems responsible for handling these various types of data. The organization of the FileManagementService implementation is shown in *Figure 3.12* where the central bean delegates import and validation functionality to a set of importer classes that in turn delegate to the lower-level subsystems.

Validation results are instantiated by the lower-level subsystems and then the FileManagementService associates these with the CaArrayFile object that represents the validated annotation or data file.

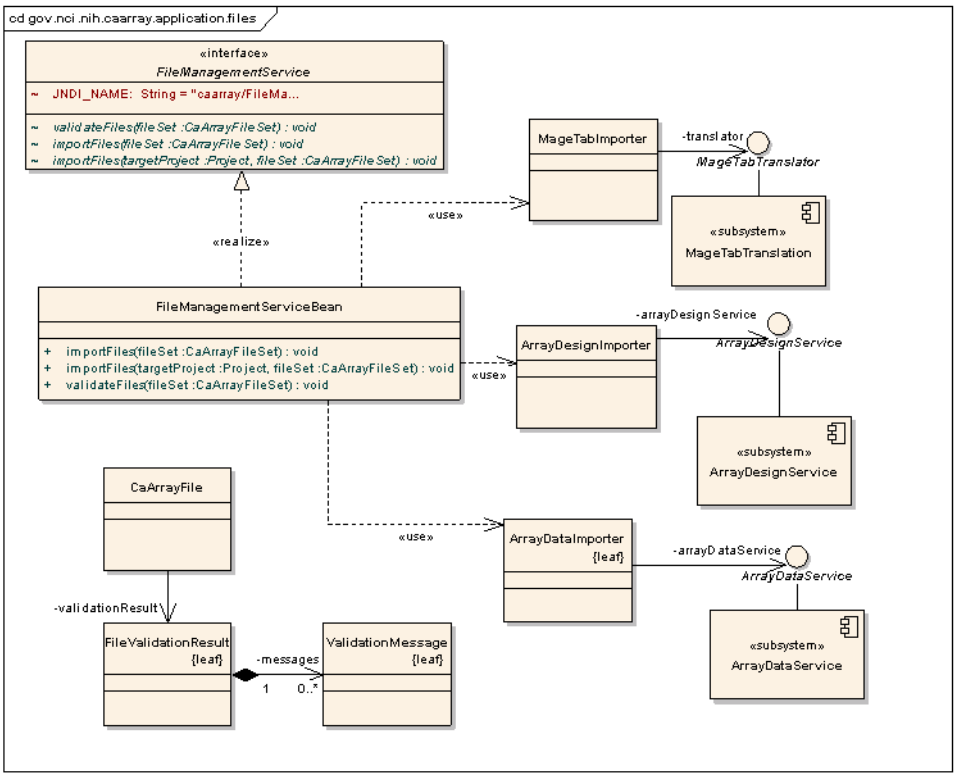

Figure 3.12 FileManagementService implementation

ArrayDataService

The ArrayDataService subsystem is responsible for validating array data files, storing array data and retrieving array data when requested by clients. The typical order of events related to a given array data file is as follows:

- An array data file is validated using the validate(arrayDataFile : CaArrayDataFile) : ValidationResult operation. Only the generated FileValidationResult is created and persisted.
- The data file is imported using the import(arrayData : AbstractArrayData) : void operation. At this point, a DataSet and associated HybridizationData and AbstractDataColumn instances are created, but individual data values are not retrieved or persisted.
- A client requests data via the getData(arrayData : AbstractArrayData, types : List<QuantitationType>) : DataSet method. If this is the first request for the AbstractDataColumns associated with the provided QuantitationTypes, the requested data is parsed from the files as columnar arrays of primitives and stored persistently as serialized, GZipped byte[] representations of the arrays. The data is then returned to the client. If the data had previously been loaded as a result of earlier calls to getData the existing serialized byte[] is deserialized.

Though caArray 1.x and 2.0 do ultimately use the database to persist array data, the approaches are radically different. The 2.0 design does not exhibit the same perfor-

mance and resource consumption when compared to 1.x. For illustration purposes, the 1.x design stores the array data from a file as a large number of rows (one per design element) with a column per data value, maintaining a complete relational representation of the entire data set. caArray 2.0 is using a single BLOB entry to store a large primitive array of data corresponding to a complete column's worth of data from a data file. For example, whereas a CEL file consumes hundreds of thousands of rows of seven columns apiece in 1.x, the new design will create 7 rows each with a single serialized, compressed representation of hundreds of thousands of data points.

In addition, since data is only parsed and persisted when first requested, untouched data files will not incur any additional storage overhead. Furthermore, only requested columns of data are parsed and persisted. This is important since only a small subset of the quantitation types of a given data type are of interest to clients (for example, 2 columns out of 20 such as "detection' and "chip signal" of a .chp file in an Affymetrix experiment).

The method import(arrayData : AbstractArrayData) : void is illustrated in *Figure 3.13* .

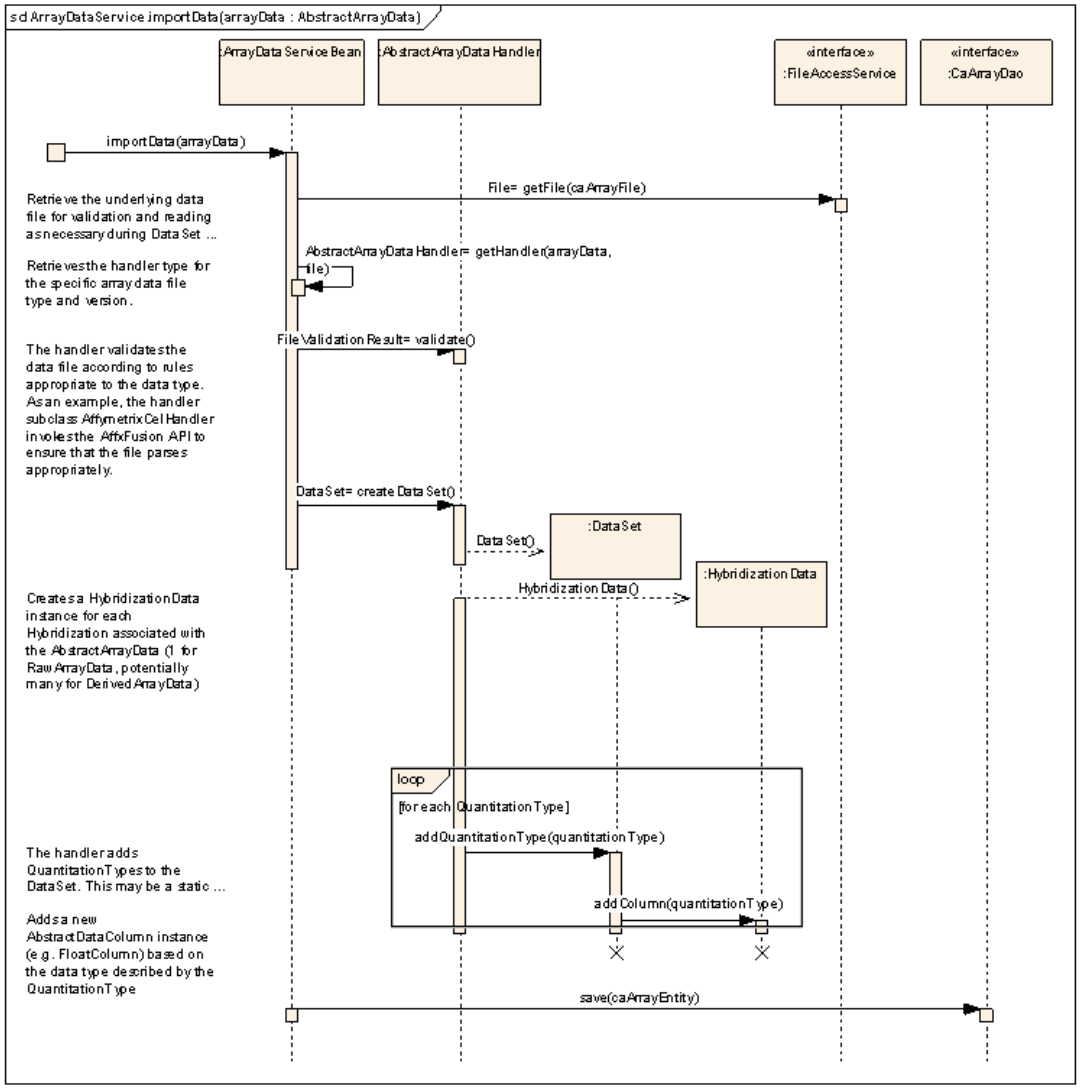

Figure 3.13 Import Operation implementation

ArrayDesignService

The ArrayDesignService is responsible for parsing, persisting, and retrieving array design annotation from the various array annotation file types. In the implementation of the subsystem, each file format is handled by a specific subclass of AbstractArray-DesignHandler. These subclasses contain the logic to parse, validate, and persist the details of a given format. The array annotation is stored in the ArrayDesignDetails and caBIO reporter annotation structures described earlier in the section on the caArray Domain Classes package. Add sentence about figure The major classes and depen-dencies are represented in *Figure 3.14*

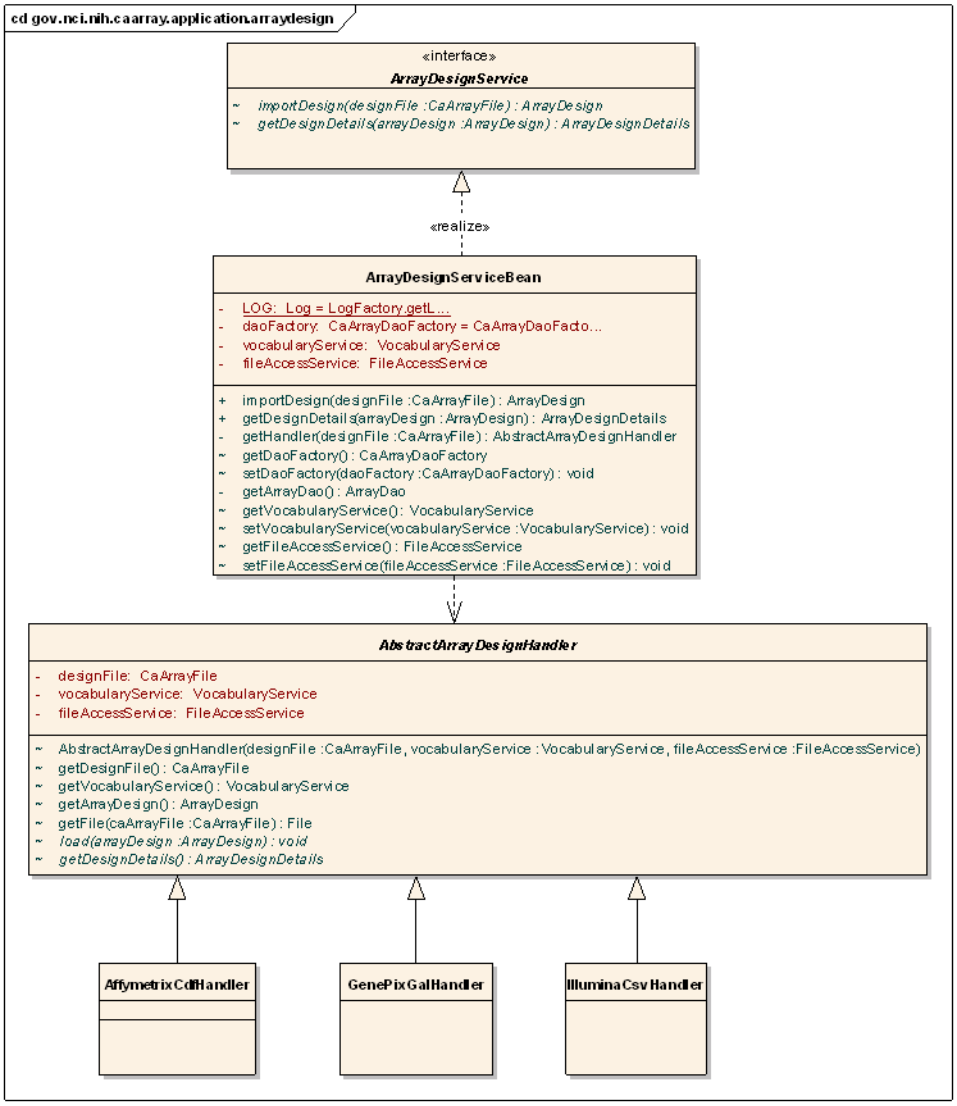

Figure 3.14 ArrayDesignService subservice implementation

MageTabParser

The MageTabParser subsystem is responsible for reading a set of files in MAGE-TAB format, validating the files and ultimately representing the contents of the files in object

model based on MAGE-TAB concepts. The major implementation classes are shown in *Figure 3.15*..

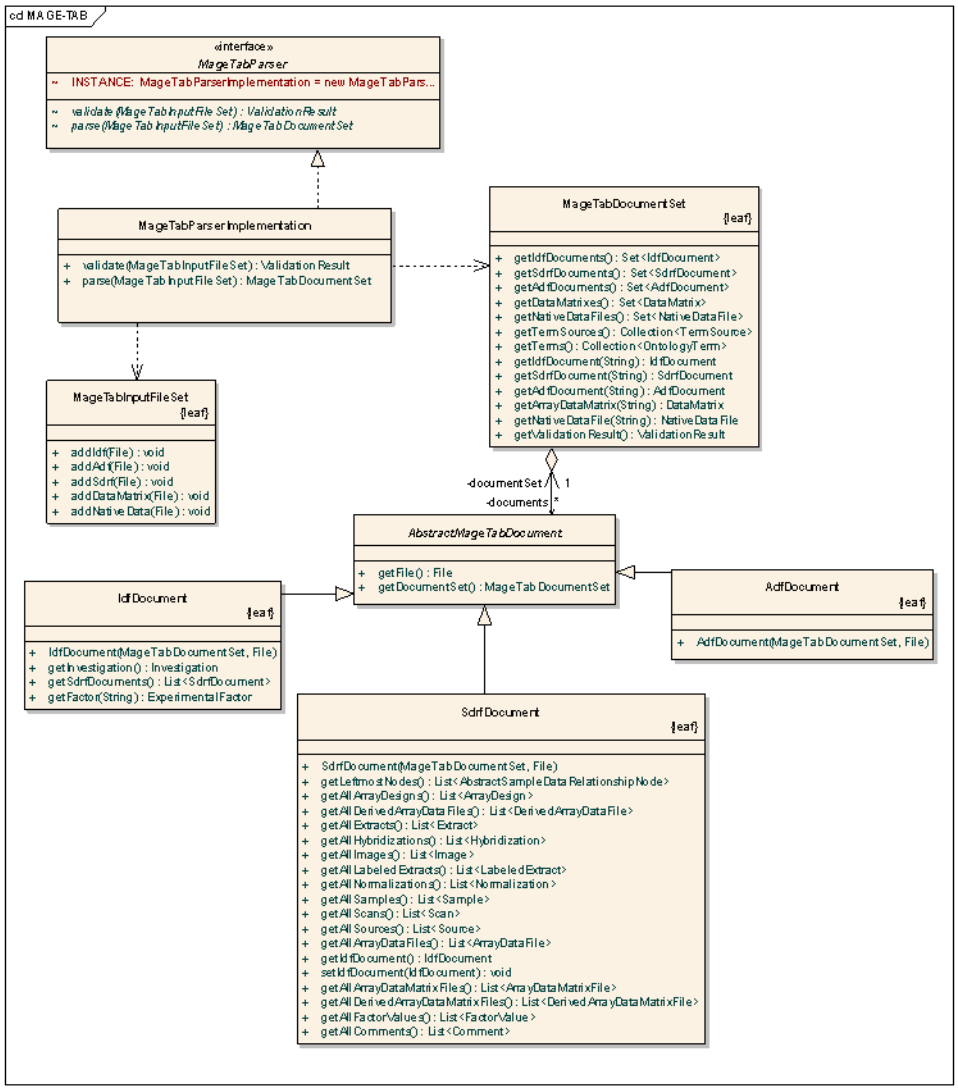

Figure 3.15 MageTabParser Subsystem implementation

MageTabTranslation

The MageTabTranslation subsystem of caArray is invoked to translate from the MAGE-TAB object model generated by the MageTabParser system to a corollary caArray Domain Class representation. It implements a set of translator classes for each MAGE-TAB document type and for shared data types (i.e. Terms and TermSources)Add sen-

tence about figure The major classes and dependencies are represented in *Figure 3.16*.

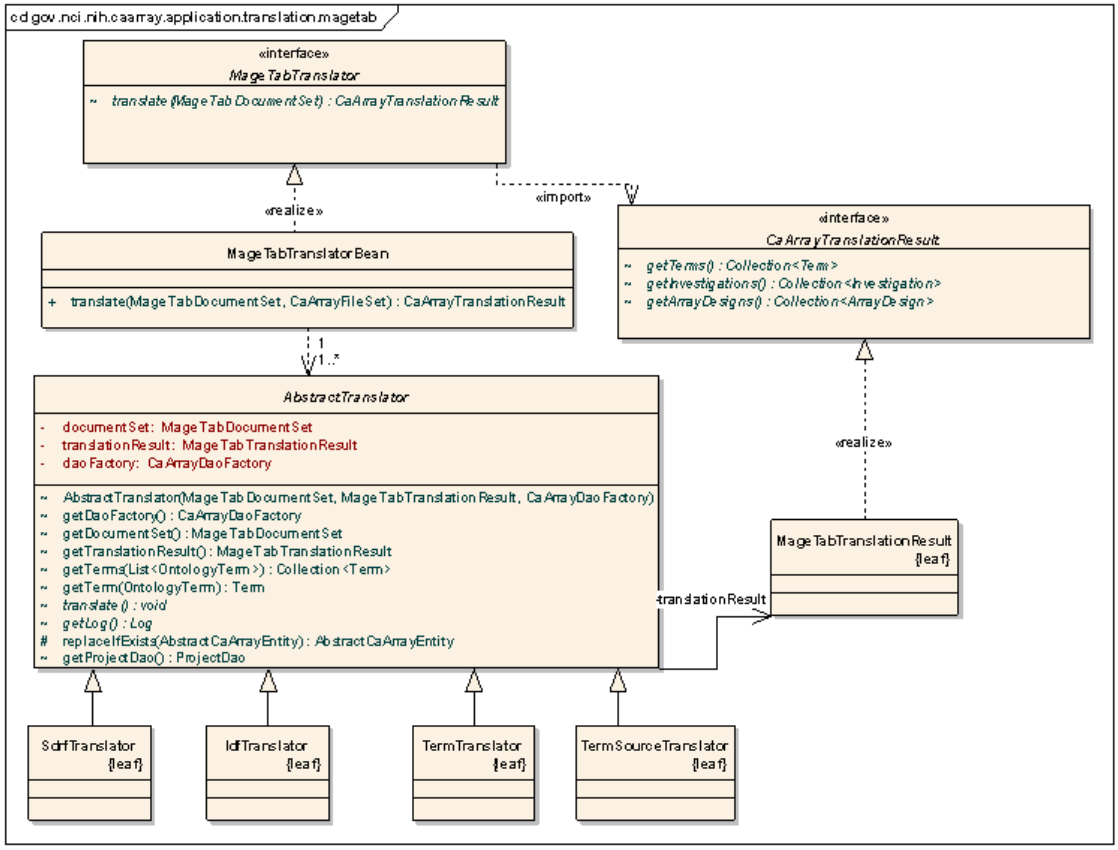

Figure 3.16 MageTabTranslator Subsystem implementation

CaArrayDataAccess

caArray uses the standard Data Access Object pattern to provide data updates and retrievals. The DAOs are exposed as Java interfaces accessed through a Factory class. The implementations of the DAOs use Hibernate 3.2 as the underlying persistence mechanism.

VocabularyService Subsystem

caArray requires the use of controlled vocabularies such as the MGED Ontology. In order to provide a consistent view of external vocabularies, caArray includes a VocabularyService subsystem that manages interfacing with external vocabulary sources, provides a consistent Term and Category view of vocabularies, and manages local storage of vocabulary elements for efficiency.

The VocabularyService subsystem makes use of the EVS to verify controlled vocabulary elements through the classes provided by caCORE 3.2. The major classes and dependencies are represented in *Figure 3.17*.

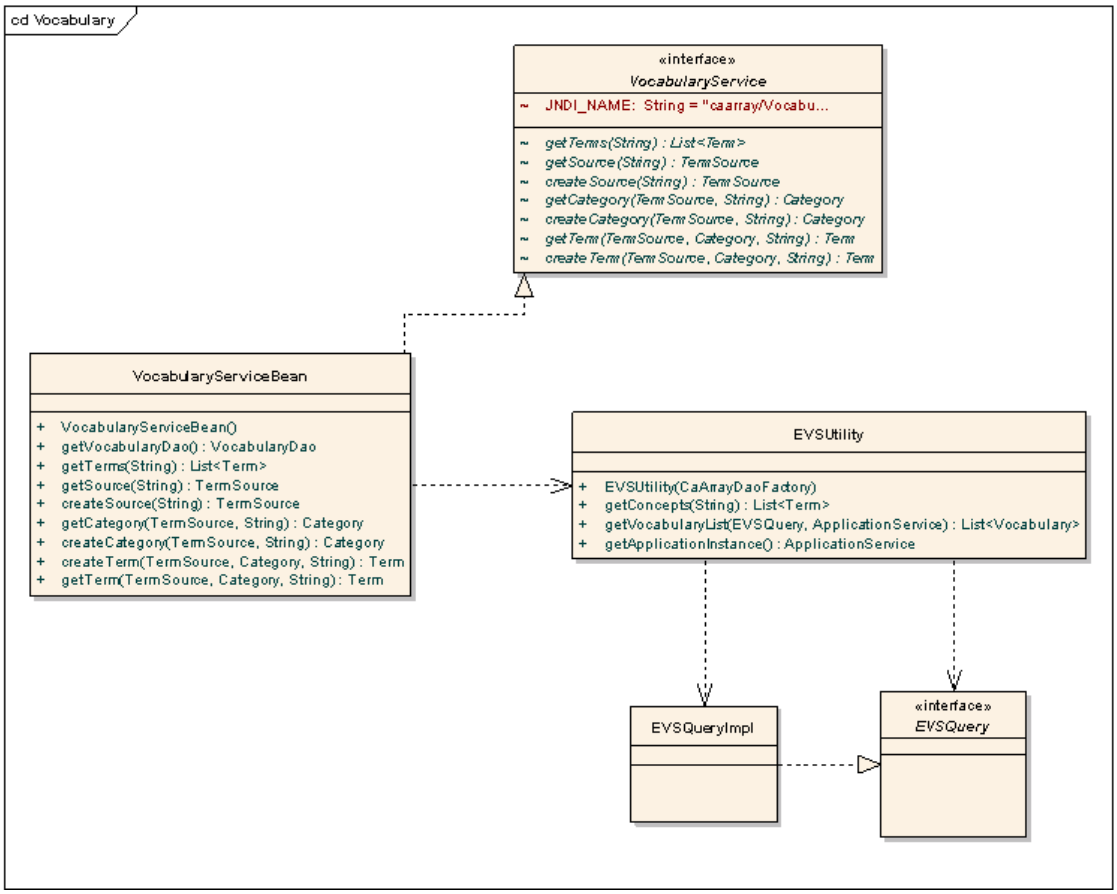

Figure 3.17 VocabularyService implementation

caarraydb

Heading level correct? The caarraydb component represents the MySQL database schema that is used to store all of caArray's persistent data. The schema is generated directly from Hibernate annotations recorded in the caArray Domain Classes so that the schema and domain classes are kept easily synchronized. caArray supports MySQL versions 4.1 and 5.0.

Use-Case Realizations

Manage Experiment Data Files

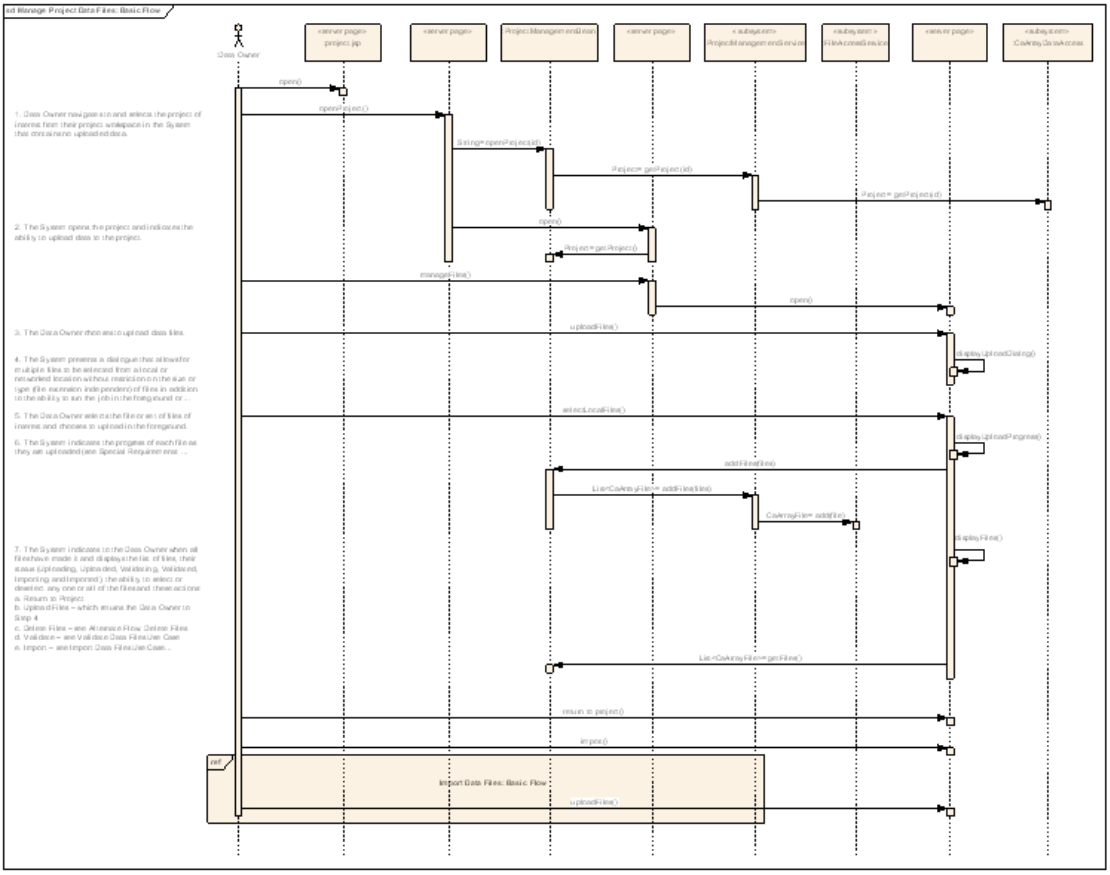

Figure 3.18 Manage Experiment Data Files basic flow sequence diagram

Import Data Files

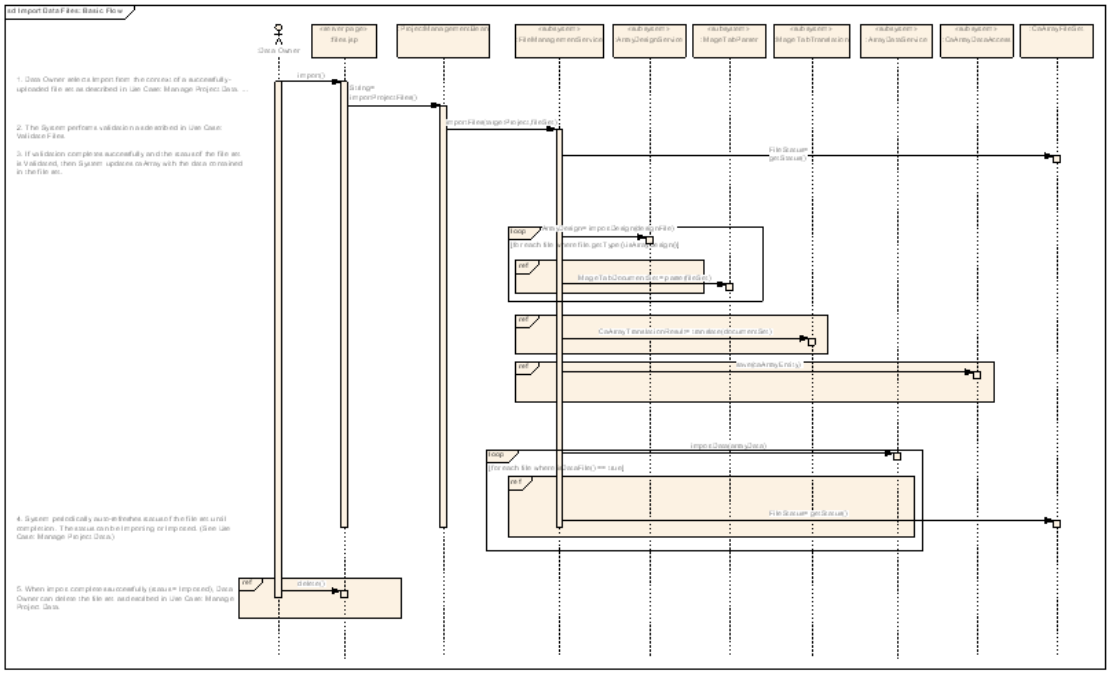

Figure 3.19 Import Data Files basic flow sequence diagram

Acquire Experiment Data via API

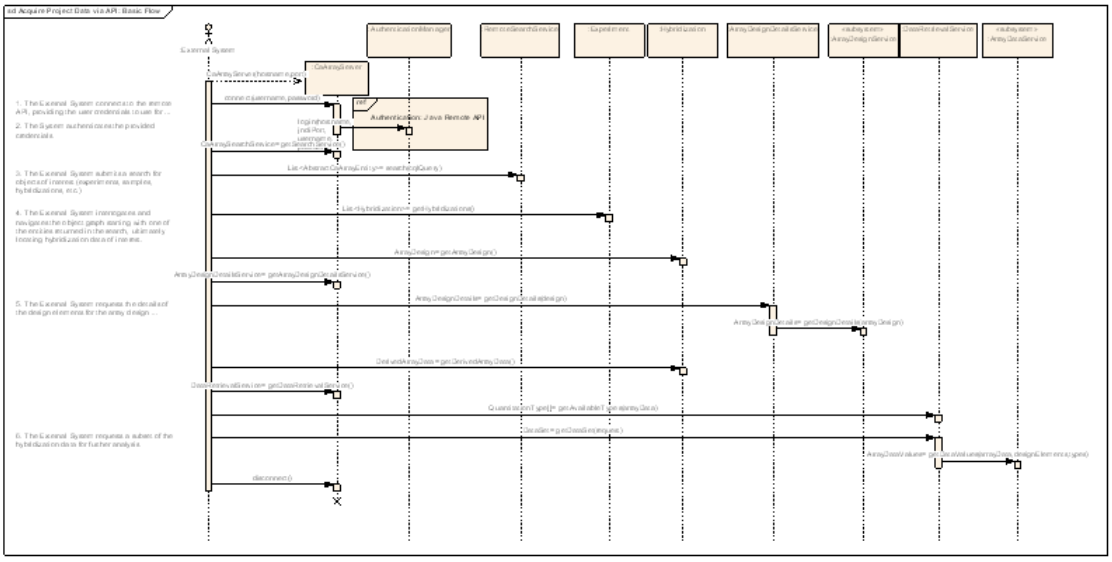

Figure 3.20 Acquire Experiment Data via API basic flow

Grid access follows a similar flow, with the CaArraySvc delegating calls to these same remote session beans.

CHAPTER

4

IMPLEMENTATION VIEW

Overview

The major physical artifacts that comprise the caArray software deployment units are illustrated in *Figure 4.1*. The major artifacts and their relationships to the subsystems they realize are described in the following section, *Artifacts*.

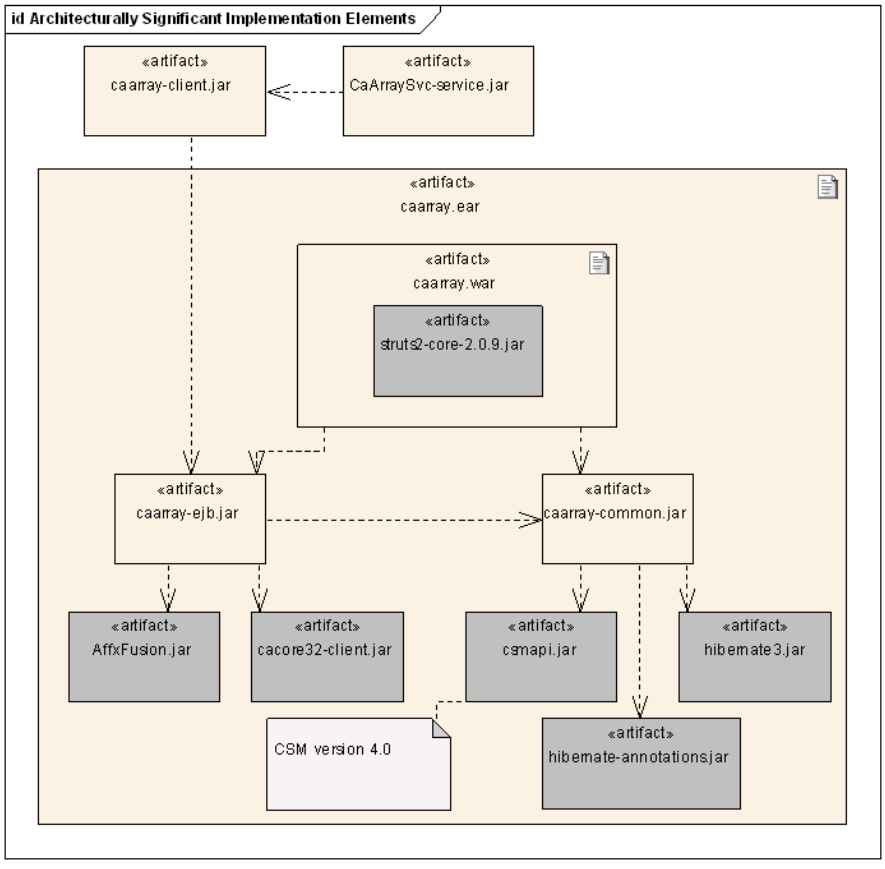

Figure 4.1 Architecturally significant implementation elements

Artifacts

caarray.ear

The caarray.ear artifact is the J2EE Enterprise Application Archive (EAR) that contains all of the web portal application and EJB components that make up the User Interface, Remote Java API, Application Logic, and Business Logic layers of the application. The EAR also contains the third-party JARs necessary to support the Application and Business Logic Layers of the application.

caarray.war

The caarray.war artifact packages the JSPs and caArray Struts 2 classes that comprise the User Interface layer of caArray. The WAR also contains the Struts 2 third-party JARs and necessary supporting JARs.

caarray-ejb.jar

The caarray-ejb.jar packages the implementation of all of the EJB subsystems and the implementation of the remote API interfaces defined in caarray-client.jar. This includes all of the subsystems in the Application Layer and VocabularyService subsystem from the Business Logic Layer of the logical model. The major third-party dependencies are to cacore32-client.jar to support communication with EVS and AffxFusion.jar which provides Affymetrix file format parsing support.

caarray-common.jar

The caarray-common.jar contains the caArray Domain Classes packages, the CaArray Data Access subsystem and the MageTabParser subsystem. The major third-party component dependencies noted are hibernate3.jar and hibernate-annotations.jar to support annotation-based Hibernate ORM mapping and to csmapi.jar to support entity access authorization.

caarray-client.jar

The caarray-client.jar contains the remote EJB interfaces required by Java Remote API clients and the caArray Grid Service. This JAR also repackages other third-party classes required by remote clients.

CaArraySvc-service.jar

The CaArraySvc-service.jar all other .jars (and .ear and .war) start with lower case letters. Is this one correct? contains the caGrid API implementation classes.

CHAPTER 5
DEPLOYMENT VIEW

The typical deployment configuration for caArray is documented in *Figure 5.1*. The NCICB deployment of caArray will be similar to the scenario modeled in the figure, with the addition of a front-end Apache web server that receives the HTTPS requests and then delegates these requests to the JBoss server where caArray is deployed. While Globus is shown as the grid service execution environment, Tomcat may optionally be used in its place. External adopters might also choose to deploy application components and execution environments to a single server.

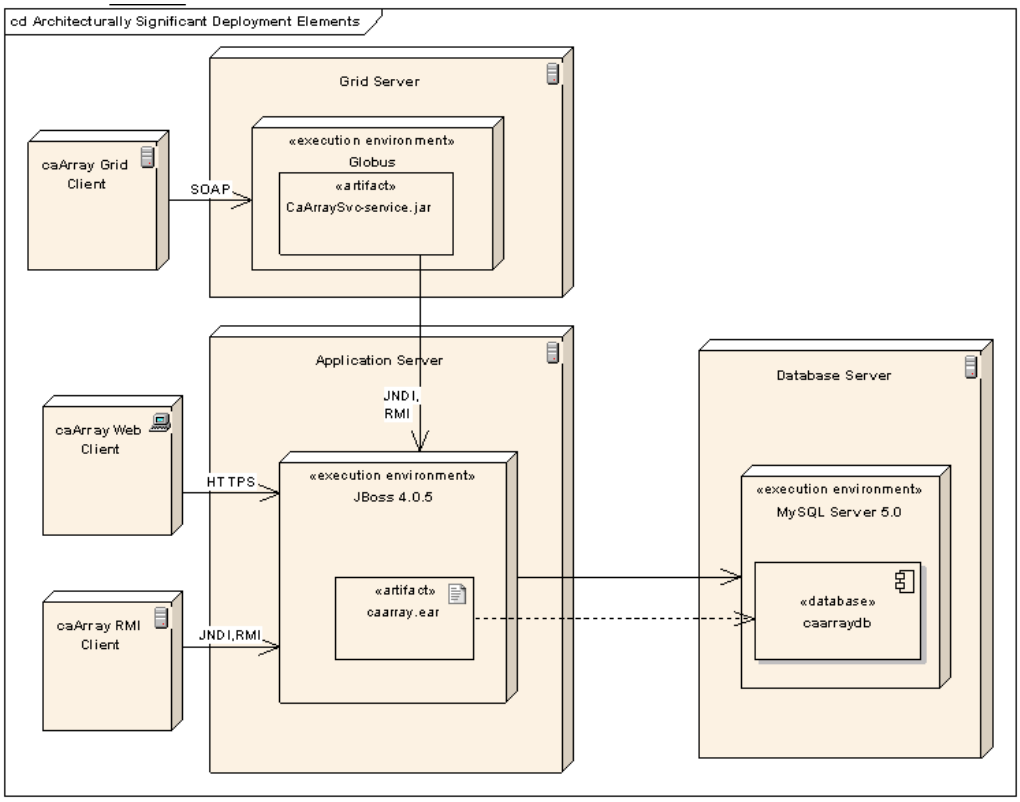

Figure 5.1 Architecturally Significant Deployment Elements

APPENDIX A
GLOSSARY

This glossary defines acronyms, abbreviations, and terminology used in this guide.

Term	***Definition***
caBIG	cancer Biomedical Informatics Grid
caBIO	Cancer Bioinformatics Infrastructure Objects
caCORE	cancer Common Ontologic Representation Environment
caDSR	Cancer Data Standards Repository
caMOD	Cancer Models Database
CGH	Comparative Genomic Hybridization
EBI	European Bioinformatics Institute
EVS	Enterprise Vocabulary Services
MAGE 1.1	MAGE 1.1 is a widely-used microarray data standard or guideline
MAGE-ML software format	XML-based standard for representation of microarray data
MIAME 1.1	MIAME1.1. is a standard or guideline for the minimum amount of information required to make a microarray record useful to others.
MGED Ontology	MGED Ontology is a controlled vocabulary standard that concisely defines terms as they relate to Microarrays and caArray as a whole
MGED	Microarray Gene Expression Data Society
MMHCC	Mouse Models of Human Cancers Consortium
NCI	National Cancer Institute
NCICB	National Cancer Institute Center for Bioinformatics
URI	Uniform Resource Identifier
URL	Uniform Resource Locators

Term	***Definition***
XML	Extensible Markup Language (**http://www.w3.org/TR/REC-xml/**) - XML is a subset of Standard Generalized Markup Language (SGML). Its goal is to enable generic SGML to be served, received, and processed on the Web in the way that is now possible with HTML. XML has been designed for ease of implementation and for interoperability with both SGML and HTML

INDEX

A

Acquire Experiment Data via API 32
acquire experiment data via API, use case 9
API
  Grid 14
  Remote Java 15
architectural
  constraints 5
  goals 5
architecture
  description of views 5
  logical view overview 11
  use case view 7
ArrayDataService 25
ArrayDesignService 27

C

caArray
  description 1
  domain classes 16
  introduction 1
  Technical Guide description 1, 2
  User's Guide text conventions 3
caarray.ear 34
caarray.war 34
caarray-client.jar 34
caarray-common.jar 34
CaArrayDataAccess 29
caarraydb 30
caarray-ejb.jar 34
CaArraySvc-service.jar 34
caBIG silver compliance 6
Cancer Array Informatics Project See caArray 1

D

data retrieval 6
data storage 6
deployment view 35
document conventions 3
domain classes 16

F

FileAccessService 22
FileManagementService 24

G

Grid API 14

H

High Performance Data Parsing 6

I

import array designs 9
Import Data Files 32
import data files, use case 9

J

J2EE 1.4 application 11

L

logical view, overview 11
login use case 8

M

Manage Experiment Data Files 31
manage experiment data files, use case 9

P

ProjectManagementService 21

R

reading materials 2
remote API usability 6
Remote Java API 15
Resources, caArray 2

S

silver compliance, caBIG 6

structural hierarchy 11
subsystems
  dependencies 11
  description 11
  figure 12

T

text conventions in user guide 3

U

use case
  acquire experiment data via API 9
  import array designs 9
  import data files 9
  login, description 8
  manage experiment data files 9
  validate experiment data files 9
use case view 7
user interface layer 13

V

validate experiment data files, use case 9
view, architecture descriptions 5
VocabularyService 29