J2EE based Web-Interface Development Framework for Clinical Imaging Databases

I. Burak Ozyurt

November 20, 2003

Contents

List of Figures

Chapter 1

Overall Architecture

The UCSD morphometric human imaging database user interface is a three tier J2EE (Java 2 Platform Enterprise Edition) application. The overall architecture is shown in Figure 1.1. It has a (thin) client tier, a middle tier (servlet/JSP based) using Jakarta Struts web framework [Cav02] and an enterprise information source (EIS) tier. The client tier consists of a web browser. The EIS tier is the Oracle database instance and the collection of stored procedures/functions and packages for low level data access functions and optimizing hot spot queries.

The middle tier currently consists of a web tier tier only. For large applications a server component framework like enterprise Java Beans (EJB) provides scalability, however foreseeable scope does not justify an EJB container.

The underlying web application framework used for user interface is Jakarta Struts [Cav02] which is a Model 2 architecture based on model-view-controller (MVC) design pattern [GHJV94]. Struts uses a controller servlet (see front controller design pattern [ACM01] and command design pattern [GHJV94]) to intercept a web request and determine what to display next. It also provides tag libraries for JSPs, validation, internationalization and error handling support.

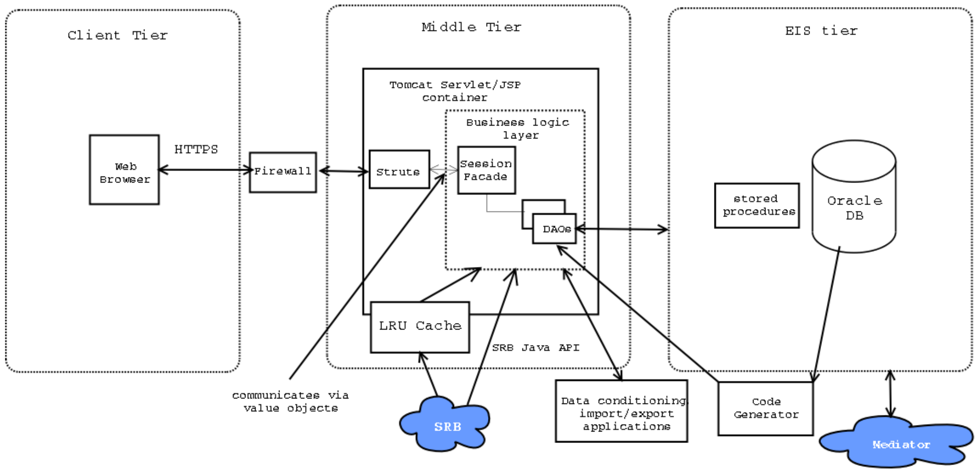
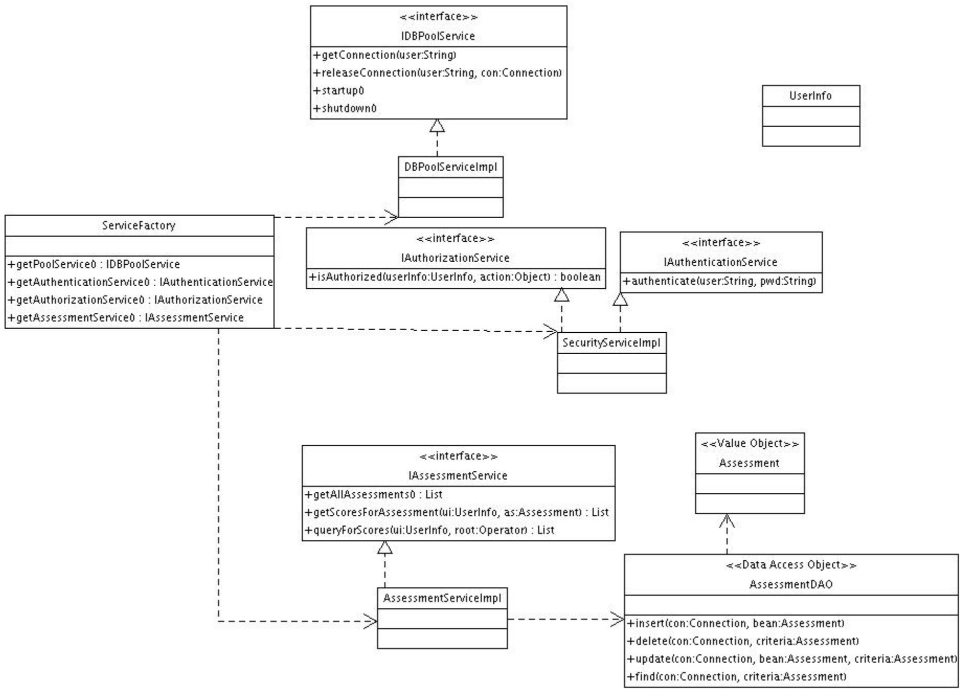


Figure 1.1: High level architecture of UCSD Human Imaging Database User Interface.

The MVC design pattern allows decoupling of presentation logic from the business logic. Here, the business logic is defined as the code manipulating business data (e.g. clinical assessments) relevant to the application. The presentation logic is the code preparing the data provided by the business logic for display to the user of the application. The decoupling provides flexibility in presentation layer selection and development responsibility separation. In the UCSD human imaging database, the business logic is further decoupled by the use of code-generated data access objects (DAO) [ACM01]. The DAOs can be seen as an object-relational mapping, since they map database tables to objects. They provide a data access layer, which is not directly accessed by the presentation layer to reduce inter-layer dependency. The DAOs provide CRUD (create, read update and delete) operations on database table level and are fine grained. The presentation layer communicates with the data access layer through session facades [ACM01], which coordinate operations between multiple DAOs in a workflow and provide a coarse grained simpler interface hiding the internals of the business logic from the presentation layer. The data transfer between presentation and data access layer is via coarse grained transfer objects (value objects) [ACM01]. The business logic layer also serves data import/export and conditioning clients which map subject data to the Oracle tables by doing necessary data cleaning, combining and transformation, which are not part of web user interface. Currently clinical assessment and visit data upload is supported.

Besides the business logic, application level authentication, authorization and database connection pooling services are needed. In order to be able to easily replace these services with different mechanisms, the abstract factory design pattern [GHJV94] is used. In Figure 1.2, the high level class diagram for the server side service framework for UCSD human imaging database in unified modeling language (UML) format [BRJ97] is shown. Here the *ServiceFactory* class creates the classes implementing the security and database connection pooling interfaces. However it returns only the interface to the service requesting client. The interfaces are the only means by which the layers communicate with each other. Even when the implementation of the interfaces change drastically over time, if the interfaces remain same, the layers can talk with each other. To support label security, a non-traditional connection pool is developed allowing named user support.



<<interface>>
IDBPoolService
+getConnection(user:String)
+releaseConnection(user:String, con:Connection)
+startup()
+shutdown()
UserInfo
DBPoolServiceImpl
ServiceFactory
+getPoolService() : IDBPoolService
+getAuthenticationService() : IAuthenticationService
+getAuthorizationService() : IAuthorizationService
+getAssessmentService() : IAssessmentService
<<interface>>
IAuthorizationService
+isAuthorized(userInfo:UserInfo, action:Object) : boolean
<<interface>>
IAuthenticationService
+authenticate(user:String, pwd:String)
SecurityServiceImpl
<<interface>>
IAssessmentService
+getAllAssessments() : List
+getScoresForAssessment(ui:UserInfo, as:Assessment) : List
+queryForScores(ui:UserInfo, root:Operator) : List
<<Value Object>>
Assessment
AssessmentServiceImpl
<<Data Access Object>>
AssessmentDAO
+insert(con:Connection, bean:Assessment)
+delete(con:Connection, criteria:Assessment)
+update(con:Connection, bean:Assessment, criteria:Assessment)
+find(con:Connection, criteria:Assessment)

Figure 1.2: High level class diagram for server side service framework.

Chapter 2

Presentation Layer

2.1 Struts Overview

Struts is a presentation framework for building web application using Java Servlets and JSPs. Struts uses the *JSP Model 2 architecture* which is based upon Model-View-Controller (MVC) design pattern (also known as Observer) [GHJV94]. In the model 2 architecture, the client request is first intercepted by a controller servlet which handles the initial processing of the request and determines which JSP page to display next.

2.2 Struts Controller and Form Validation Interactions

In Struts there is one controller servlet `ActionServlet`, which acts like a command dispatcher. `ActionServlet` delegates the request processing to the corresponding command, an `Action` class which is extendended from `org.apache.struts.action.Action` which has an `execute()` method that needs to be implemented to process the user request and depending on the outcome of the user request processing, needs to determine the next action. The `ActionServlet` determines which action to pick, by reading it from a XML-based configuration file (`struts-config.xml`). The Struts config file used for UCSD Morph BIRN web interface is shown in Appendix B). The actions are declared in the `<action-mappings>` section of the `struts-config.xml` file. Struts passes the submitted HTML form parameters to the `execute()` method via an `ActionForm` which it populates from the received HTTP request. Below is an action declaration from `struts-config.xml` file.

```
<action  path="/selectscores"
    type="clinical.web.actions.SelectScoresAction"
    name="asSelectForm"
    scope="session"
    input="/pages/SelScore_full.jsp">
  <forward name="success" path="/pages/SelectDerived_full.jsp" />
</action>
```

Here `path` is the name of the action, `type` is the fully qualified Java class name for the class that extends `org.apache.struts.action.Action`. The attribute `name` is the name of the form bean associated with this action. The form beans extend `org.apache.struts.action.ActionForm` and are declared in the `<form-beans>` section of the `struts-config.xml`. The attribute `scope` is used to identify the scope in which the form bean is placed. It can be `request` or `session`. The attribute `input` is the application-relative path to the input form to which control should be returned if a validation error is encountered. The `forward` elements are used to declare the possible next pages

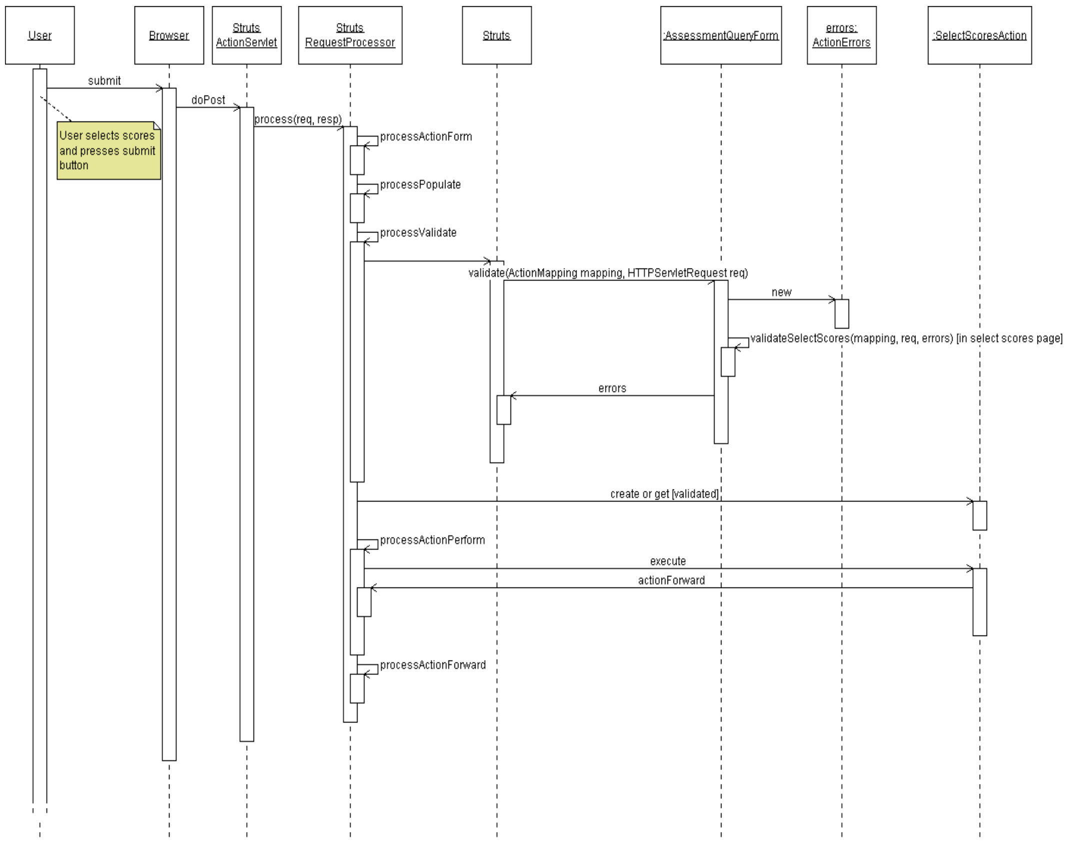


Figure 2.1: User-Struts-Application Interactions for web-tier side form validation and action processing.

this action will return to the user. Here `name` is a unique identifier used to reference this forward in the application. In the `execute()` of the Struts action, this name is used to select and return the ActionForward class. The `path` attribute is an application relative URI to which the control should be forwarded or redirected.

The `org.apache.struts.action.ActionForm` provides form data validation support using Template Method Design pattern [GHJV94]. A class extending the ActionForm can provide implementation of `validate()` method which can check the fields to be set by the user. If any requires data is missing or the user has entered incorrect data, the `validate()` method returns an `ActionErrors` class containing validation errors, which results in forwarding back to the same input form with validation errors shown, so that the user can correct them and resubmit the HTML form. Only after all validation errors are corrected, the `execute()` method on the corresponding `Action` class is called. (See Figure 2.1).

The currently available Struts actions for UCSD Morph-BIRN web interface is shown in Figure 2.2.

2.3 Struts Application Initialization

Struts allows application plugins for application life cycle management support. In Figure 2.3, the initialization of the application is shown. Here the `ServicesPlugin` class implements the

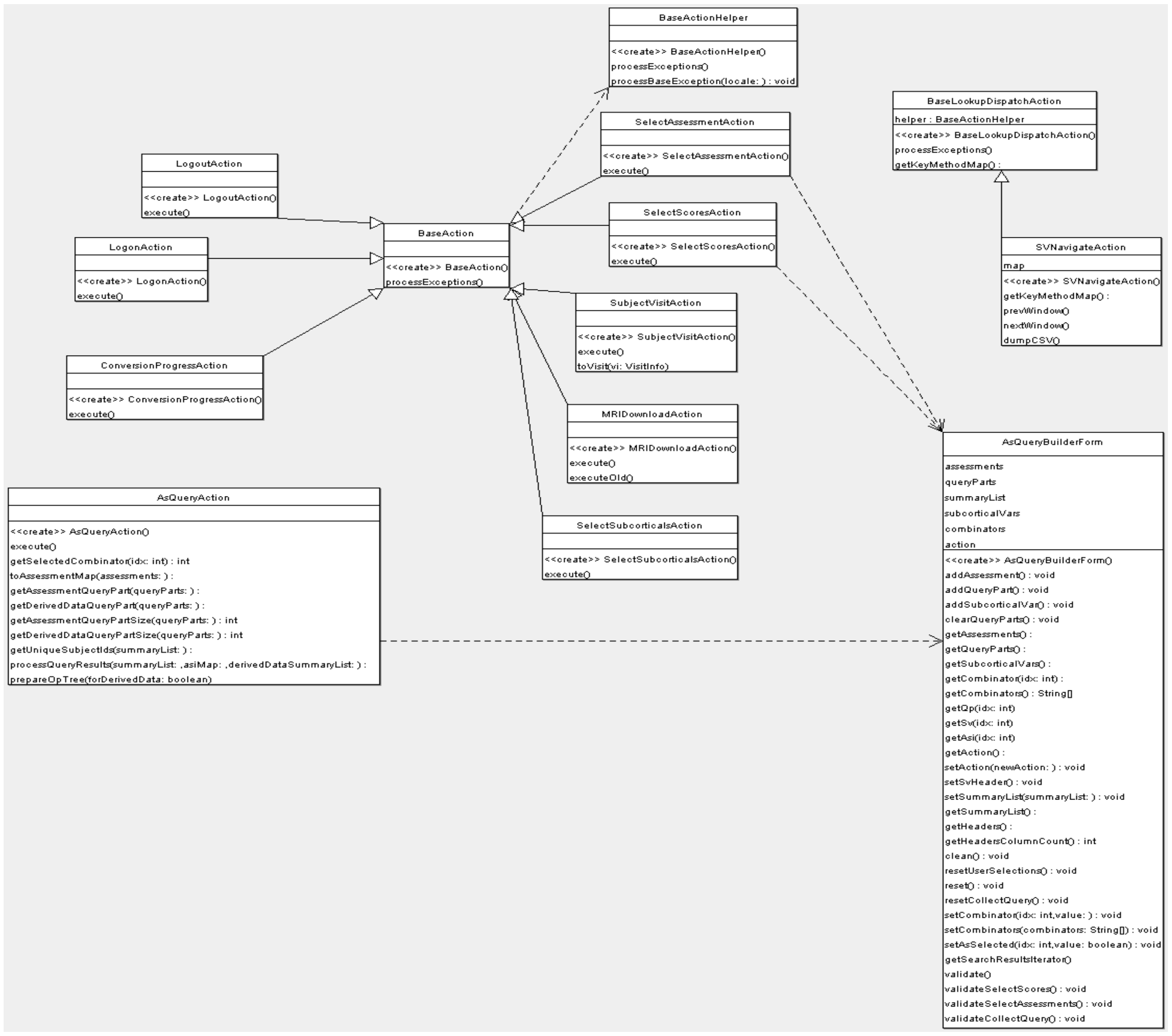

BaseActionHelper
<<create>> BaseActionHelper()
processExceptions()
processBaseException(locale:) : void
BaseLookupDispatchAction
helper : BaseActionHelper
<<create>> BaseLookupDispatchAction()
processExceptions()
getKeyMethodMap() :
SelectAssessmentAction
<<create>> SelectAssessmentAction()
execute()
LogoutAction
<<create>> LogoutAction()
execute()
SelectScoresAction
<<create>> SelectScoresAction()
execute()
BaseAction
<<create>> BaseAction()
processExceptions()
LogonAction
<<create>> LogonAction()
execute()
SVNavigateAction
map
<<create>> SVNavigateAction()
getKeyMethodMap() :
prevWindow()
nextWindow()
dumpCSV()
SubjectVisitAction
<<create>> SubjectVisitAction()
execute()
toVisit(vi: VisitInfo)
ConversionProgressAction
<<create>> ConversionProgressAction()
execute()
MRIDownloadAction
<<create>> MRIDownloadAction()
execute()
executeOld()
AsQueryBuilderForm
assessments
queryParts
summaryList
subcorticalVars
combinators
action
<<create>> AsQueryBuilderForm()
addAssessment() : void
addQueryPart() : void
addSubcorticalVar() : void
clearQueryParts() : void
getAssessments() :
getQueryParts() :
getSubcorticalVars() :
getCombinator(idx: int) :
getCombinators() : String[]
getQp(idx: int)
getSv(idx: int)
getAsi(idx: int)
getAction() :
setAction(newAction:) : void
setSvHeader() : void
setSummaryList(summaryList:) : void
getSummaryList() :
getHeaders() :
getHeadersColumnCount() : int
clean() : void
resetUserSelections() : void
reset() : void
resetCollectQuery() : void
setCombinator(idx: int,value:) : void
setCombinators(combinators: String[]) : void
setAsSelected(idx: int,value: boolean) : void
getSearchResultsIterator()
validate()
validateSelectScores() : void
validateSelectAssessments() : void
validateCollectQuery() : void
AsQueryAction
<<create>> AsQueryAction()
execute()
getSelectedCombinator(idx: int) : int
toAssessmentMap(assessments:) :
getAssessmentQueryPart(queryParts:) :
getDerivedDataQueryPart(queryParts:) :
getAssessmentQueryPartSize(queryParts:) : int
getDerivedDataQueryPartSize(queryParts:) : int
getUniqueSubjectIds(summaryList:) :
processQueryResults(summaryList: ,asiMap: ,derivedDataSummaryList:) :
prepareOpTree(forDerivedData: boolean)
SelectSubcorticalsAction
<<create>> SelectSubcorticalsAction()
execute()

Figure 2.2: Struts actions for UCSD web interface.

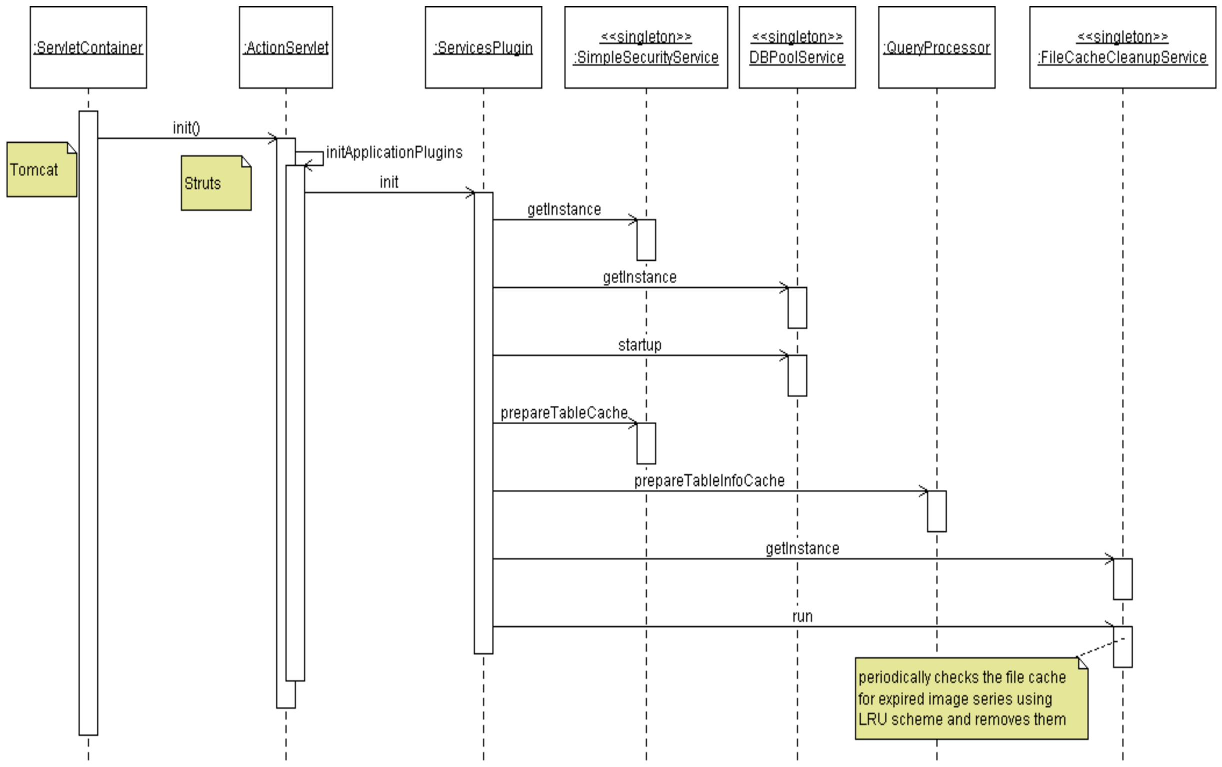

:ServletContainer
:ActionServlet
:ServicesPlugin
<<singleton>> :SimpleSecurityService
<<singleton>> DBPoolService
:QueryProcessor
<<singleton>> :FileCacheCleanupService
init()
Tomcat
initApplicationPlugins
Struts
init
getInstance
getInstance
startup
prepareTableCache
prepareTableInfoCache
getInstance
run
periodically checks the file cache for expired image series using LRU scheme and removes them

Figure 2.3: Servlet container-Struts initialization sequence diagram.

`org.apache.struts.action.Plugin` interface. During ServletContainer initialization (here Apache Tomcat), Struts `ActionServlet init()` method is called, which in turn initializes on the application plugins. The `ServicesPlugin.init()` method, gets the `SimpleSecurityService` and `DBPoolService` singletons, which actually creates them. A singleton ensures that there is only single instance of that class can be instantiated and provides global point of access [GHJV94]. `DBPoolService.startup()` initializes the database connection pool using the `SimpleSecurityService` to get the named database users. `SimpleSecurityService.prepareTableCache()` caches the tables and views available to the admin user, which is used to select the allowed view of the database table in case one or columns contain sensitive material for the user. `QueryProcessor.prepareTableInfoCache()` method caches table column names used for SQL column name to value object property name matching. Last, `FileCacheCleanupService` singleton is initialized and cache cleanup thread is started, which periodically checks the local image series cache using least recently used (LRU) cache eviction scheme.

Chapter 3

Session Facade Layer

Session Facade is a J2EE design pattern based on Facade design pattern [GHJV94] which defines a higher-level interface that makes the subsystem easier to use. The goal is to provide a coarse grained, simple interface to the presentation code, to ensure separation of business logic from the presentation logic and prevent tight coupling of presentation layer with business logic which is a common problem with many web application development methods based on procedural scripting languages. The loose coupling of presentation layer from the business logic facilitates paralell development, allows the server-side to serve different kind of clients without any change. In UCSD Morph BIRN web interface, the business logic is accessed through well-defined interfaces which are the access points for the presentation layer to the session facade layer. The implementation of the interfaces can change over time and can vary drastically from the previous implementations. Since the interface provide a contract between presentation layer and the session facade layer, if the new implementation does not break the contract the presentation layer will work with new implementation without any change. In Figure 3.1, the class diagram for Session Facade Layer interfaces and their implementations are shown for the UCSD Morph BIRN web interface. The implementations directly access data access layer and coordinate between multiple data access objects, query building logic and auxiliary services like database connection pooling, security (authorization).

In Figure 3.2, the sequence diagram for assessment score selection portion of the assessment query wizard is shown. Here the Struts framework is shown as the initiator, which is triggered by a user clicking *Continue* button on the Assessment Score Selection Form. Struts finds the corresponding action for the request from its `ActionMapping` table and invokes the method `execute()` on it. In `execute()`, the Struts populated AssessmentQueryForm is retrieved from the user's session and the selected scores are recorded. Then, using `ServiceFactory`, the session facade interface for the derived data processing is retrieved (`IDerivedDataService`). On the interface, `getAllSubcorticalVariables()` is called, which internally calls data access layer to retrieve derived data variable information from the database. If successful, the `execute()` method returns the forward information for the following page.

3.1 Data Maintenance Example - Subject/Visit Management

For the prospective studies, clinical data needs to br input to the database by means of a user interface, preferably a low maintenance thin client like a web browser. Data maintenance involves data creation, update and deletion, by multiple (possibly) concurrent users where the ensuring of the data integrity can be challenging.

In Figures 3.3 and 3.4, two sequence diagrams are given to show the session facade and lower level classes providing data maintenance support mainlt subject/visit maintenance. A subject

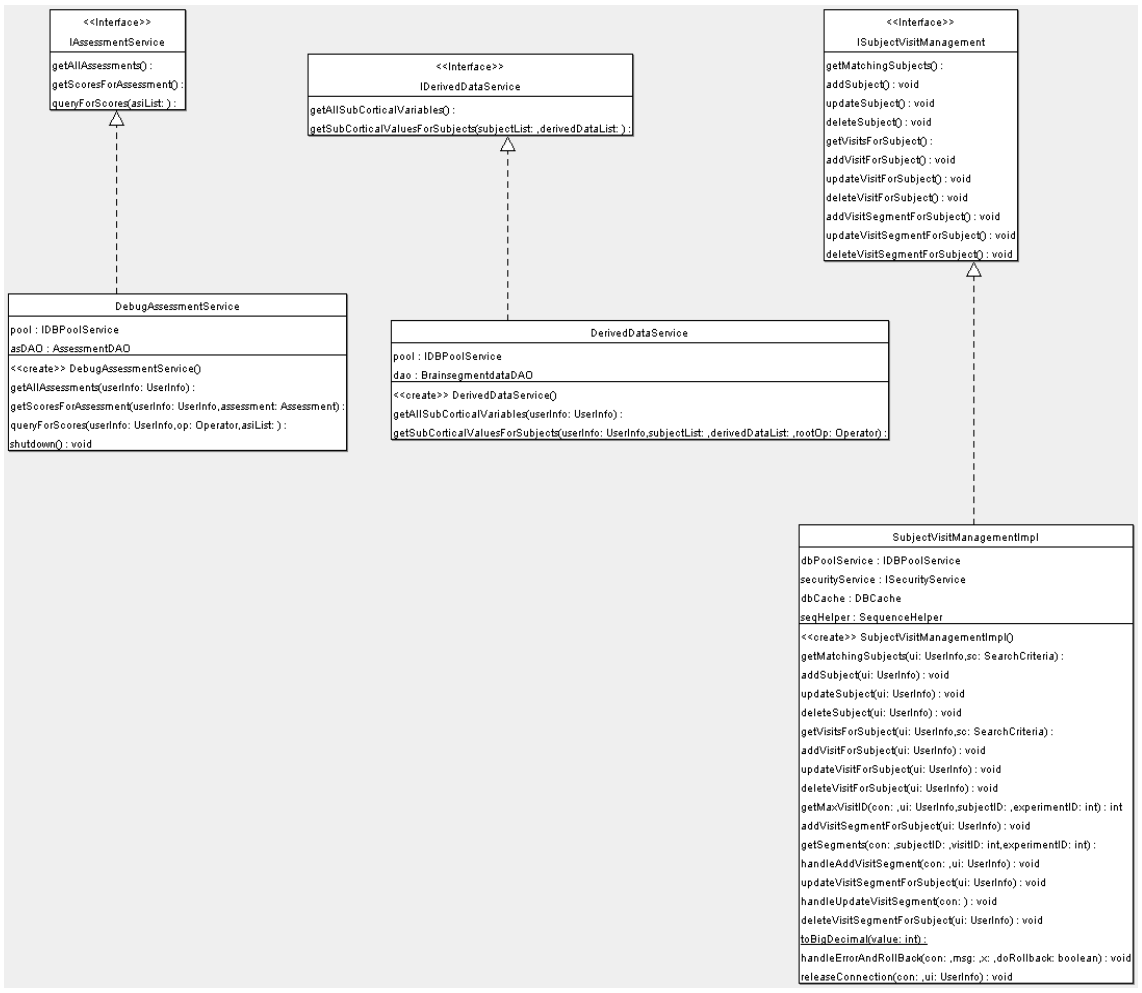

<<Interface>>
IAssessmentService
getAllAssessments() :
getScoresForAssessment() :
queryForScores(asiList:) :
<<Interface>>
IDerivedDataService
getAllSubCorticalVariables() :
getSubCorticalValuesForSubjects(subjectList: ,derivedDataList:) :
<<Interface>>
ISubjectVisitManagement
getMatchingSubjects() :
addSubject() : void
updateSubject() : void
deleteSubject() : void
getVisitsForSubject() :
addVisitForSubject() : void
updateVisitForSubject() : void
deleteVisitForSubject() : void
addVisitSegmentForSubject() : void
updateVisitSegmentForSubject() : void
deleteVisitSegmentForSubject() : void
DebugAssessmentService
pool : IDBPoolService
asDAO : AssessmentDAO
<<create>> DebugAssessmentService()
getAllAssessments(userInfo: UserInfo) :
getScoresForAssessment(userInfo: UserInfo,assessment: Assessment) :
queryForScores(userInfo: UserInfo,op: Operator,asiList:) :
shutdown() : void
DerivedDataService
pool : IDBPoolService
dao : BrainsegmentdataDAO
<<create>> DerivedDataService()
getAllSubCorticalVariables(userInfo: UserInfo) :
getSubCorticalValuesForSubjects(userInfo: UserInfo,subjectList: ,derivedDataList: ,rootOp: Operator) :
SubjectVisitManagementImpl
dbPoolService : IDBPoolService
securityService : ISecurityService
dbCache : DBCache
seqHelper : SequenceHelper
<<create>> SubjectVisitManagementImpl()
getMatchingSubjects(ui: UserInfo,sc: SearchCriteria) :
addSubject(ui: UserInfo) : void
updateSubject(ui: UserInfo) : void
deleteSubject(ui: UserInfo) : void
getVisitsForSubject(ui: UserInfo,sc: SearchCriteria) :
addVisitForSubject(ui: UserInfo) : void
updateVisitForSubject(ui: UserInfo) : void
deleteVisitForSubject(ui: UserInfo) : void
getMaxVisitID(con: ,ui: UserInfo,subjectID: ,experimentID: int) : int
addVisitSegmentForSubject(ui: UserInfo) : void
getSegments(con: ,subjectID: ,visitID: int,experimentID: int) :
handleAddVisitSegment(con: ,ui: UserInfo) : void
updateVisitSegmentForSubject(ui: UserInfo) : void
handleUpdateVisitSegment(con:) : void
deleteVisitSegmentForSubject(ui: UserInfo) : void
toBigDecimal(value: int) :
handleErrorAndRollBack(con: ,msg: ,x: ,doRollback: boolean) : void
releaseConnection(con: ,ui: UserInfo) : void

Figure 3.1: Class Diagram for Session Facade Layer interfaces and their implementations.

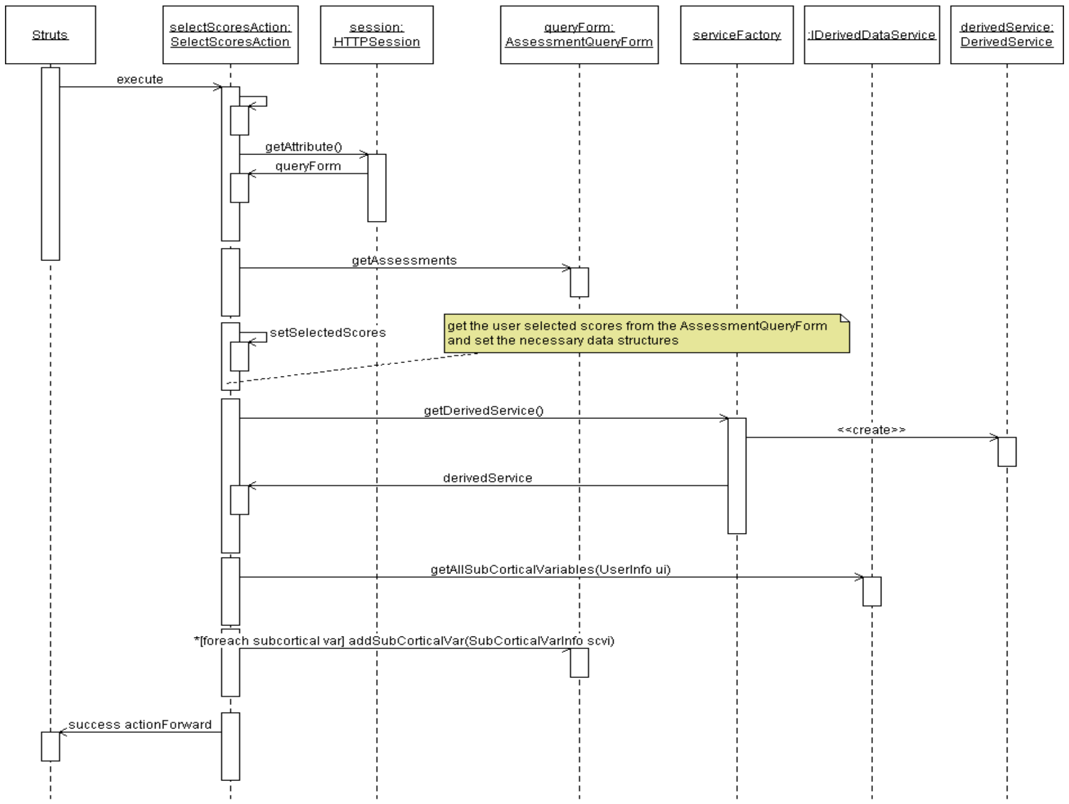

Struts
selectScoresAction: SelectScoresAction
session: HTTPSession
queryForm: AssessmentQueryForm
serviceFactory
:IDerivedDataService
derivedService: DerivedService
execute
getAttribute()
queryForm
getAssessments
setSelectedScores
get the user selected scores from the AssessmentQueryForm and set the necessary data structures
getDerivedService()
<<create>>
derivedService
getAllSubCorticalVariables(UserInfo ui)
*[foreach subcortical var] addSubCorticalVar(SubCorticalVarInfo scvi)
success actionForward

Figure 3.2: Selecting a score Struts action sequence diagram.

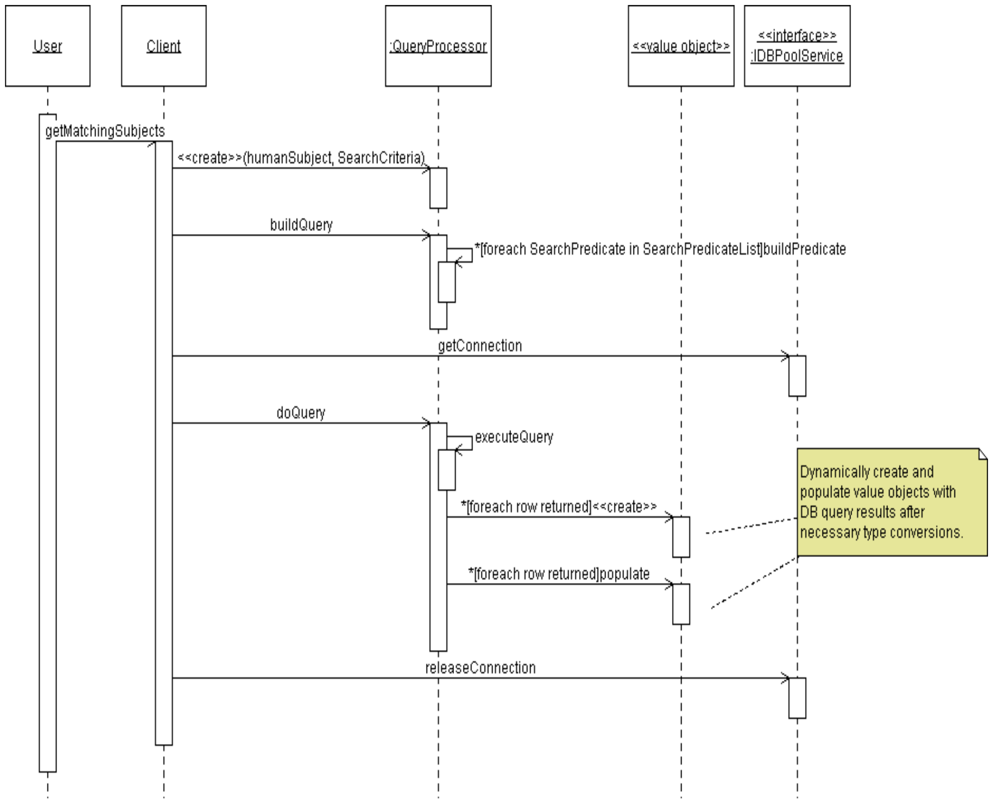

User
Client
:QueryProcessor
<<value object>>
<<interface>> :IDBPoolService
getMatchingSubjects
<<create>>(humanSubject, SearchCriteria)
buildQuery
*[foreach SearchPredicate in SearchPredicateList]buildPredicate
getConnection
doQuery
executeQuery
*[foreach row returned]<<create>>
*[foreach row returned]populate
releaseConnection
Dynamically create and populate value objects with DB query results after necessary type conversions.

Figure 3.3: Get matching subjects sequence diagram.

has zero or more visits. Each visit can have one or more segments. In Figure 3.3, A user sends `getMatchingSubjects` message to the client (here session facade) passing a `SearchCriteria` object containing a `SearchPredicateList` object which contains zero or more `SearchPredicate` objects. The client creates a `QueryProcessor` object, which is a generic SQL query builder and processor for single table queries. It relies on the one-to-one relationship between a value object and its corresponding table. `QueryProcessor` requires a search predicate list and the class type of the value object for which a query will be build and sent to the database server. Then it calls `buildQuery()` on `QueryProcessor` to build the necessary dynamic query iterating thru the `SearchPredicateList`. Then `doQuery()` is called to send the query to the database and return the result set converted to value objects. `QueryProcessor` is mainly used to build more complex queries than the Data Access Object `find()` method provides.

Once a subject is selected, a new visit can be added as shown in Figure 3.4. When the client receives `addVisitForSubject` signal, it gets from ServiceFactory the `ISubjectVisitManagement` interface which is the session facade for subject/visit data maintenance. It creates and populates a `Visit` value object and a `VisitSegment` value object and adds it to the `Visit` object. Then, it calls `ISubjectVisitManagement.addVisitForSubject`, which internally gets a database connection from the connection pool for the serviced user, creates an `Expcomponent` value object and populates it from the `Visit` object. It sets `Expcomponent` uniqueID using the `SequenceHelper` object, creates a `ExpcomponentDAO` object and calls `ExpcomponentDAO.insert()`. After that, it iterates through the segments and adds them to the database by calling internal `handleAddVisitSegment()` method. Then it commits the transaction and releases the connection back to the connection pool.

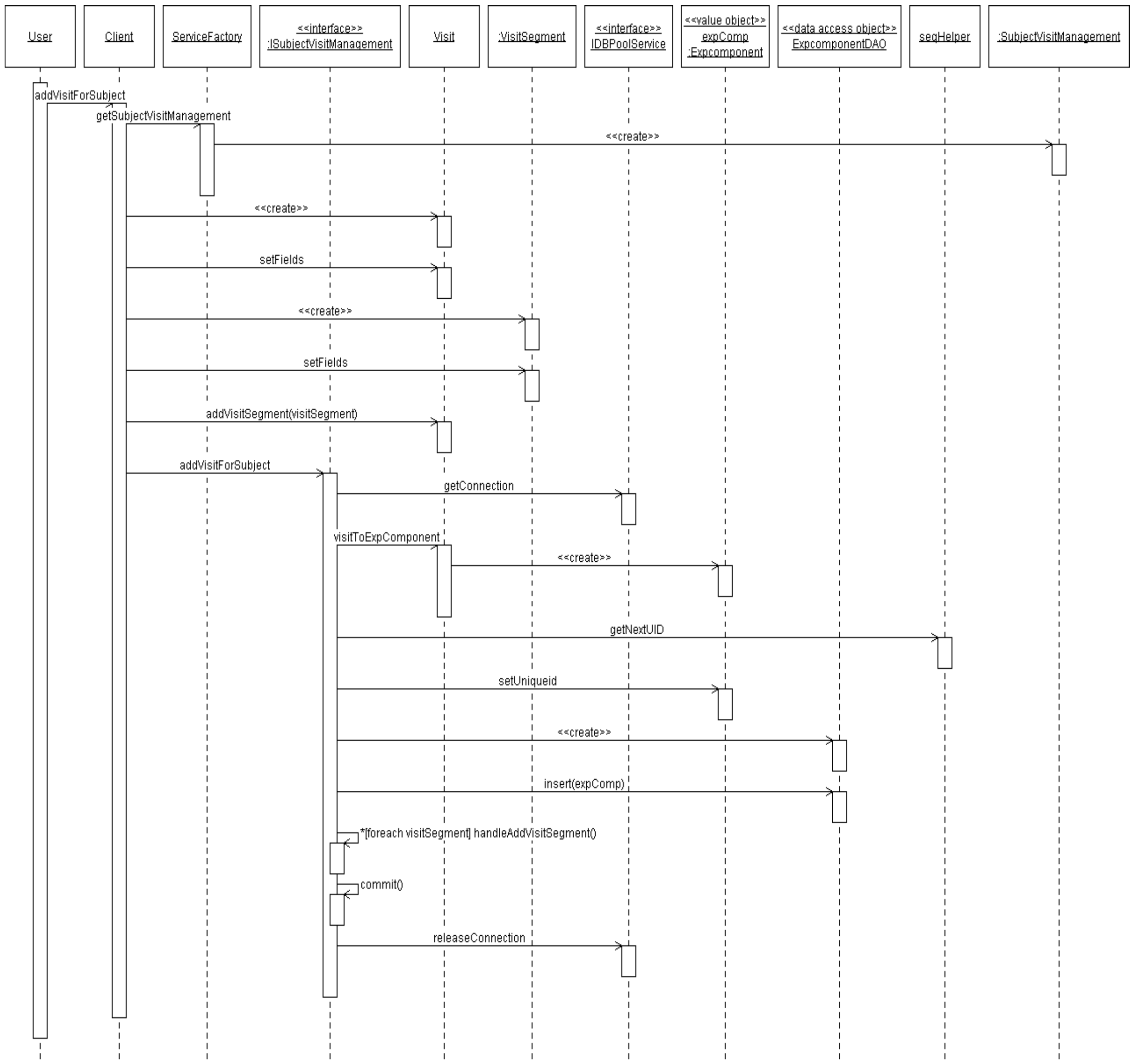


Figure 3.4: Adding a visit for a subject sequence diagram.

Chapter 4

Data Access Layer

4.1 Object-to-Relational Mapping

While relational database constructs seem similar to object-oriented contstructs, on the surface, there are many important differences between them which make transitioning from the relational word to the object world challenging. Especially proper mapping of associations and inheritance is difficult. There are many commercial and open source O/R Mapping tools available, each of which comes with its own prerequistes, limitations.

For maximum flexibility and minimum performance impact, UCSD Morph BIRN Web Interface uses a simple one-to-one mapping (each table is mapped to a class) with code generated Data Access Objects which are mainly used for data manipulation and complex queries with large result sets are handled through direct SQL and stored procedures for efficiency.

4.1.1 Data Access Objects

Data access objects abstract and encapsulate the access to the data source and provide a uniform interface to the upstream layers, which facilitates the portability of the application. A data access object acts like an adapter bwtween data source (e.g. database, directory service, XML files etc.) and the component using the data source. For UCSD Morph BIRN Web Interface, the data source is an Oracle database and its communication language is SQL. Here, the DAO hides the SQL calls from the DAO client and provides an object oriented, uniform interface. Since SQL data types and Java data types are different, the DAOs need to do the necessary type conversions between SQL and Java data types.

A typical DAO, provides the following interface to its clients

```
public void insert(Connection con, <Value Object> bean)
public List find(Connection con, <Value Object> criteria)
public void update(Connection con, <Value Object> bean,
                   <Value Object> criteria)
public void delete(Connection con, <Value Object> criteria)
```

Thus, each DAO provides CRUD (Create/Read/Update/Delete) operations for its corresponding schema object (table or view). A **value object** is a Java bean used by the corresponding DAO and different software layers to transfer coarse grained data between layers. There is one value object class for each schema object where the properties match the columns of the schema object. The `insert()` method takes a value object and a database connection as input, and creates an SQL INSERT statement using the property values in the value object and inserts it into the database.

Please note that `insert()`, `update()` and `delete`() methods do not commit their changes to the database to allow maximum user control for transaction demarcation.

The `find()` method takes a value object as search criteria and a database connection as input. It uses the user set properties on the value object to create the `WHERE` clause of the SELECT statement. Because of the structure of the value object the predicates in the where caluse can be only equalities. To retrieve all the rows in the corresponding table pass an empty value object as criteria to `insert()`. The retrieved database table rows are converted to a corresponding value object and returned as a list.

The `update()` method takes a value object as the data to be updated, an another value object as search criteria and a database connection. It creates an `INSERT` statement and does the necessary database update.

The `delete()` method takes a value object as search criteria to find the rows to be deleted and a database connection. It generates the corresponding `SQL DELETE` statement and deletes the corresponding rows from the database.

4.2 Code Generation

Usually most of the code of an database application is dedicated to data source access and manipulation, which has an inherent structure suitable, after generalization, to code generation. The code generation increases productivity and decreases maintenance cost for adjusting the application code to the database schema changes, since after a schema change, the data access layer can regenerated. A GUI tool is provided (See Appendix C), to generate DAOs, value objects and EJB primary keys from the database schema. The tool does not override custom code in the generated source code given that the code is put inside the specified delimiters.

Chapter 5

Assessment Query Builder

The UCSD human imaging database web application provides a query builder wizard for clinical assessment queries. Figure 5.1 shows a high level UML sequence diagram illustrating the steps involved in the building and execution of a clinical assessment query. The first step is entering the search criteria. Through a set of sequential web pages, the user selects the assessments and the scores of interest, and then the system shows the list of possible search parameters based on these selections. The user enters his/her search criteria and presses the query button. The Struts controller servlet intercepts the requests and selects the `AsQueryAction` object to process the request and calls the method execute() on it. `AsQueryAction` interacts with the business logic layer via the `ServiceFactory` to get the assessment business service `IAssessmentService`. It creates a logical operator tree (an intermediate representation for the final SQL query) and passes its root to the method queryForScores() in `IAssessmentService`, which, in turn, creates an `AssessmentQueryBuilder` object to build the SQL query to be sent to the database. Here the visitor design pattern [GHJV94] is used. The `AssessmentQueryBuilder` visit() method recursively visits every node in the logical operator tree and builds the inner and outer SQL queries, which are combined and sent to the database. The results are returned to the `AsQueryAction` which in turn passes the query results to the query result page for rendering. The user can navigate through the search results or export the results in CSV format to be used with SPSS or other statistical packages for statistical analysis.

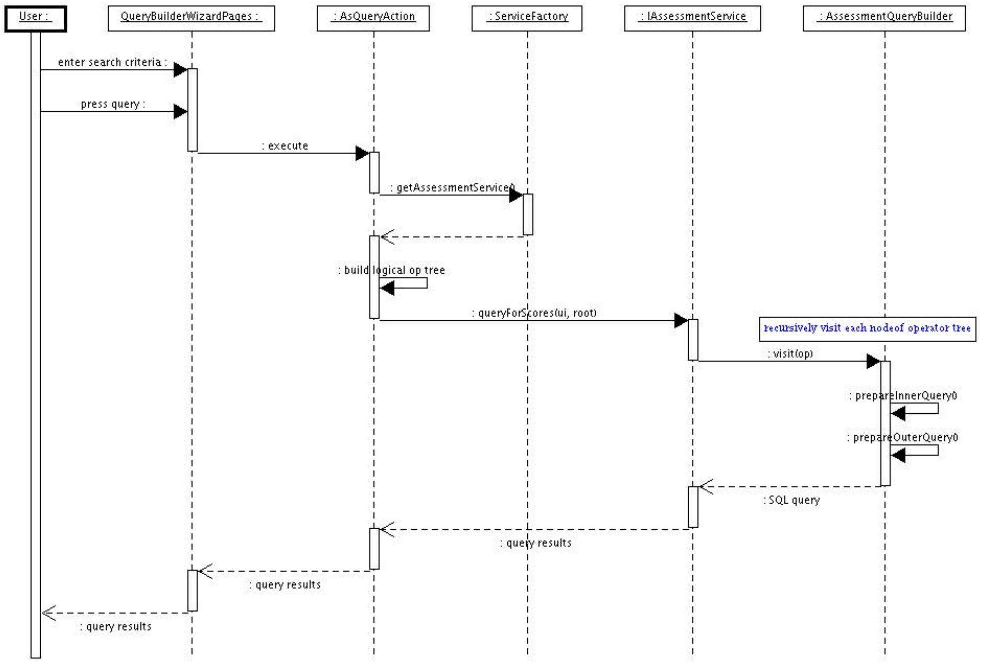


Figure 5.1: High level sequence diagram for assessment query building use case.

Chapter 6

SRB-Database interaction

The user can drill-down into the visit and imaging information of a specific subject from the search query results. Besides subject visit information, there is also image preview and retrieval form SRB (See Figure 1.1). The SRB URIs are retrieved from the Oracle database and an individual slice of DICOM image is retrieved via Jargon (SRB Java API) and converted to jpeg. The whole image series is retrieved using bulk unload Scommand for efficiency reasons. To increase response time under heavy load, a least recently used (LRU) file cache is used in the middle tier server. 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 lock mechanism is simulated with file locks. There is a cache cleanup thread in the middle tier that wakes up periodically and checks the cache for size and age and cleans up the least recently accessed image series. A cache hit is an order of magnitude faster than a new request. A new request can take up-to 40 seconds to process, including DICOM to AFNI conversion and data streaming to the client.

The image conversion object interactions are shown in Figure 6.1. The object interactions for the image series retrieval and DICOM to AFNI format conversion is shown in Figure 6.2.

Screen shots for a typical usage scenario of the web interface are shown in Figures 6.3,6.4,6.5,6.6,6.7.

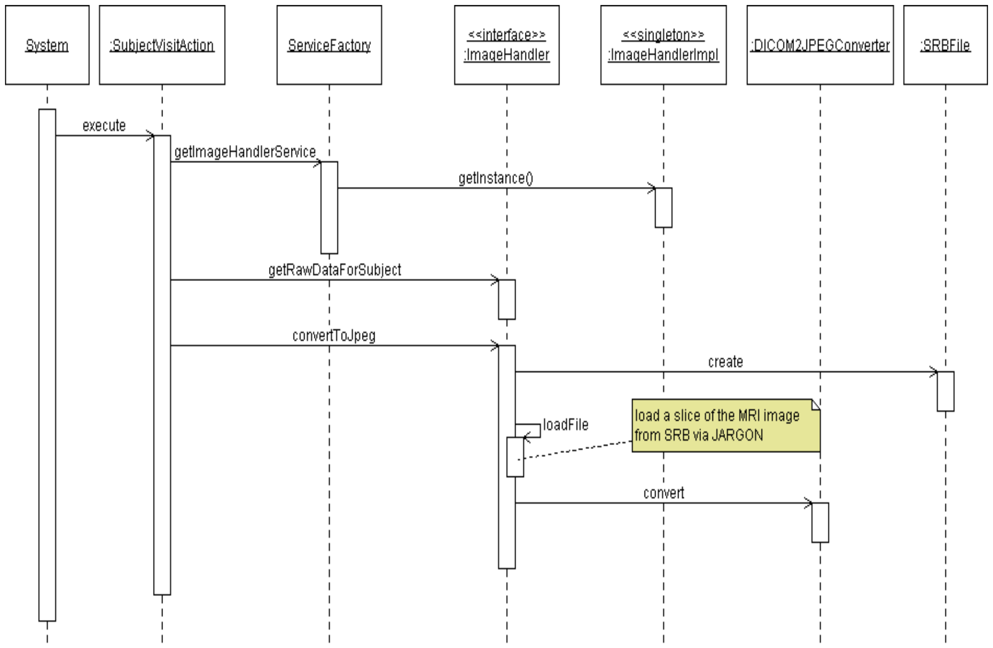


Figure 6.1: SRB-JARGON based image data retrieval and on-the-fly image conversion sequence diagram.

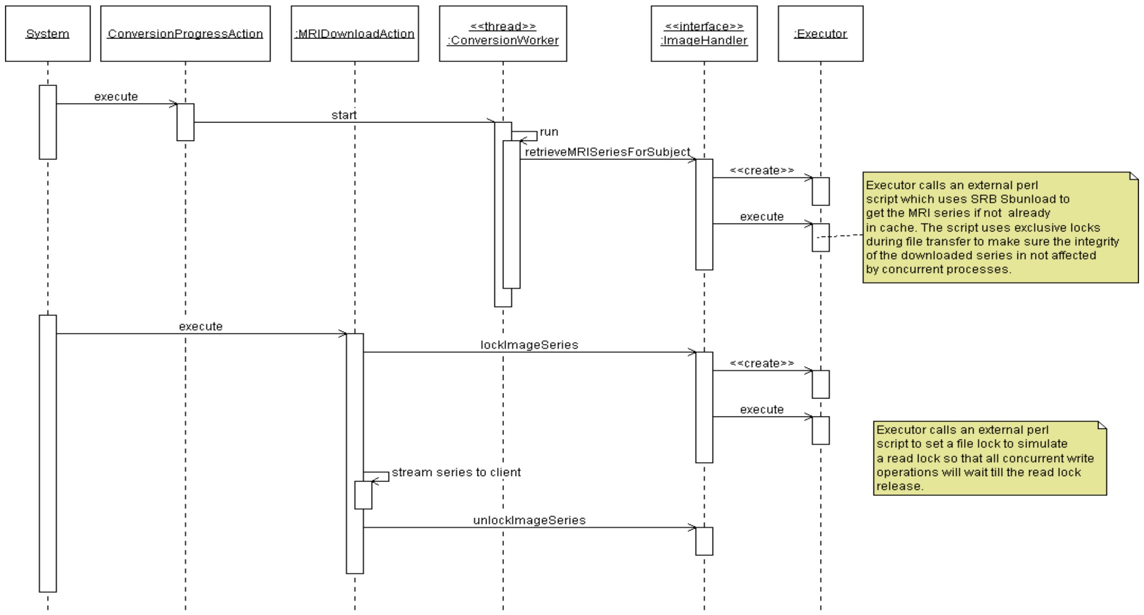
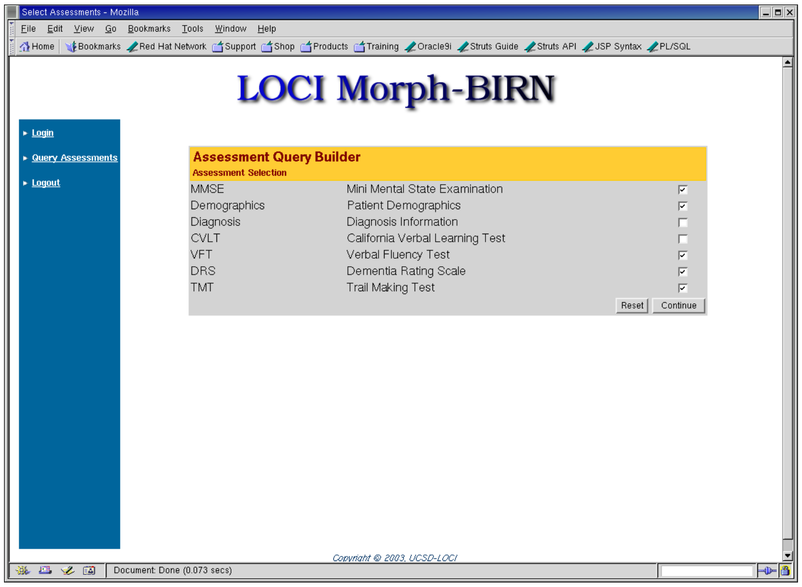

System
ConversionProgressAction
:MRIDownloadAction
<<thread>> :ConversionWorker
<<interface>> :ImageHandler
:Executor
execute
start
run
retrieveMRISeriesForSubject
<<create>>
execute
Executor calls an external perl script which uses SRB Sbunload to get the MRI series if not already in cache. The script uses exclusive locks during file transfer to make sure the integrity of the downloaded series in not affected by concurrent processes.
execute
lockImageSeries
<<create>>
execute
Executor calls an external perl script to set a file lock to simulate a read lock so that all concurrent write operations will wait till the read lock release.
stream series to client
unlockImageSeries

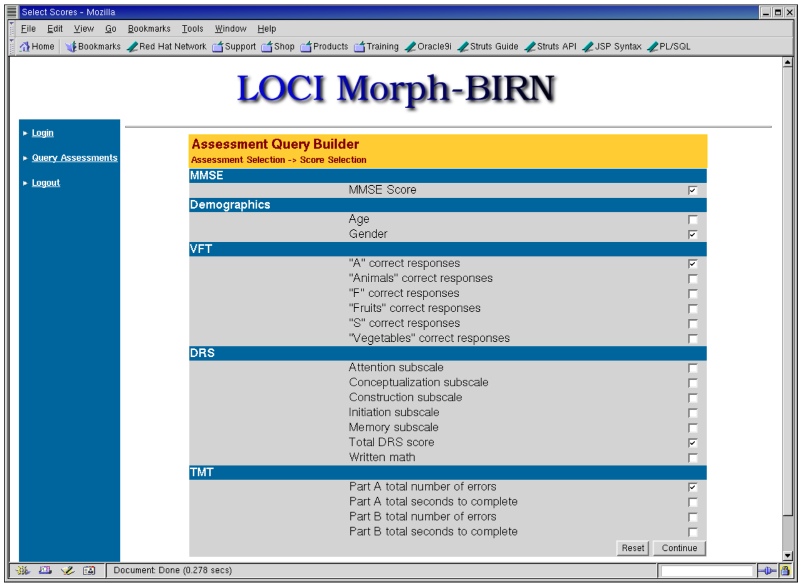
Figure 6.2: SRB Scommand based image series download and AFNI conversion sequence diagram.


Select Assessments - Mozilla
File Edit View Go Bookmarks Tools Window Help
Home Bookmarks Red Hat Network Support Shop Products Training Oracle9i Struts Guide Struts API JSP Syntax PL/SQL
LOCI Morph-BIRN
Login
Query Assessments
Logout
Reset
Continue
Copyright © 2003, UCSD-LOCI
Document: Done (0.073 secs)

Assessment Query Builder
Assessment Selection

MMSE	Mini Mental State Examination	✓
Demographics	Patient Demographics	✓
Diagnosis	Diagnosis Information	☐
CVLT	California Verbal Learning Test	☐
VFT	Verbal Fluency Test	✓
DRS	Dementia Rating Scale	✓
TMT	Trail Making Test	✓

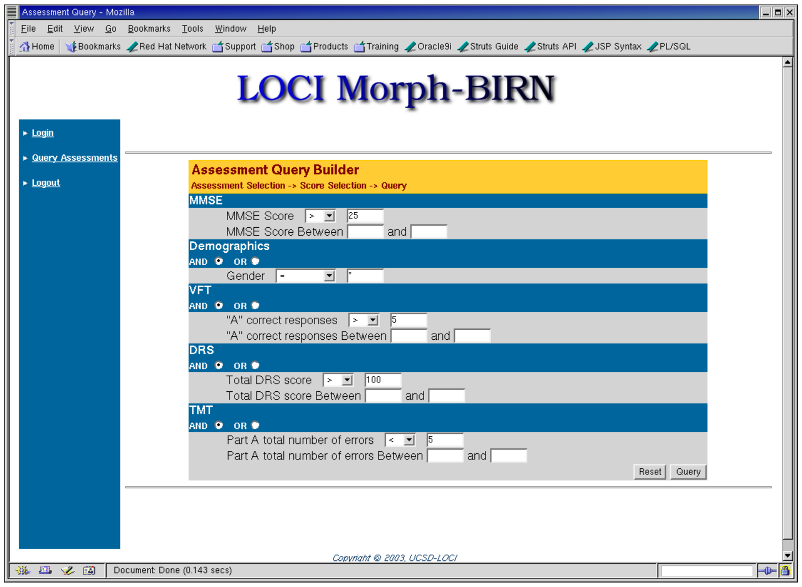
Figure 6.3: Clinical Assessment Selection Screen.


Select Scores - Mozilla
File Edit View Go Bookmarks Tools Window Help
Home Bookmarks Red Hat Network Support Shop Products Training Oracle9i Struts Guide Struts API JSP Syntax PL/SQL
LOCI Morph-BIRN
Login
Query Assessments
Logout
Reset
Continue
Document: Done (0.278 secs)

Assessment Query Builder
Assessment Selection -> Score Selection

MMSE		
	MMSE Score	✓
Demographics		
	Age	☐
	Gender	✓
VFT		
	"A" correct responses	✓
	"Animals" correct responses	☐
	"F" correct responses	☐
	"Fruits" correct responses	☐
	"S" correct responses	☐
	"Vegetables" correct responses	☐
DRS		
	Attention subscale	☐
	Conceptualization subscale	☐
	Construction subscale	☐
	Initiation subscale	☐
	Memory subscale	☐
	Total DRS score	✓
	Written math	☐
TMT		
	Part A total number of errors	✓
	Part A total seconds to complete	☐
	Part B total number of errors	☐
	Part B total seconds to complete	☐

Figure 6.4: Assessment score Selection Screen.



Assessment Query - Mozilla
File Edit View Go Bookmarks Tools Window Help
Home Bookmarks Red Hat Network Support Shop Products Training Oracle9i Struts Guide Struts API JSP Syntax PL/SQL
LOCI Morph-BIRN
Login
Query Assessments
Logout
Reset
Query
Copyright © 2003, UCSD-LOCI
Document: Done (0.143 secs)

Assessment Query Builder
Assessment Selection -> Score Selection -> Query

MMSE
MMSE Score > 25
MMSE Score Between and

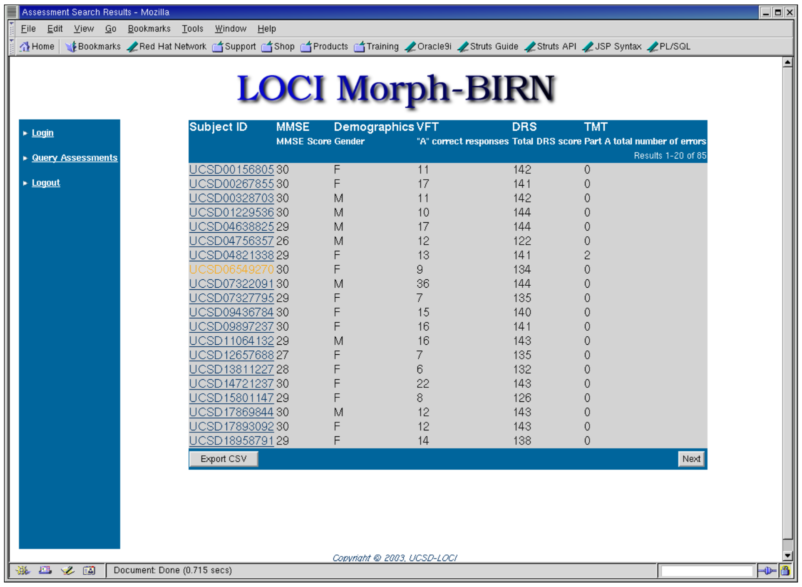
Demographics
AND [✓ ●] OR ○
Gender =

VFT
AND [✓ ●] OR ○
"A" correct responses > 5
"A" correct responses Between and

DRS
AND [✓ ●] OR ○
Total DRS score > 100
Total DRS score Between and

TMT
AND [✓ ●] OR ○
Part A total number of errors < 5
Part A total number of errors Between and

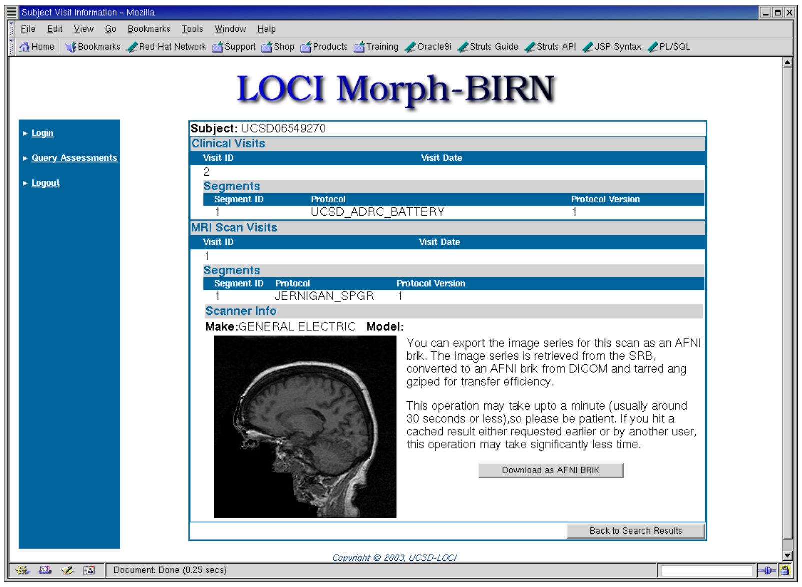
Figure 6.5: Assessment Query Criteria selection.



Assessment Search Results - Mozilla
File Edit View Go Bookmarks Tools Window Help
Home Bookmarks Red Hat Network Support Shop Products Training Oracle9i Struts Guide Struts API JSP Syntax PL/SQL
LOCI Morph-BIRN
Login
Query Assessments
Logout
Export CSV
Next
Copyright © 2003, UCSD-LOCI
Document: Done (0.715 secs)

Subject ID	MMSE	Demographics	VFT	DRS	TMT
	MMSE Score	Gender	"A" correct responses	Total DRS score	Part A total number of errors
					Results 1-20 of 85
UCSD00156805	30	F	11	142	0
UCSD00267855	30	F	17	141	0
UCSD00328703	30	M	11	142	0
UCSD01229536	30	M	10	144	0
UCSD04638825	29	M	17	144	0
UCSD04756357	26	M	12	122	0
UCSD04821338	29	F	13	141	2
UCSD06549270	30	F	9	134	0
UCSD07322091	30	M	36	144	0
UCSD07327795	29	F	7	135	0
UCSD09436784	30	F	15	140	0
UCSD09897237	30	F	16	141	0
UCSD11064132	29	M	16	143	0
UCSD12657688	27	F	7	135	0
UCSD13811227	28	F	6	132	0
UCSD14721237	30	F	22	143	0
UCSD15801147	29	F	8	126	0
UCSD17869844	30	M	12	143	0
UCSD17893092	30	F	12	143	0
UCSD18958791	29	F	14	138	0

Figure 6.6: Assessment score Selection Screen.


Subject Visit Information - Mozilla
File Edit View Go Bookmarks Tools Window Help
Home Bookmarks Red Hat Network Support Shop Products Training Oracle9i Struts Guide Struts API JSP Syntax PL/SQL
LOCI Morph-BIRN
Login
Query Assessments
Logout

Subject: UCSD06549270

Clinical Visits

Visit ID	Visit Date
2	

Segments

Segment ID	Protocol	Protocol Version
1	UCSD_ADRC_BATTERY	1

MRI Scan Visits

Visit ID	Visit Date
1	

Segments

Segment ID	Protocol	Protocol Version
1	JERNIGAN_SPGR	1

Scanner Info

Make:GENERAL ELECTRIC **Model:**

You can export the image series for this scan as an AFNI brik. The image series is retrieved from the SRB, converted to an AFNI brik from DICOM and tarred ang gziped for transfer efficiency.

This operation may take upto a minute (usually around 30 seconds or less),so please be patient. If you hit a cached result either requested earlier or by another user, this operation may take significantly less time.

Figure 6.7: Subject Visit Info Screen.

Appendix A

Getting and Building J2EE based web interface for local clinical imaging databases

A.1 Prerequisites

- Java SDK 1.3 or higher
- Apache Ant for build (http://ant.apache.org/)
- Jakarta Tomcat servlet/JSP container (http://jakarta.apache.org/tomcat/)
- DCMTK for DICOM header reading/editing
- AFNI and SRB Scommands if you want DICOM to AFNI conversion part

A.2 Getting the code

You can get the code from the BIRN CVS server, if you have BIRN CVS account by typing;

```
cvs checkout clinical
```

A.3 Preparation for build

Assuming the full path to the directory you have checked out UCSD Morph Human Imaging Database Web Interface and web application framework as $CLINICAL_HOME

- Copy $CLINICAL_HOME/build.properties.template to $CLINICAL_HOME/build.properties and edit using for Tomcat installation directory information
- Copy $CLINICAL_HOME/conf/users.xml.example to $CLINICAL_HOME/conf/users.xml and create database user - web user mappings accordingly.
- Copy $CLINICAL_HOME/conf/clinical.properties.example to $CLINICAL_HOME/conf/clinical.properties and modify according to your system configuration

- Copy $CLINICAL_HOME/conf/log4j.properties.example to $CLINICAL_HOME/conf/log4j.properties to setup rotating logging file(s) location.
- In $CLINICAL_HOME/web/WEB-INF/struts-config.xml file, edit the entry

  ```
  <set-property property="db_url"
             value="jdbc:oracle:thin:@fmri-gpop:1521:orcl1"/>
  ```

  section according to your database host, port and database name.

  The entry is in the plug-in section of the Struts configuration file

  ```
   <plug-in className="clinical.web.ServicesPlugin">

       <set-property property="driver_class"
               value="oracle.jdbc.driver.OracleDriver"/>
  <!-- for mbirn -->

        <set-property property="user_info_file"
            value="/WEB-INF/users.xml"/>
        <set-property property="db_url"
             value="jdbc:oracle:thin:@fmri-gpop:1521:orcl1"/>
  ```

A.4 Build and Deploy

In $CLINICAL_HOME, run

```
$ ant
```

A.5 Running web user interface

- Start Tomcat

  ```
  $ \$TOMCAT_HOME/bin/startup.sh
  ```

  item Stop Tomcat

  ```
  $ \$TOMCAT_HOME/bin/shutdown.sh
  ```

Appendix B

Struts configuration file

```
<?xml version="1.0" encoding="ISO-8859-1" ?>

<!DOCTYPE struts-config PUBLIC
          "-//Apache Software Foundation//DTD Struts Configuration 1.1//EN"
          "http://jakarta.apache.org/struts/dtds/struts-config_1_1.dtd">

<struts-config>

 <form-beans>
     <form-bean name="asSelectForm"
          type="clinical.web.forms.AsQueryBuilderForm" scope="session"/>

     <form-bean name="logonForm"
          type="clinical.web.forms.LogonForm" scope="request"/>

     <form-bean name="subVisitForm"
            type="clinical.web.forms.SubjectVisitForm" scope="request"/>

     <form-bean name="svmForm"
           type="clinical.web.forms.SubjectVisitManagementForm"
           scope="session"/>
 </form-beans>

 <global-exceptions>

 </global-exceptions>

   <global-forwards>

        <!-- Default forward to "Welcome" action -->
        <!-- Demonstrates using index.jsp to forward -->

        <forward name="welcome" path="/Welcome.do"/>
        <!-- IBO -->
        <forward name="logout" path="/logout.do"/>
        <forward name="querywizard" path="/querywizard.do"/>
```

```
    <forward name="asquery" path="/asquery.do"/>
    <forward name="logon" path="/logon.do"/>
     <forward name="login" path="/pages/logon_full.jsp"/>
    <forward name="as_sel" path="/pages/SelAs.jsp"/>

</global-forwards>

<action-mappings>

   <action path="/logon"
      type="clinical.web.actions.LogonAction"
      name="logonForm"
      scope="request"
      input="logon">
    <forward name="success" path="/pages/AfterLogin_full.jsp" />
    <forward name="failure" path="/pages/logon_full.jsp" />
   </action>

   <action  path="/asquery"
      type="clinical.web.actions.SelectAssessmentAction"
      name="asSelectForm"
      scope="session"
      input="/pages/SelAs_full.jsp">
    <forward name="success" path="/pages/SelScore_full.jsp" />
   </action>


  <action path="/querywizard"
     name="asSelectForm"
     scope="session"
     input="/pages/SelAs_full.jsp"
     parameter="/pages/SelAs_full.jsp"
     type="org.apache.struts.actions.ForwardAction"/>

  <action  path="/selectscores"
      type="clinical.web.actions.SelectScoresAction"
      name="asSelectForm"
      scope="session"
      input="/pages/SelScore_full.jsp">
    <forward name="success" path="/pages/SelectDerived_full.jsp" />
  </action>

  <action  path="/selectderived"
      type="clinical.web.actions.SelectSubcorticalsAction"
      name="asSelectForm"
      scope="session"
      input="/pages/SelectDerived_full.jsp">
    <forward name="success" path="/pages/CollectQuery_full.jsp" />
  </action>

  <action path="/collectquery"
  type="clinical.web.actions.AsQueryAction"
  name="asSelectForm"
  scope="session"
  input="/pages/CollectQuery_full.jsp">
 <forward name="success" path="/pages/SVResults_full.jsp" />
</action>

<action path="/svresults"
   type="clinical.web.actions.SVNavigateAction"
   name="asSelectForm"
   parameter="action"
   scope="session"
  input="/pages/SVResults_full.jsp">
  <forward name="success" path="/pages/SVResults_full.jsp" />
</action>

<action path="/subvisit"
   type="clinical.web.actions.SubjectVisitAction"
   name="subVisitForm"
   scope="session"
   input="/pages/SubjectVisit_full.jsp"
   >
   <forward name="success" path="/pages/SubjectVisit_full.jsp" />
</action>


<action path="/downloadafni"
   type="clinical.web.actions.MRIDownloadAction"
   name="subVisitForm"
   scope="session"
   input="/pages/conv_status_full.jsp" >
  <forward name="success" path="/pages/conv_status_full.jsp" />
</action>

<action path="/viewqueryres"
   name="asSelectForm"
   scope="session"
   parameter="/pages/SVResults_full.jsp"
   type="org.apache.struts.actions.ForwardAction"/>

 <action path="/convertafni"
     name="subVisitForm"
     scope="session"
    type="clinical.web.actions.ConversionProgressAction">
  <forward name="success" path="/pages/conv_status_full.jsp" />
 </action>

 <action path="/findsubjects"
    name="svmForm"
    scope="session"
    type="clinical.web.actions.SubjectQueryAction">
```

```
      <forward name="success" path="/pages/SubjectList_full.jsp" />
     </action>

     <action path="/navsubjects"
       type="clinical.web.actions.SubjectSRNavigateAction"
       name="svmForm"
       parameter="action"
       scope="session"
       input="/pages/SubjectList_full.jsp">
      <forward name="success" path="/pages/SubjectList_full.jsp" />
      <forward name="edit_subject" path="/pages/SubjectDetail_full.jsp" />
     </action>

     <action path="/editsubject"
       type="clinical.web.actions.SubjectManagementAction"
       name="svmForm"
       parameter="action"
       scope="session"
       input="/pages/SubjectDetail_full.jsp">
      <forward name="success" path="/pages/SubjectDetail_full.jsp" />
      <forward name="edit_visit" path="/pages/SubjectDetail_full.jsp" />
     </action>

     <action path="/logout"
       type="clinical.web.actions.LogoutAction">
       <forward name="success" path="/index.jsp" />
     </action>

  </action-mappings>

  <controller
     processorClass="org.apache.struts.tiles.TilesRequestProcessor"/>

  <message-resources parameter="resources.application"/>

 <plug-in className="org.apache.struts.tiles.TilesPlugin" >
    <set-property property="definitions-config"
               value="/WEB-INF/tiles-defs.xml" />
    <set-property property="moduleAware" value="true" />
    <set-property property="definitions-parser-validate" value="true" />
 </plug-in>

 <plug-in className="org.apache.struts.validator.ValidatorPlugIn">
    <set-property
      property="pathnames"
      value="/WEB-INF/validator-rules.xml,/WEB-INF/validation.xml"/>
 </plug-in>

 <plug-in className="clinical.web.ServicesPlugin">
     <set-property property="driver_class"
           value="oracle.jdbc.driver.OracleDriver"/>
       <set-property property="user_info_file"
              value="/WEB-INF/users.xml"/>
       <set-property property="db_url"
              value="jdbc:oracle:thin:@fmri-gpop:1521:orcl1"/>
       <set-property property="mri_jpegs_dir"
               value="/mri_images" />
       <set-property property="cache_root"
               value="/WEB-INF/cache"/>
    </plug-in>

</struts-config>
```

Appendix C

Simple O/R Mapping Tool Help

C.1 Main Screen

After running `codegen.sh`, you should see the main screen. Here you will see a tree pane for the schema objects (tables and views) for the Oracle database to which you will connect to extract schema information and generate Java code for a simple one-to-one object relational mapping. On the right pane titled Code Generation you have fields which are prepopulated and default code type to generate as dao (*Data Access Object*).

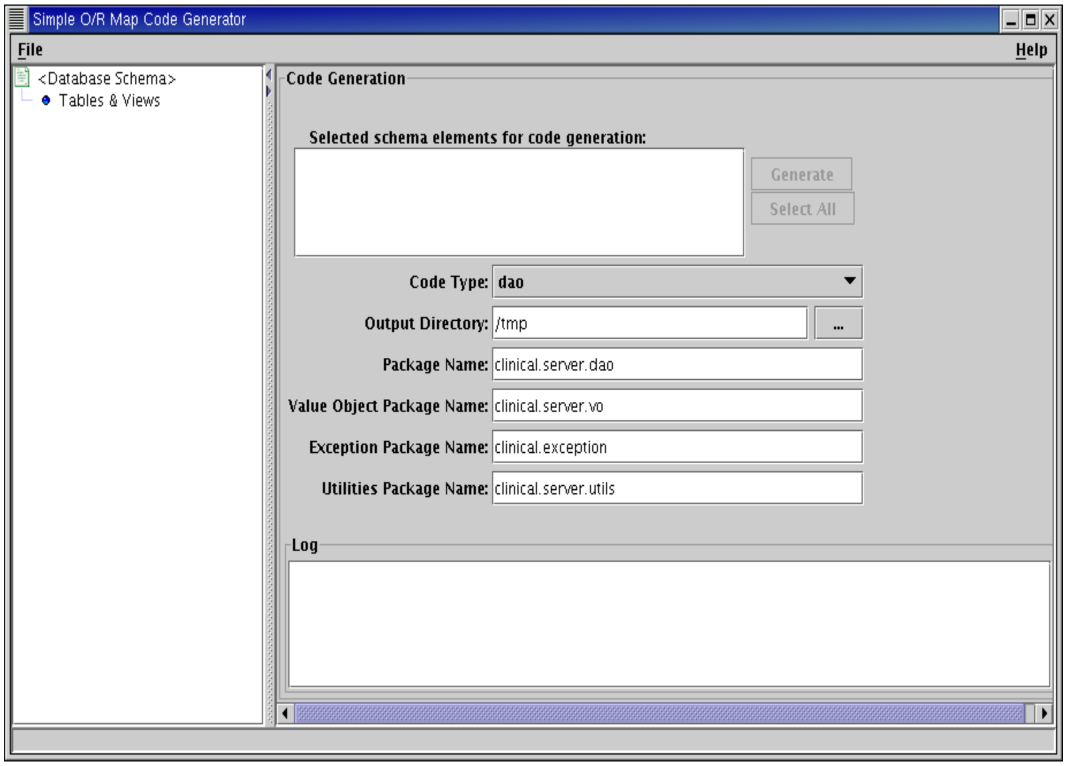
Currently three types of Java classes can be generated

- **Data Access Object** (dao) - These objects form the data access layer and they are responsible for CRUD (create, read update and delete) operations on their corresponding table. Each DAO has a find method to which you can pass a *value object* (vo) with some properties set using setter methods and it will internally create the corresponding SQL and return the results as a list of populated value objects. Each DAO also has an insert(), delete() and update() method.
- **Value Object** (vo) - A value object is a Java bean used by the corresponding DAO and different software layers to transfer coarse grained data between layers. Code generator generates a value bean for each selected schema object where the properties match the columns of the schema object.
- **Primary Key Object** (pk) - Currently not used. A primary key object maps to the primary key of its corresponding schema object.

C.2 Schema Extraction

The first step in code generation is schema extraction from an Oracle database. First, select *Load Schema Objects* from *File* Menu. The following dialog box will appear. Initially the fields in the dialog box will be empty. However, the code generator saves the last setting after each run.

Here you need to provide database host, port, name and user/schema information for the schema where your database tables /views reside. Then, press OK. After successful schema extraction, the *Generate* and *Select All* buttons in the Code Generation panel will be enabled. And you will see the tree and schema object list box populated. Now you can select the schema objects from the list, adjust the parameters like code type and output directory and press *Generate* button. Here, two data access objects are generated for `NC_USERCLASS` and `NC_EXPSEGMENT` schema objects and saved to `/tmp/clinical/server/dao` directory. The Package Name is the name of the package



Selected schema elements for code generation:

Code Type: dao

Output Directory: /tmp

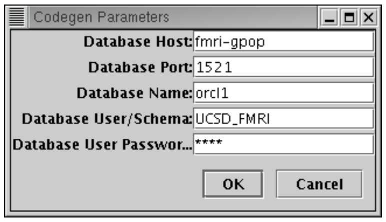
Package Name: clinical.server.dao

Value Object Package Name: clinical.server.vo

Exception Package Name: clinical.exception

Utilities Package Name: clinical.server.utils

Figure C.1: Database connection parameters dialog.



Database Host: fmri-gpop

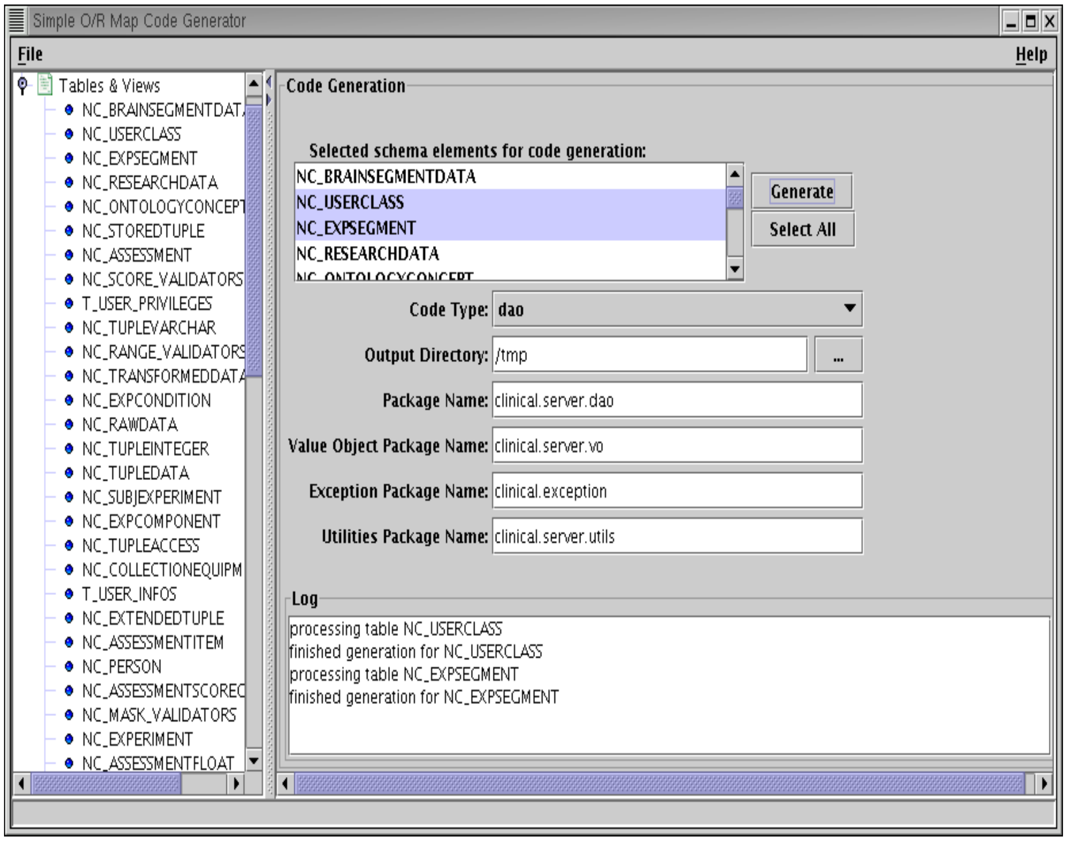
Database Port: 1521

Database Name: orcl1

Database User/Schema: UCSD_FMRI

Database User Passwor...: ****

Figure C.2: Database connection parameters dialog.



Selected schema elements for code generation:
NC_BRAINSEGMENTDATA
NC_USERCLASS
NC_EXPSEGMENT
NC_RESEARCHDATA
NC_ONTOLOGYCONCEPT

Code Type: dao
Output Directory: /tmp
Package Name: clinical.server.dao
Value Object Package Name: clinical.server.vo
Exception Package Name: clinical.exception
Utilities Package Name: clinical.server.utils

```
processing table NC_USERCLASS
finished generation for NC_USERCLASS
processing table NC_EXPSEGMENT
finished generation for NC_EXPSEGMENT
```

Figure C.3: Code Generator user interface look after code generation.

for the data access objects. Since data access objects depend both on code generated (like value objects) and programmer written Java code residing in clinical.exception and clinical.server.utils packages, if you want to use a different package structure, you need to move the dependencies in clinical.exception and clinical.server.utils packages to your package structure and set the Package Name, Value Object Package Name, Exception Package Name and Utilities Package Name fields accordingly.

C.3 Custom Code in Generated Java Code and Code Regeneration

During development, there may be cases where you need to customize some of the generated code. If you don't want your custom code to disappear after code regeneration, you should put your custom code within delimeters like shown below.

```
 /*+++    */
  // Enter your code here
    String myVar;
    public String getMyVar() { return this.myVar; }
/*+++    */
```

Appendix D

Database Security

Sharing of biomedical human subject data is highly sensitive and the security requirements are much higher than those for most other database applications. The HIPPA confidentiality requirements must be met. Another complicating factor is having shared and private data within the same database. Separating private and shared data into two different databases solves this problem, however it creates increased maintenance and development costs. Hence, for the UCSD human imaging database, the first approach is used.

Even though security can be handled in application levels, pushing it to the database layer centralizes the security and decouples it from the client applications, which allows uniform enforcement of security rules.

Having both private and shared data together in the same database means that both private and public data must reside in the same database tables, which requires a row level security on the database side.

D.1 Virtual Private Database

Oracle provides row level security via a mechanism called Virtual Private Databases (VPD). With VPD you establish policies in the form of predicates (where clauses) that are attached to every query that the user presents to the database. Using VPD, a security structure must only be built once in the database server. Since the security policies are attached to the data instead of the application(s) using the data, security rules are enforced at all times and from any access approach. The SQL query to access the data is the same regardless of the client; the filtering predicates are applied during query processing by the Oracle database server transparently. VPD works under the assumption that a client will connect via a named user connection. Traditional connection pooling is not supported, since connections will not have any identity in that case.

D.2 Oracle Label Security

Oracle label security [LM02] is a collection of procedures and constraints built on top of VPD to enforce row level security. Each table which needs row level security is augmented by a (hidden) column which contains a *security label* for each row. The labels designate which user have access to what types of data. The collection of security rules and access requirements is called a *security policy.* The security labels are always associated with a security policy. A fine granular security is provided via levels, optional compartments and groups. A *level* is a ranking that denotes the sensitivity of the information it labels. A *compartment* refines the access to the information within a label. A *group* refines a compartment and allows representation of a hierarchy of users. A security label has the following format

```
level : compartment 1,..., compartment n : group 1, ... , group n
```

Each named database user is assigned *user labels* to set his/her maximum read and write labels, minimum write labels, default and row levels.

To illustrate label security, assume there are two levels (public, private), then two compartments (lab1, lab2) are defined. Also assume that there is a hierarchy of groups within each compartment, namely group1 and its two subgroups group2 and group3. Let consider a possible label *public:lab1:group2*. Any row which has this label in a protected table can be seen (given sufficient user label privileges) by a user working for lab1 and being a member of group2 or group1. Since group1 is higher in hierarchy, any user in group1 can access data from lower hierarchies.

In the UCSD human imaging database, currently, two levels (PUBLIC and PRIVATE) and two compartments (LOCI, BIRN) are defined. Currently, 13 tables for assessment and subject visit data are under label security control. In some tables where there are some sensitive columns mixed with less sensitive columns, even row level security is not enough fine granularity. In those cases a view is created on top of the protected tables and exposed to the less privileged named database users instead of the full table.

Bibliography

[ACM01] Deepak Alur, John Crupi, and Dan Marks. *Core J2EE Patterns: Best Practices and Design Strategies*. Prentice Hall, 2001.

[BRJ97] Grady Booch, James Rumbaugh, and Ivar Jacobson. *The Unified Modelling Language User Guide*. Addison Wesley, 1997.

[Cav02] Chuck Cavaness. *Programming Jakarta Struts*. O'Reilly, 2002.

[GHJV94] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns - Elements od Resuable Object-Oriented Software*. Addison Wesley, 1994.

[LM02] Jeff Levinger and Rita Moran. *Oracle Label Security Administrator's Guide Release 2*. Oracle, 2002.