CBFDAP Web App Developer Guide

I. Burak Ozyurt

August 13, 2013

Contents

List of Figures

Chapter 1

Overview

1.1 CBFDAP web app directory structure

Under the root directory *BIRN/clinical*, you will find the following toplevel directories

- bin - where scripts and external binaries (mostly used for DICOM/AFNI conversion, Web services deployment etc).
- conf - configuration files used by the application. Mostly configuration file templates, since most of the configuration files are generated from templates combined with the ant questionaire.
- lib - the third party libraries used.
- log - if configured application specific logging info will be saved here
- src - where the Java source code resides
- web - where the JSPs,images, JavaScript, CSS and web configuration files resides

1.2 Packages Overview

CBFDAP web app currently consists of 1390 Java classes under *src* directory, and 436 JSP files under *web/pages* directory in the source distribution in CVS. CBFDAP web app uses Struts Tiles templating framework. Nearly each JSP is accompanied with another JSP acting as the template binder for its corresponding JSP going into body slot in the template. The packages organizing the Java code are summarized below.

- clinical.web.workflow - Provides processing job implementations and related utility functionality
- clinical.web.workflow.cbf - Provides individual and group CBF processing job implementations
- clinical.web.workflow.cbf.group - Provides second level CBF group processing job implementations for baseline, ROI and standard space voxelwise group analysis.
- clinical.web.workflow.remote - Under construction. Will provide master-worker type remote processing of jobs spread across multiple machines.
- clinical.web.workflow.common - Provides common utility functions for job processing

- clinical.web.scheduler - Provides the implementation of job scheduler that managent all long running tasks.
- clinical.comm - extended XML-RPC communication library used by remote administration
- clinical.exception - Provides exceptions used by the code generated data access layer classes.
- clinical.server - Provides data access layer, data transfer objects and some portions of session facade layer serving the web application.
- clinical.server.dao - Provides data access object (DAO) interfaces (CRUD + QBE) for the data access layer.
- clinical.server.dao.oracle - Oracle specific DAO interface implementation as generated by codegen application.
- clinical.server.dao.postgres - Postgres specific DAO interface implementation as generated by codegen application.
- clinical.server.facade - Provides SRB image series and local file cache handling, subject/visit details query session facade interfaces.
- clinical.server.image - Provides data access layer, data transfer objects and some portions of session facade layer serving the web application.
- clinical.server.impl - Provides implementation of the session facade interface ImageHandler.
- clinical.server.upload - Old version of batch assessment/experiment data upload/ conversion/ conditioning functionality for UCSD Morph BIRN database.
- clinical.server.utils - Provides server side utilities for Oracle CLOB support.
- clinical.server.vo - Provides value objects for the UCSD Morph BIRN database management/querying application(s).
- clinical.test - Provides unit tests using the Junit framework and Assessment Query functional testing using the HttpUnit test framework.
- clinical.tools - Provides tools used to provide auxiliary functionality for the web application, including installation tools.
- clinical.tools.install - Provides an installation tool to interactively create a users.xml file and a custom Ant task for it.
- clinical.tools.maintenance - A simple Java scripting tool to create quick and database maintenance programs for specific tasks in java using the DAOs and VOs
- clinical.tools.dbadmin - Provides command line and GUI tools for managing different aspects of the CBDDAP database.
- clinical.tools.dbadmin.migration - Provides a command line tool to migrate an experiment from one CBFDAP database instance to another including all the associated image/job processing data.
- clinical.tools.security -
- clinical.tools.security.server - Provides a simple server using extended XML-RPC as remote procedure call mechanism for remote administration of the CBFDAP web app

- clinical.tools.upload - Provides a tool for data curation and upload to the database. Mainly used for UCSD ADRC data preparation/transformation and upload for MBIRN.
- clinical.upload - Provides batch assessment, experiment/visit uploading functionality with data conversion and conditioning support UCSD ADRC data.
- clinical.upload.conditioner - Provides data conditioning support used when the raw data needs filtering more involved than type conversion, one-to one mapping.
- clinical.utils - Provides generic utilities ranging from database connection pool with named users, CSV parser, date arithmetic and formatting to name a few.
- clinical.web - Provides web based user interface for clinical assessment, derived data querying and clinical subject data management functionality using Struts web framework.
- clinical.web.actions - Provides Struts web framework controller actions.
- clinical.web.common - Provides interfaces for the session facade for the business logic and auxiliary services.
- clinical.web.common.query - Provides generic assessment, subcortical derived data query building functionality and dynamic query building and processing for single tables.
- clinical.web.exception - Provides exceptions used by the presentation layer.
- clinical.web.forms - Provides Struts web framework form beans and helper objects for UCSD Morph BIRN database user interface.
- clinical.web.game - Provides generic assessment management engine (GAME) support.
- clinical.web.game.forms - Provides Struts form beans as generated by the Clinical Assessment Layout Management (CALM) tool.
- clinical.web.helpers - provides helper classes used mainly by struts actions.
- clinical.web.helpers.security - provides helper classes used for authentication and authorization.
- clinical.web.image - Provides asynchronous DICOM to AFNI conversion coordinator.
- clinical.web.services - Provides generic assessment, subcortical derived data query building functionality and dynamic query building and processing for single tables.
- clinical.web.soap - stats web service for Slicer
- clinical.web.tags - custom JSP tags
- clinical.web.tags.sec - Provides custom JSP tags for session expiration and user login checking for site navigation, conditional execution of parts of JSP based on a user's privileges.
- clinical.web.vo - Provides web tier side value objects used by Struts forms in transfering and manipulation of presentation data.
- clinical.xml - XCEDE import/export web services support
- clinical.xml.export - XCEDE export web service
- clinical.xml.gui - provides a XCEDE export GUI web service client.
- clinical.xml.importer - XCEDE import web service (in development)

1.3 Configuration Files and Build Process

- *conf/siteid_map.properties* - a lookup table of siteID - Site name map. Used to infer the site for a subject from its subject ID due to the absence of an explict site ID for multi-site queries.
- *conf/commons-logging.properties* - properties file for Apache Commons Logging framework as used by Struts (mainly setup to delegate for Log4j logging framework).
- log4j.properties.template - template file used by Ant to generate Log4j configuration file.
- *conf/clinical.properties.template* - template file CBFDAP web app specific properties used to generate *clinical.properties* file from information gathered during Ant build questionaire. Any updates to clinical that change what updates CBFDAP needs in order to function with this version of clinical should have the minimum and maximum major and minor version numbers changed in this file.
- *conf/users.xml.example* - example users.xml file for database, user and privileges configuration. A users.xml file will be generated from information gathered during Ant build questionaire by Ant.
- *conf/as_var_map.xml.example* - score/type mapping/conversion definition example file used for multi-site queries, currently not used in fBIRN and might be broken since the result combining logic used has changed dramatically.
- *conf/resources/application.properties* - resource file for externalized (parametrized) messages like error messages, button labels etc.

For efficiency and as a development process aid the Ant generated configuration files are not regenerated the next time you run Ant. To force Ant to regenerate the configuration files, you just need to delete the generated one. Do not make a permanent change in the generated configuration file. Do the change in the template configuration file or always use Ant to build them for you through the questionaire. If you make changes to the template file don't check them in unless the change will be applicable in a general manner.

1.4 Lifecycle operations

CBFDAP web app uses Struts as its web framework. As any J2EE application, HID web app lives within an application server, which provides life cycle management for the application. Some resources like connection pooling, cleanup services, remote administration etc need to be started before the web app starts serving users and the resources/services used should be properly released backed or closed when the application server (here the servlet container) shuts down.

1.4.1 Startup

The *ServicesPlugin* class implements Struts *PlugIn* interface for the lifecycle management. The *init* method on *ServicesPlugin* class initializes this application when the Struts framework starts. The initialization steps are as follows

- bootstrap Security Service
- prepare query processor global cache
- get the application specific properties and save them to application scope

- initialize ServiceFactory
- initialize DAOFactory
- initialize the assessment variable mapping for quasi-mediator
- startup image file cache cleanup thread

1.4.2 Shutdown

The *destroy* method on *ServicesPlugin* class, shuts down the connection pool and the remote admin server.

Chapter 2

Common Services

2.1 Security

CBFDAP web app has a simple application security mechanism, initiallly planned to be a place-holder till a more advanced security mechanism (like PKI) that will be adopted IN BIRN. However, it has survived till today and it is one of the parts of CBFDAP web app that may need enhancement. The class diagram for Security services is shown in Figure 2.1. The class *SimpleSecurityService* implements four security interfaces for authentication, authorization, run-as management for multi-site queries and remote administration (from local remote client for user management). The security service data is persisted along with the database connection parameters in an XML file namely *users.xml*. The structure and semantics of the *users.xml* is specified in the *users.xsd* XML schema file.

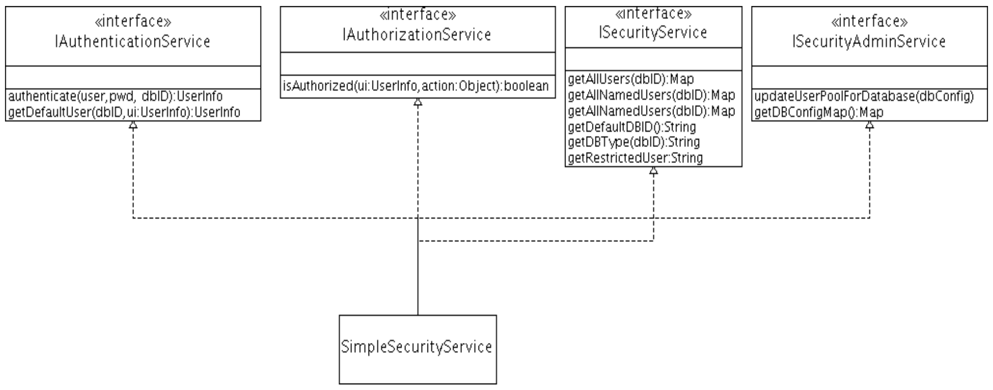

Figure 2.1: Security Service Class Diagram.

2.2 Connection Pool

CBFDAP web app provides a non-conventional two level connection pool to facilitate multi-site queries and allow Oracle Label security. You can configure multiple (possibly remote) databases, multiple named users per database and multiple database connections per named user allowing multiple CBFDAP web app users to share the same named user. A simplified class diagram is shown in Figure 2.2

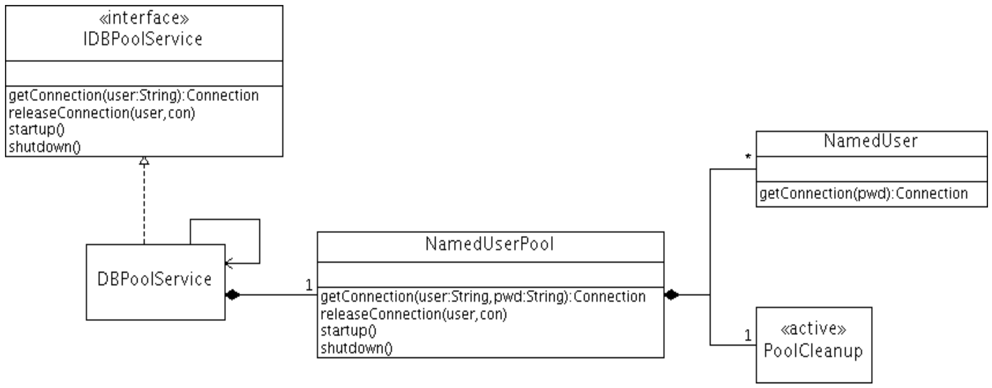
«interface»
IDBPoolService
getConnection(user:String):Connection
releaseConnection(user,con)
startup()
shutdown()
DBPoolService
1
NamedUserPool
getConnection(user:String,pwd:String):Connection
releaseConnection(user,con)
startup()
shutdown()
*
NamedUser
getConnection(pwd):Connection
1
«active»
PoolCleanup

Figure 2.2: Multi-level Connection Pool Class Diagram.

2.3 Data Access Layer

CBFDAP web app uses a code-generated Data Access Object (DAO) based simple object-to-relational mapping strategy with query by example (QBE) as the main retrieval mechanism. Even though QBE is not a very powerful approach, it was adequate enough for most of the data maintenance related queries. Multi-table complex queries are build by handcoded query builders using an abstract syntax tree (AST) representation of the user query translated to native SQL from AST representation. Also there is a query by criteria (QBC) service for single table complex queries as provided by the implementation of the *IQueryProcessor* interface. The *AbstractQueryProcessor* provides the common QBC functionality. The *OracleQueryProcessor* and *PostgresQueryProcessor* provide Oracle and Postgres specific QBC functionality, respectively.

Transparent multiple database support for DAOs is provided by using abstract factory design pattern. The *DAOFactory* class creates the corresponding DAO for the corresponding database type for the provided database ID and returns the DAO interface instead of the actual implementation.

2.4 LRU Image Cache Cleanup Service

This is part of the service related image preview and retrieval from SRB. This functionality was for early MBIRN and might be easily upgraded for other usage scenarios. However, currently it is not adopted for fBIRN usage.

To increase response time under heavy load, a least recently used (LRU) file cache is used in the middle tier server. The active class *FileCacheCleanupService* coordinates LRU file cache mechanism by using external Perl scripts for the cache eviction etc. To assure proper concurrent operation, during image series retrieval from SRB and DICOM to AFNI conversion, exclusive file locks are used. To avoid race conditions during data streaming and cache cleanup, a read/write lock mechanism is simulated with file locks. The cache eviction is based both on file age and size.

2.5 Remote Administration server

The goal of remote administration is to dynamically update/change or monitor system parameters and security administration without bringing down the tomcat server. The remote administration has two parts;

- the admin server running in the CBFDAP getCollectiveNameweb application (clients can connect on port 11111)

- a remote admin client running in its own process space (i.e. as a standalone app different than the CBFDAP web app).

The provided remote client is intentionally harcoded to work from the localhost (i.e. same machine as the CBFDAP web app (clinical)) for security reasons. The admin server main class is is *AdminServer* from *clinical.tools.security.server* package. *AdminServer* uses extended XML-RPC as implemented in *clinical.comm* package for communication protocol with its clients.

Check REMOTE_ADMIN_SETUP_GUIDE.txt for info on configuration and usage.

Chapter 3

Workflow Management

3.1 Job Scheduler

To handle all the required workflows for individual and group processing of CBF jobs, CBFDAP uses a builtin job scheduler whose class diagram is shown in Figure 3.1. The *JobScheduler* class implements the workflow engine. It maintains a priority queue of *IJob* interface implementations. Each job such as individual CBF processing for GE scanners implements the *IJob* interface. The *IJob* interface provides the *execute()* method for the core workflow functionality. In addition it provides lifecycle methods such as *cancel()*, *shutdown()* and *cleanup()* and introspection/metadata methods such as *getMumberOfStages*, *getContextAsJSON()*, *getJobFactory()*.

The *cancel()* lifecycle method needs to be implemented for an cancelable job. This method sets a a flag. The logic in *execute()* method needs to check this flag before starting any time consuming substep for timely job cancellation. The *cleanup()* method allows the job to cleanup its temporary files for example. It is called by the job scheduler at the end of the job. The *shutdown()* method is called before permanent removal of the job from job queue.

A job can have human intervention step such CBF job with manual ventricular annotation in CBFDAP. The job scheduler queries the *IJob* interface via the *getNumberOfStages()* method for the number of stages of the particular job. After each stage, the job scheduler waits until the job is resumed. The user interacts with the workflows thru the job management panel of the web interface from which running jobs can be canceled or waiting jobs can be resumed.

If a context is attached to the job (used for user-interaction and/or persistent jobs surviving server restart) *getContextAsJSON* method returns it as string in JSON representation. Jobs that require human intervention can survive server startups. These kind of jobs can remain idle in the system sometimes for weeks till the job owner attends them. The job scheduler persists the job context for each job in the database and uses it to revive human intervention waiting jobs after a server maintenance. To accomplish this it calls *getJobFactory()* method on *IJob* interface which creates a new job instance using the passed in job context information. This method is called on each revival eligible job interrupted during the server shutdown by the job scheduler before resuming them.

The job scheduler receives status updates from the jobs its manages using an event driven mechanism. It implements the *IJobEventListener* interface. Each job type that is interested providing status information sends a *JobEvent* to *IJobEventListener* registered with the managed job by the job scheduler.

The handling of a job submission submitted by an end user is diagrammed in Figure 3.2.

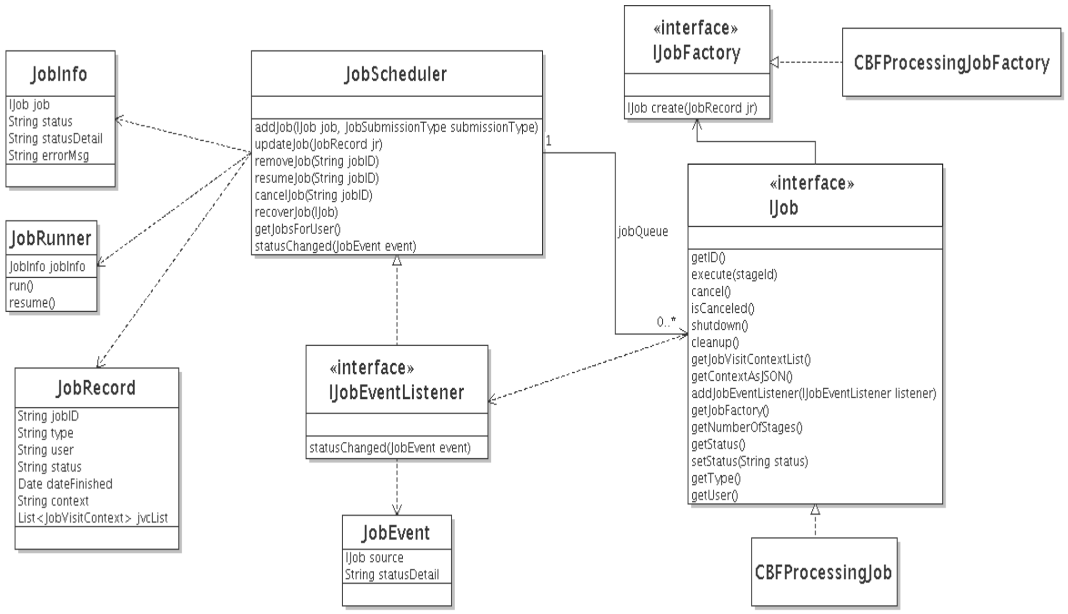

Figure 3.1: Job Scheduler Class Diagram.

3.2 Efficient Multiple CBF Processing Workflow Support

- Each CBF processing workflow is identified by a combination of input options selected by the user during job submission.
- On the file system, for each unique combination a new derived data directory is created with an increasing numeric ID suffix for uniqueness. The directory naming format is derived.¡ID¿
- The combinations of the input options is maintained in the relational database schema and an XML containing this information is created under the derived directory for the corresponding workflow on the file system.
- CBFDAP, CBF workflow handler maintains rules about processing option types, how they map to internal processing steps of the CBF workflows, their outputs and order.
- At each job submission, CBFDAP checks if there are any previous runs. If so, it checks, using the maintained rules, the longest uninterrupted chain of the internal processing steps of the previous runs.
- If the longest uninterrupted chain has nonzero length,the processing uses the output files generated by the matching previous run at the end of the matching chain avoiding unnecessary repetition of already finished internal processing steps.
- For example, if a second worflow on the same scan data also needs fieldmap, the fieldmap corrected BRIKs from the derived data folder of the previous run is be used. Only the non-matching portion of the second workflow is be processed.

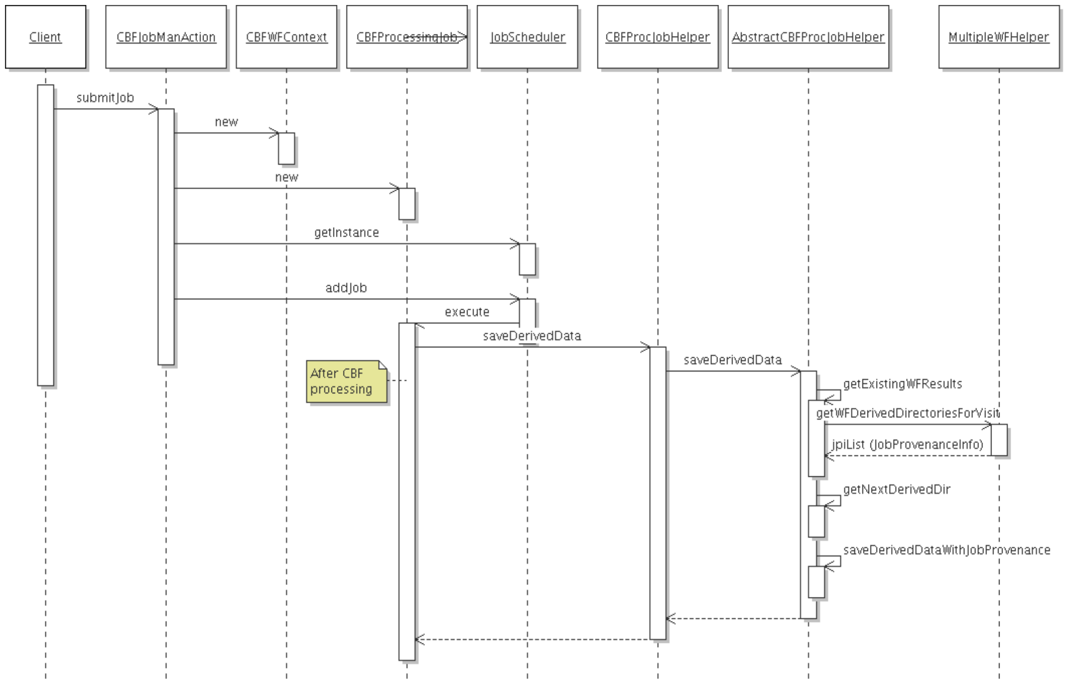

Figure 3.2: CBF Job Processing Sequence Diagram.

Chapter 4

Consolidated Search Services

4.1 CBFDAP Search Framework

The CBFDAP search framework consists of

- a generic web client component that the user interacts with to build a search query.
- a generic server side search mechanism

The user can search on two main types of CBFDAP metadata

- Clinical Assessments
- Provenance Data including quality measures

The search framework currently used for data set selection for the group analysis job (such as ROI analysis) submission. A more detailed sequence of operations for an end user search on CBFDAP is as follows

- Using the web client search component in CBFDAP web app on any of the three locations identified as needing search, the user adds conditions to limit the dataset returned/operated on.
- The web client search component builds an intermediate representation for the conditions and sends it to the server
- The query integrator on the server side splits the conditions into three main queryable types (clinical assessment, provenance, derived data) and sends them to their corresponding query processor components.
- Each query processor converts the conditions into corresponding database queries for the relevant database tables and returns the results back to the query aggregator. Each query processor runs asynchronously.
- The query aggregator combines results coming from the individual query processors also ensuring that all the boolean conditions are satisfied before returning the combined resultset to the user.

The main goal of CBFDAP search framework is to provide an unified, dynamic interface for CBF data set retrieval needs in CBFDAP web app.

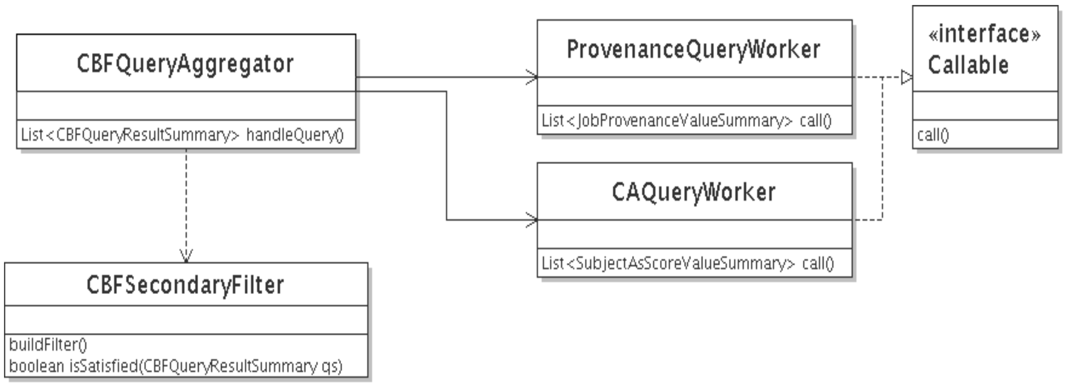
CBFQueryAggregator
List<CBFQueryResultSummary> handleQuery()
ProvenanceQueryWorker
List<JobProvenanceValueSummary> call()
«interface»
Callable
call()
CAQueryWorker
List<SubjectAsScoreValueSummary> call()
CBFSecondaryFilter
buildFilter()
boolean isSatisfied(CBFQueryResultSummary qs)

Figure 4.1: Query Aggregation Class Diagram.

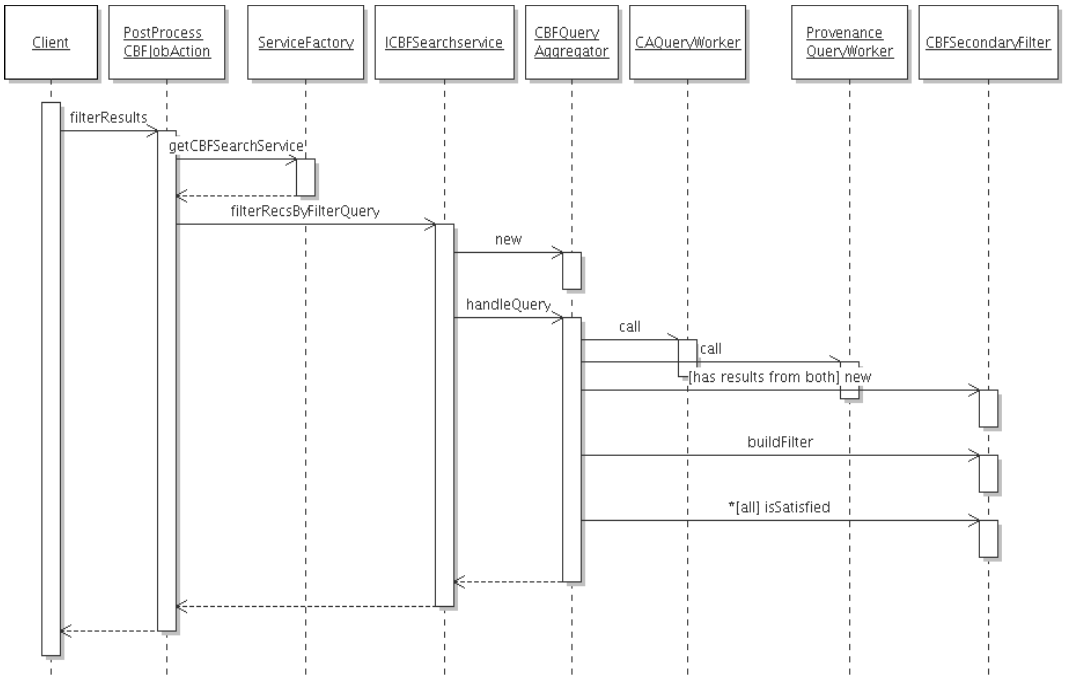

Figure 4.2: Generalized Search Sequence Diagram.

Chapter 5

Data Management

5.1 Assessment Management

5.1.1 Score Values Display Layout Management

clinical.web.helpers.ScoreValuesDisplayHelper prepares display layout beans for the assessment value display as used in *SVResults.jsp* and *SegmentMan.jsp*.

SubjectVisitManagementForm in clinical.web.forms package is the form bean for JSPs *SubjectList.jsp*, *SubjectDetail.jsp*, *VisitMan.jsp*, *SubjectMan.jsp*, *ShowEntries.jsp*, *SegmentMan.jsp*. SubjectVisitManagementForm.getScoreValuesDisplayLayout() uses ScoreValuesDisplayHelper to prepare a *ScoreValuesDisplayLayout* presentation object holding row by row layout for an assessment.

SubjectVisitManagementForm.getScoreValuesDisplayLayout() is stateful, i.e. increments the instance variable *currentAsIdx* every time it is called.

SubjectManagementHelper in clinical.web.action package also uses ScoreValuesDisplayHelper.

5.2 Subject Management

5.2.1 Adding a new subject

A new subject creation is handled by *SubjectAdd.jsp* using the form bean *SubjectVisitManagementForm*. The corresponding Struts action is *SubjectManagementAction*.

5.2.2 Finding/Selecting a subject

In the Find Subjects screen *SubjectSearch.jsp*, there are two possible ways to find a subject. Either you query for the subject you are looking for or select it from the list provided in the 'Find Subjects' screen. The controller for this JSP page is *SubjectManagementAction* with the form bean *SubjectVisitManagementForm*. This page is dispatched from the left menu, also using *SubjectManagementAction* as controller via the action method *showFindSubjectsScreen*.

If you choose a subject from the subjects list and doubleclick on it, the action will be handled by *manageSubjects* method of the controller *SubjectManagementAction*.

If you choose query for subjects option, this action will be handled by *findSubjects* method of *SubjectManagementAction*. The results will shown via the JSP *FoundSubjectList.jsp*. Pressing the *Edit* button next to the desired subject in this screen is also handled by *manageSubjects* method of *SubjectManagementAction*.

5.2.2.1 Details of *manageSubjects* action

In *manageSubjects* action, first, the corresponding record from *nc_humansubject* table is retrieved, then the corresponding visits (if any) are retrieved from the database and sorted by descending date. For the Subject Management screen to be shown, only those visits belonging to the current experiments are set in the form bean. After that the experiment records for the experiments this subject is participating are retrieved from the database.

The visit type is retrieved from database if not cached already using an implementation of *IDBCache* interface gotten via the *ServiceFactory*. IDBCache provides caching support for the mostly static tables of the CBFDAP database. The caching mechanism provided by the implementation uses lazy loading and the cache is not released till the web server/container is restarted. Whenever the nearly static state cached is changed via the web application, the cache is refreshed by using *forceRecache* option in *IDBCache* method calls. Then, the protocol info is retrieved from cache via *IDBCache* interface. Afterwards, the bookkeeping for the studies of the visit to be shown in the 'Subject Management' screen is done. Most of the functionality in *manageSubjects* method is delegated to the helper class *SubjectManagementHelper*. Also the segment information for the visit/study to be displayed is set in the form bean.

In the current implementation, a segment can directly belong to a visit without being associated with a study. Those segments are grouped under the virtual study *Default Study*. Also, the segments are numbered uniquely under a visit meaning that the segments belonging to a study are not numbered starting from one, for each visit they are numbered starting from one.

5.2.3 Subject Management Screen

In subject management screen (*SubjectMan.jsp*, you view subject summary data , his/her latest visit and its segments, navigate between visits etc. The controller for this JSP page is *SubjectManagementAction*. Also visit/study/segment management is spawn from this page. The controller *SubjectManagementAction* also handles visit/study/segment management. The form bean is *SubjectVisitManagementForm*.

5.2.4 Segment Management

SegmentManagementAction is the controller for segment and corresponding assessment management. The assessment management is delegated to other components. It provides segment edit and add functionalities which are persisted to the database. Change segment functionality is is for switching between segments in the same segment management screen. The assessment related functions provided are adding an assessment (first or second entry for double entry) editing any of the assessment entries via delegation to GAME. The first entry can be deleted from the database also using *SegmentManagementAction*. The methods in *SegmentManagementAction* use the corresponding Session Facade interfaces for business logic.

SegmentManagementAction also provides entry points for the reconciliation of the assessment entries, namely preparing view data to be shown on the reconciliation screen and showing a summary of both entries with indication of any missing data in any of the entries as an reconciliation aid.

Each segment contains associated protocol information. The helper class *SubjectManagementHelper* is responsible for gathering protocol information amongst others.

5.3 Experiment Management

Experiment Management involves creation of new experiments, adding new study groups, enrolling/ unenrolling subjects to an experiment. The main page (screen) for experiment management is *Exp-*

Man.jsp. The controller is *ExperimentManagementAction* and the form bean is *ExperimentManagementForm.*

5.3.1 Finding/Selecting an Experiment

The main page for viewing all available experiments and navigating into one is *ExpFind.jsp.* The controller is *ExperimentManagementAction* and the form bean is *ExperimentManagementForm.*

5.3.2 Study Group Management

5.3.2.1 Adding a new Study Group

A new study group data is entered in the form provided by *StudyGroupMan.jsp.* The controller for this page is *StudyGroupManagementAction.java.* The form bean used is *ExperimentManagementForm.*

Chapter 6

Clinical Data Queries

6.1 Assessment Query Builder Wizard

The *SelAs.jsp* is the page where the assessment on which scores the query will built, is selected. The controller is *SelectAssessmentAction*. Also, in this page one of saved query templates can be loaded. The form bean is *AsQueryBuilderForm*.

The *CollectQuery.jsp* is the page where the user constructs a query on the scores from asssessments he/she selected. The controller is *AsQueryAction.java* and the form bean is *AsQueryBuilderForm.java*. The Struts action path */collectquery* dispatches the assessment query to *AsQueryAction* which forwards the results to *SvResults.jsp* page.

AsQueryAction handles query build and submission to one or more databases (*doQuasiMediatedQuery*), going to previous wizard page or saving the query for later usage as a template.

In *AsQueryAction.doQuasiMediatedQuery*, first the user query input is validated, then an operator tree (intermediate representation of the query) which is traversed by visitors to generate an SQL query or XML representation etc, is generated. For the derived data, another operator tree is generated, if there are any derived data related queries. The *MultiSiteQueryHelper* is responsible for preparing queries for multiple databases and sending them to the connected databases in parallel and combining the results. The parallel queries are handled by *MultiSiteQueryWorker* classes, which delegate the query submission to *DebugAssessmentService* or *DerivedDataService* classes, calling either *queryForScores()* or *getSubCorticalValuesForSubjects()* methods.

The *DerivedDataService.queryForScores()* method uses the visitor *MultiSiteAssessmentQueryBuilder* to build the SQL query for the given site.

The query results are prepared by *AsQueryHelper.processQueryResults()* method.

The query results are displayed by *SVresults.jsp* grouped by subject. Subject details drilldown is thru the generated links in *SvResults.jsp* which dispatch the request to action */subvisit* handled by the controller *SubjectVisitAction.java* and the subject details are shown in *SubjectVisit.jsp*.

6.2 Transparent Handling of DBMS SQL differences in complex queries

The query builders in CBFDAP web app use strategy design pattern to separate invariant part of SQL query building from variant part. The variant part is provided by the implementations of *ISQLDialect* interface. The query builders only interact with *ISQLDialect* interface, while the factory method in *ServiceFactory* classes is responsible to provide an apropriate implementation depending on the DBMS type (currently either *OracleSQLDialect* or *PostgresSQLDialect*). For new complex query building tasks for which single table mapped DAOs are inefficient and/or inadequate, you should be using this mechanism to cope with DBMS supported SQL language differences.

Chapter 7

Subject Assessment Management

Assessments are associated with a particular subject, hence they are managed wrt to their corresponding subject. A simplified class diagram for Subject Assessment Management section is provided in Figure 7.1. *ServiceFactory* handles the actual creation of the implementation of *ISubjectAssessmentManagement* based on the database configuration and the database ID provided by the user. The logic of some of the important methods in *ISubjectAssessmentManagement* is summarized below.

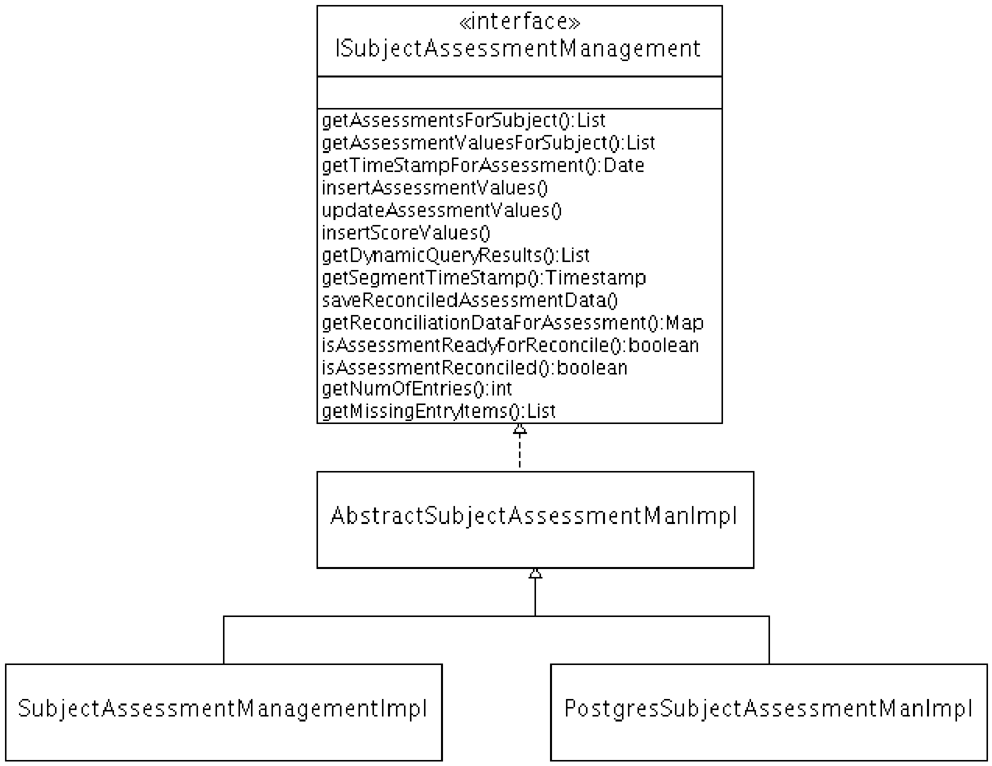
«interface»
ISubjectAssessmentManagement
getAssessmentsForSubject():List
getAssessmentValuesForSubject():List
getTimeStampForAssessment():Date
insertAssessmentValues()
updateAssessmentValues()
insertScoreValues()
getDynamicQueryResults():List
getSegmentTimeStamp():Timestamp
saveReconciledAssessmentData()
getReconciliationDataForAssessment():Map
isAssessmentReadyForReconcile():boolean
isAssessmentReconciled():boolean
getNumOfEntries():int
getMissingEntryItems():List
AbstractSubjectAssessmentManImpl
SubjectAssessmentManagementImpl
PostgresSubjectAssessmentManImpl

Figure 7.1: Simplified class diagram for Subject Assessment Management.

The *insertScoreValues* method, first retrieves the *Storedassessment* record for the assessment of the subject for the given experiment, visit and segment. If the record is not available, it throws an *SubjectAssessmentManagementException*. For each of the provided score values calls the *insertScoreValue* method to insert the score value to corresponding *NC_ASSESSMENTXXXX* table , also handling the missing value(s) if any.

The *insertAssessmentValues* method, looks for a corrsponding *Storedassessment* record in the database, if not creates a new one. It also updates informant information (if necessary) and for each of the provided score values calls the *insertScoreValue* method to insert the score value to corresponding *NC_ASSESSMENTXXXX* table , also handling the missing value(s) if any.

7.1 Reconciliation

For the double entry functionality, data reconciliation functionality is provided by portions of *ISubjectAssessmentManagement* interface and its implementations.

To retrieve data for both unvalidated entries for reconciliation use the *getReconciliationDataForAssessment* method on *ISubjectAssessmentManagement*. This method retrieves the unvalidated score values form the database, gets the question text information to be used as visual clue for the user during reconciliation from the database if the question data is available and creates a lookup table keyed by score name of *ReconScoreValueInfo* objects.

To check for readiness of an assessment for reconciliation, invoke the *isAssessmentReconciled* method. This method , like the rest of the reconciliation related functionality assumes that the score names within a score are unique. This method will return false if any of the two entries have any score values have a missing value without proper indication of a reason for the missing value.

To check if the assessment is alredy reconciled invoke the *isAssessmentReconciled* method.

The reconciliation information is persisted to the database via a call to the *saveReconciledAssessmentData* method. This method only creates a third entry in the corresponding *NC_ASSESSMENTXXXX* tables for the mismatched first and second entries to score the reconciled and validated score value, for other score values with both matching entries, the first entry is tagged as valid by convention.

The controller for reconcilation functionality is *ReconciliationAction*, the form bean is *ReconciliationForm* and the corresponding JSP is *Recon.jsp*.

7.2 GAME

GAME consists of two classes, namely *AssessmentManagementAction.java* and *AssessmentManagementHelper.java*. *AssessmentManagementAction* class is the Struts action handling the screens of any assessment generated by CALM.

The *setFormDataForPage* method sets the fields corresponding for the page to be shown using reflection. The type metadata is gotten from the corresponding database variables info (score information).

7.2.1 AssessmentManagementHelper

The *AssessmentManagementHelper* class is responsible for preparing/querying meta data for the form handling by accessing/modifying the form bean via Java Reflection. The *AssessmentManagementHelper* class is a singleton discovering and keeping a cache of form bean metadata lazily initialized at the first time the singleton is accessed. The discovery and caching mechanism can be summarized as follows:

1. From the package name, determine the full path of the class files for the form beans.

2. For each form bean detected

(a) question the form bean class about its assessment data as stored in the database and form page variable mapping information.

The form bean package is currently set to *clinical.web.game.forms.* All the CALM generated form beans are stored under this package. For each form bean, its metadata is stored in an object of class type *FormBeanInfo*. This object groups metadata per form page. For an online assessment, each page of the paper form is mapped (by convention) to two JSPs (one for the online form and one for Tiles template engine) both generated by CALM and stored under $CLINICAL_HOME/web/pages/assessment directory. For each assessment there is a single form bean generated by CALM and put under $CLINICAL_HOME/src/clinical/web/game/forms directory.

A particular *FormBeanInfo* object holds *PageVariableInfo* objects for online form property name to database score name mapping, *PageQuestionInfo* objects for question related metadata and *MandatoryFieldMetaData* objects to hold metadata associated with mandatory fields in the form. The question types currently supported are as follows;

1. single-answer, single score
2. single-answer, multiple score
3. multiple-answer, single score
4. multiple-answer, multiple score

A *PageVariableInfo* object holds the form property name associated with the corresponding score in the database for the assessment, the corresponding score name as in the database, the page number on which the form property value displayed/retrieved and an optional lookup table for metadata associated with the form property(page variable) as name-value pairs.

A *PageQuestionInfo* object holds the question number as assigned by CALM, question type (single-answer or multiple-answer), list of score name(s) (as in the database) associated with this question, the minimum and maximum number of answers allowed if the question is a multiple-answer one, a lookup table for score name and corresponding ID association used for web form hidden field name generation if the question is a multiple-answer one, and the page number on which the question will be displayed. GAME numbers pages internally starting from one.

A *MandatoryFieldMetaData* object holds the mandatory field name and an optional lookup table for metadata associated with the mandatory field as name-value pairs. One usage scenario is for dynamic dropdown for the clinical rater to associate an SQL query with the mandatory field to dynamically populate the corresponding dropdown.

7.2.2 AssessmentManagementAction

The controller *AssessmentManagementAction* is responsible for the lifecycle management of an online assessment excluding the startup of the online assessment management workflow. When a user selects an assessment from the available assessments dropdown list on Segment Management screen and presses 'Add Assessment' button, the controller *SegmentManagementAction* intercepts it to start the lifecycle. In the *addAssessment* method of *SegmentManagementAction* class, the singleton *AssessmentManagementHelper* is called to retrieve the corresponding form bean name for the selected assessment. Any dynamic dropdowns in the form are populated with the help of *AssessmentManagementHelper*. Currently, only Clinical rater mandatory field dropdown is supported, but a more generic mechanism is not difficult to implement when/if need arises. The *addAssessment* method forwards directly to the controller named *CADispatcherAction* which is responsible to figure out which online form page to show as the next screen. The controller of each JSP page generated by CALM is *AssessmentManagementAction.*

7.2.2.1 Hooking up a new assessment with the GAME

While CALM generates the form bean and the corresponding JSPs for the online assessment, and updates struts-config.cml.template file, it does not hook up the form with *CADispatcherAction*. Currently , you need to go through following steps to hook a new assessment (assuming your clinical assessment is named ca in CALM);

1. in *clinical.web.Constants* class, add a new constant variable TO_CA with value *to_ca*.
2. update switch logic in *clinical.web.actions.CADispatcherAction*
3. In *web/WEB-INF/struts-config.xml.template*, edit action named `<action path="/cadispatcher"`, add a forward tag like

   ```
   <forward name="to_ca" path="/man_ca_Page1.do?action=Display"/>
   ```

   where *man_ca_Page1* is the action path param of the first form (page) of the online assessment.
4. delete *web/WEB-INF/struts-config.xml* to force regeneration by ant from the template file.
5. run ant (to recreate struts-config.xml from the template and compile code also)