



Programmers' Technical Reference Manual

United States Agency
for International Development

Table of Contents

1.	Introduction	4
2.	System Scope	4
2.1.	Hardware & Software Requirements.....	5
2.1.1.	Hardware	5
2.1.2.	Software.....	6
2.1.3.	Network	6
2.2.	Programming Skills Required	6
2.3.	Development Environment.....	6
2.4.	System Design Tools.....	7
3.	FinA Functional Design	8
3.1.	Data Flow	8
3.2.	Data Model.....	10
3.3.	Basic Application Design	11
3.4.	Basic Class Design	11
4.	Technical System Design	11
4.1.	Overview	11
4.2.	Application's Tiers	12
4.3.	UI Manager.....	13
4.4.	Menu Customization & Actions	13
4.5.	Message Bundle.....	14
4.6.	Configuration	15
4.7.	Business Logic	15
4.7.1.	Overview	15
4.8.	FinA Enterprise Java Beans	15
4.9.	Transactions' Types	18
4.10.	Security.....	19
4.10.1.	Entity Beans.....	19
4.10.2.	Client.....	19
4.10.3.	Server.....	19
4.11.	Return Processing.....	20
4.12.	Database	23
4.12.1.	Tables Structure Overview	23
4.12.2.	Stored Procedures.....	25
4.13.	Procedures involved in Reporting.....	26
4.14.	Configuration	26
4.15.	Patterns.....	26
4.15.1.	Overview	26
4.15.2.	EJBTree Pattern	27
4.15.3.	EJBTable Pattern	28
4.16.	Reporting	31
4.16.1.	OpenOffice.org CALC	31
4.17.	Application Server	32
4.18.	Configuration Properties	33
4.19.	Build (version) Management.....	34
4.20.	Development Environment.....	34
4.21.	Directories.....	34
4.22.	The Build Procedure	35
5.	Installation Package Build.....	35
6.	Database Modification Procedure.....	39
7.	Coding Standards	39
8.	Naming Conventions	39
9.	Code Security	40
10.	Reference.....	41

List of Figures

FIGURE 2.4.1, GENERATE HTML.....	7
FIGURE 3.1, FINA-DEPLOYMENT DESIGN.....	8
FIGURE 3.1.1, FINA INTERNATIONAL DATA FLOW DIAGRAM.....	9
FIGURE 3.2.1, FINA INTERNATIONAL SYSTEM DATA MODEL.....	10
FIGURE 3.2.2, THE BANK REGULATORS OFF-SITE MONITORING SYSTEM DATA MODEL.....	10
FIGURE, 3.3.1, FINA INTERNATIONAL – ARCHITECTURE.....	11
FIGURE 4.4.1, ACTION REGISTRATION.....	14
FIGURE 4.7.1, FINA BUSINESS LOGIC.....	15
FIGURE 4.10.3, LOGIC SEQUENCE.....	20
FIGURE 4.11.1, SEQUENCE DIAGRAM.....	23
FIGURE 4.15.2.1, EJB TREE PATTERN.....	27
FIGURE 4.15.2.2, AMEND ACTION LOGIC.....	28
FIGURE 4.15.3.1, EJB TABLE PATTERN.....	29
FIGURE 4.15.3.2, POP-UP MENU PATTERN.....	30

1. Introduction

The purpose of the United States Agency for International Development (USAID) FinA Programmers' Technical Reference Manual is to guide the programmer with step-by-step instructions to implement new ideas and make changes into the software package.

All files necessary to read and understand the guide are provided in the **Programmers' Guide/DOC** subfolder:

/DOC/ClassDiagram_JavaDoc	Folder
/DOC/DataBaseDiagram	Folder
/DOC/Driver_JavaDoc	Folder
/DOC/SourceCode	Folder
/DOC/Functional Specifications	Word Document
/DOC/helloworld_action.zip	Java Samples
/DOC/ProgGuide_Apendif1_CamelManual.pdf	Appendix
/DOC/ProgGuide_Apendif2_iso15408-3.pdf	Appendix
/DOC/Xls2Xml.zip	Demo Converter
/DOC/XML_Return.dtd	Document Type Definition

2. System Scope

FinA International is an information technology (IT) tool that supports the off-site banking supervision activities of the Supervisor of the Banking System (Supervisor). FinA is a tool that enhances the Regulator's analytical capabilities and increases the overall effectiveness of supervisory strategies. One of the key components of effective bank supervision is the review and analysis of returns (call reports) that banks are required to submit periodically to the Supervisor. The content, format, and periodicity of these returns are defined in the banking laws and regulations of the country.

Financial data from banks is sent to the Supervisor in the form specified in the legal/regulatory framework. The information is then analysed by the Supervisor to ensure compliance with regulation, to assess the financial stability of the institution, and to analyse economic trends. Based on the data returned by a credit institution, its financial condition is rated using financial ratios. These ratios are used to compare a bank against its peers and predefined standards for that country.

The main report generated by the system is the Universal Bank Performance Report (UBPR), which contains a report for each bank of its current financial position, the CAMELS analysis, time series analysis of its key financial indicators, and its peer group and ranking within that group. This report also provides the same analysis for the banking system as a whole.

The usefulness of off-site supervision is dependent upon the timeliness, accuracy, and usefulness of the financial data submitted and the timeliness and quality of the analysis

performed. Through automation that delivers improved accuracy and speed of data, the FinA system can significantly enhance the quality of off-site supervision.

Bank returns are submitted to the Supervisor in a variety of formats (Excel, text files, etc.). These files must then be converted to XML files to be used in the FinA system. Once the files are converted to FinA XML, they can be processed by the system and placed in the database. DTD of this XML can be found in DOC/return.dtd file. For an example of such converter from XLS to XML file, see in DOC/XLS2XML.ZIP. The converter maps data from the existing format (XLS, DBF, TXT, etc) to the FinA structure described as Metadata.

2.1. Hardware & Software Requirements

2.1.1. Hardware

FinA is written in Java and has three-tier system architecture and accordingly has minimal requirements on hardware for the client machines.

□ Hardware For FinA Client Work Station:

Minimum Configuration: Pentium II class - RAM 64 MB, 40 MB free disk space, 10/100 Mbps Ethernet NIC, SVGA 800x600

Recommended Configuration: Pentium II class - RAM 256 MB, 40 MB free disk space, 10/100 Mbps Ethernet NIC, SVGA 800x600

□ Hardware For FinA App Server:

Minimum Configuration: Pentium II class – random access memory (RAM) 128 MB/50 MB for application server files, plus necessary space for database (approximately 200 MB for one year of data for smaller Central Banks), 10/100 Mbps Ethernet NIC, SVGA 800x600

Recommended Configuration: Pentium III class - RAM 512MB, 20GB HDD, 10/100 Mbps Ethernet NIC, SVGA 800x600

□ Hardware For Database Server:

For the minimum as well as recommended hardware configuration information, please go to <http://www.microsoft.com/sql/default.asp>

Note: *The amount of RAM required for running the Database server depends on the size of the database and the administration tools being used. A typical installation of a Database server requires a minimum of 300-400 MB of disk space.*

2.1.2. Software

The list of software requirements follows:

- ❑ Operating system for the client: MS Windows 98/NT/2000/XP
- ❑ Software for the client and application server: FinAClient and FinAServer
(included in the installation package)
- ❑ Database Server: MS SQL 7.0/2000

2.1.3. Network

FinA does not need a high-speed network because all logistical calculations take place on the server. Actually, 1 Mbps IP-based LAN is sufficient to run and maintain the system. An IP protocol must be installed and properly configured in working environment. (for more information, see Chapters 4.6, Configuration; and 4.10, Security). Basic recommendations are for 10 Mbps network from the Workstations to the Application server and 100 Mbps from the Application server to the database server (if they are running on different machines).

Note that the current version of FinA, version 0018d7, cannot be installed and configured as standalone. For further information please refer to the FinA Installation and Configuration Manual.

2.2. Programming Skills Required

Programmers that plan to work on the FinA system should have at least the following skills:

OS - Windows 2000/XP
Java - Java2, Forte 4, JSP (preferable)
EJB - JBoss 3.0
Spreadsheets -OpenOffice.org 1.0 Calc
MS SQL Server 2000/7.0 (Maintains, Stored procedures, etc)

2.3. Development Environment

- ❑ The application is written in Java using Forte (community edition) on a Windows 2000/XP platform against a Microsoft SQL Server 7/2000 database.
- ❑ The Java JDBC driver is used for the interface.
- ❑ Swing components are used throughout the UI
- ❑ JBoss 3.0 is the application server
- ❑ Data Base design is done in Visio 2000
- ❑ System design was done in the Together Control Center
- ❑ FinA v0018e was tested only for Windows/MSSQL environment

2.4. System Design Tools

System design and documentation was done using "Together Soft". For modifications to the system, "Rational Rose", "Visio" or any other tool can be used. Below are the procedures for "Together Soft".

Note: If a programmer does not plan to make any major changes to the system, s/he will not need to use any system design tools.

To see FinA System design, copy file FinaJdoc.zip from the installation CD and unzip it in C:\TogetherSoft\Together6.0.1\myprojects\ directory. Then run "Together" and open C:\TogetherSoft\Together6.0.1\myprojects\FinA\ FinA.tpr.

If a programmer wants to add or change something in the programmers' documentation, from the "Together Soft" menu s/he should click on **Project/Documentation/Generate HTML**. Then select **All** under **Scope** (see *Figure 2.4.1, Generate HTML* below) and click **OK**.

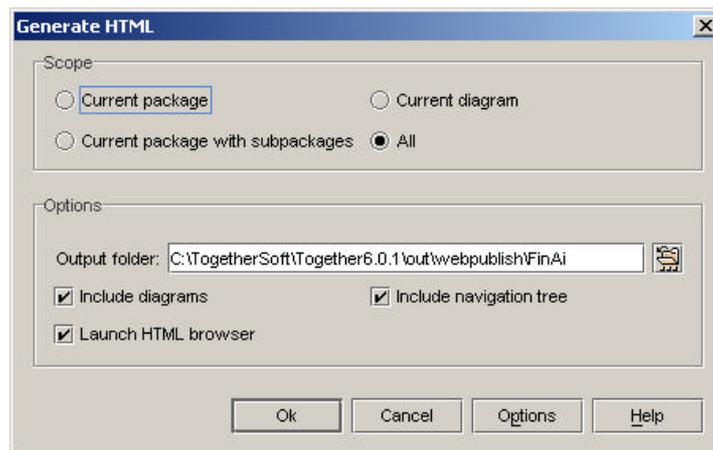


FIGURE 2.4.1, GENERATE HTML

The system will then begin to create the documentation. It takes approximately three to five minutes to complete the generation process.

3. FinA Functional Design

Refer to **/DOC/Functional Specifications** for information on the functional specifications for FinA. A design chart is presented on *Figure 3.1, FinA-Deployment Design*.

FinA - Deployment Design

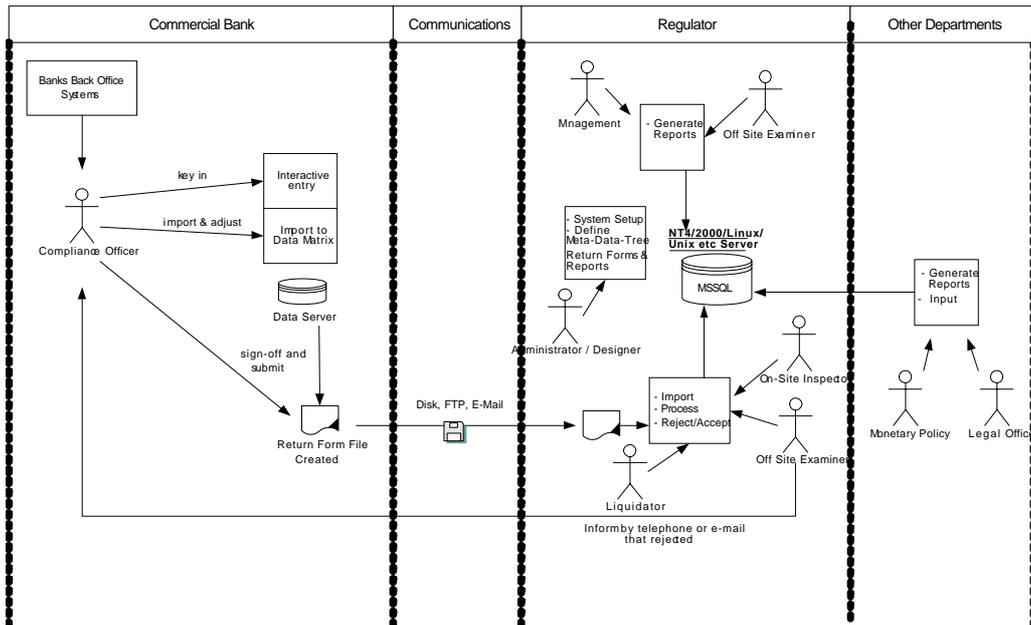


FIGURE 3.1, FINA-DEPLOYMENT DESIGN

3.1. Data Flow

The FinA system data flow is illustrated in *Figure 3.1.1, FinA International Data Flow Diagram*.

FinA International Data Flow Diagram

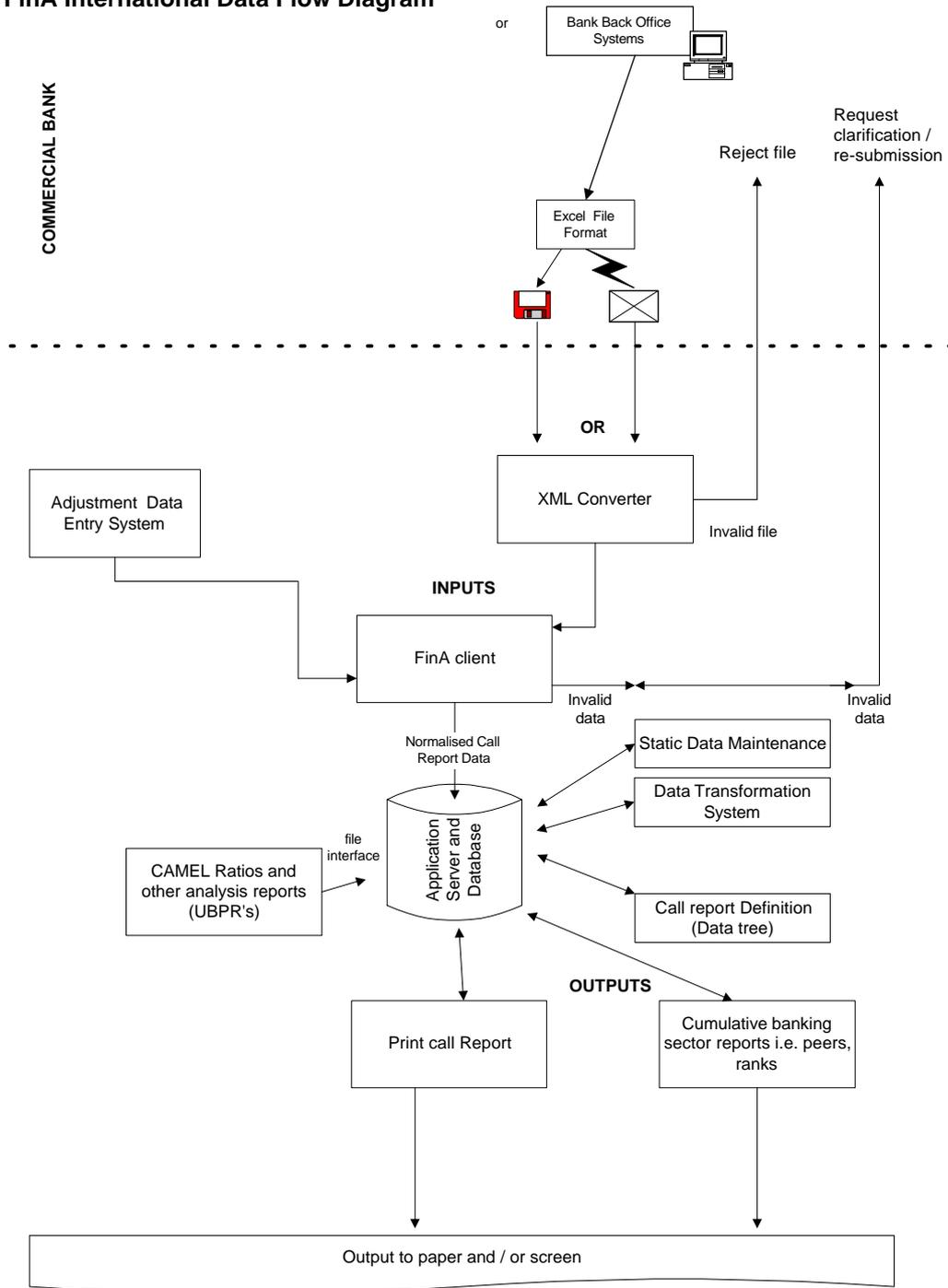


FIGURE 3.1.1, FINA INTERNATIONAL DATA FLOW DIAGRAM

3.2. Data Model

The data model of FinA is presented on *Figure 3.2.1, FinA International System Data Model* and the off-site monitoring system data model is presented on *Figure 3.2.2, The Bank Regulators Off-Site Monitoring System Data Model*.

FinA International System Data Model

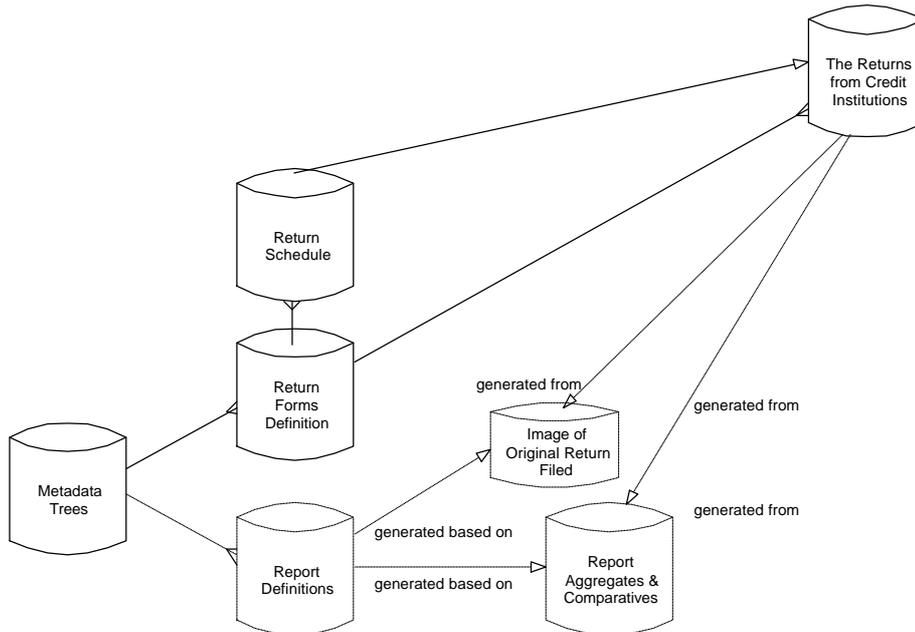


FIGURE 3.2.1, FINA INTERNATIONAL SYSTEM DATA MODEL

The Bank Regulators Offsite Monitoring System Data Model

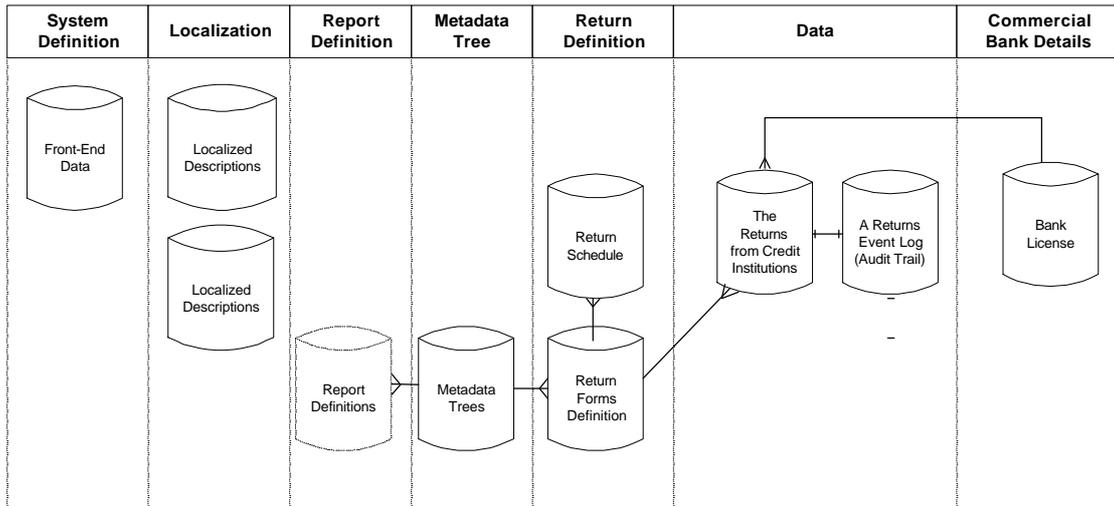
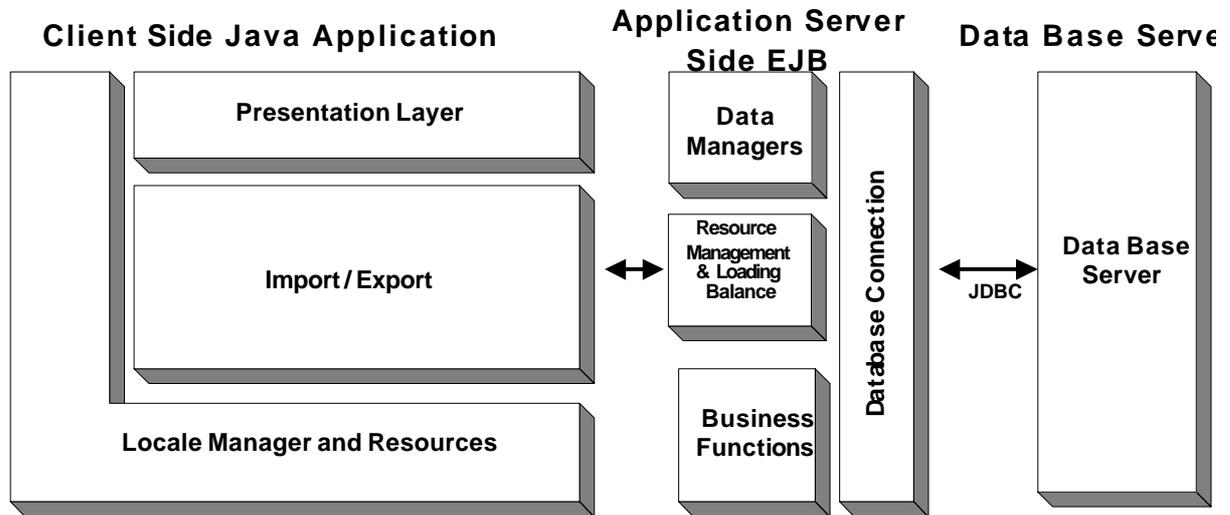


FIGURE 3.2.2, THE BANK REGULATORS OFF-SITE MONITORING SYSTEM DATA MODEL

3.3. Basic Application Design

The architecture of the system is presented in *Figure, 3.3.1, FinA International – Architecture.*

FinA International- Architecture



FIGURE, 3.3.1, FINA INTERNATIONAL – ARCHITECTURE

Client Side contains:

- ❑ Presentation layer - user interface coded in Java SWING.
- ❑ Locale management tools - font, formats, character sets can be defined in client side modules.
- ❑ Import/export tools - Users can import and export data into the system according to the user access rights and permissions.

Application server:

- ❑ Maintains all functionality and calculations, prepares data for presentation.
- ❑ Creates data connection with database through the JDBC driver
- ❑ Manages and sorts data, and generates relevant SQL queries.
- ❑ Manages resources and does load balancing.

All data is stored in the MS SQL7/2000 database server.

3.4. Basic Class Design

For basic class design information refer to file: */DOC/ClassDiagram_JavaDoc*

4. Technical System Design

4.1. Overview

All parts of FinA follow similar design patterns. It is necessary to be familiar with these patterns and rules in order to understand the architecture of FinA and to be able to modify the code or add modules to FinA.

This chapter describes all of the basic design patterns, which are necessary to become a FinA developer.

Technologies used for FinA development:

- ❑ Java 2 Platform version 1.3.1
- ❑ Swing
- ❑ Forte Development Environment 3.0 or later
- ❑ Enterprise Java Beans (JBoss 3.0)
- ❑ XML (Apache Xerces)
- ❑ JDBC
- ❑ Open Office.org

JBoss is an open source application. Version 3.0 is used for FinA. It is based fully on J2EE and supports all of the J2EE stacks. JBoss is located in FINA_INSTALL_DIR/server/jboss and has the following directory structure:

/bin	contains executables to run and stop the server
/catalina	Tomcat WEB container
/client	class libraries required for client applications
/lib	JBoss core class libraries
/server	contains sub-folders for different configurations, FinA uses “default” configuration
/server/default	
/server/default/conf	configuration files for default JBoss configuration
/server/default/deploy	deployment directory
/server/default/lib	extended libraries for default configuration
/server/default/log	contains server logs
/server/default/tmp	temporary files

In standard configuration JBoss 3.0. is used as Application Server, but it is possible to port FinA on any J2EE compatible server. In FinA the JBoss core is used as the application server. It is an Enterprise JavaBeans (EJB) server. JBoss core has minimal memory and disk space requirements. JBoss runs very effectively on a machine with 64 megabytes of RAM, and requires only a few megabytes of disk space. JBoss also supports ‘hot’ deployment. This means that deploying a Bean is as simple as copying its JAR file into the deployment directory. If this is done while the Bean is already loaded, JBoss automatically unloads it, then loads the new version.

OpenOffice.org Calc is another open source application used in FinA. See Chapter 4.16.1, OpenOffice.org Calc for more information on how OpenOffice.org interacts with FinA.

4.2. Application’s Tiers

Client application of FinA is based on Swing user interface. In the standard configuration interaction with the server takes place by the RMI protocol. The application may also be configured for working with IIOP or with other protocols, depending on the Application Server.

fina2.Main and fina2.UIManager classes are the core of the FinA Client Application. With the assistance of these classes, different modules of the program can have access to the resources, register themselves, and interact with each other.

4.3. UI Manager

UIManager provides ability to manage the following parts of the Client side user interface:

- ❑ actions
- ❑ message bundles
- ❑ configuration

UIManager is available as a “public static” field in fina2.Main class and is accessible from any part of client application. For example, fina2.Main.ui.getString(key) – returns String with specified key from current locale’s message bundle.

4.4. Menu Customization & Actions

Actions are instances of classes implementing javax.swing.Action interface. Top actions, which must be available globally (generally top menu bar actions) should be registered in UIManager at program start time (inside initTopActions() method). Actions, which act only in the context of separate dialog boxes or windows are local actions and should not be added to UIManager.

This feature simplifies menu bar customization process for the user. The user selects top actions available in the system from a list and attaches them to menu bar items.

To illustrate what is necessary to create new FinA action, please review the “Hello World” example below:

- ❑ Create new class HelloWorldAction extended from javax.swing.AbstractAction

```
public class HelloWorldAction extends AbstractAction {
    public HelloWorldAction() {
    }

    public void actionPerformed(java.awt.event.ActionEvent event) {
    }
}
```

- ❑ Write initialization, resources allocation, etc., code in constructor of the class (this code will be executed at program startup)

```
public HelloWorldAction() {
    super();

    main.setLoadingMessage(ui.getString("fina2.helloWorld"));
    /* above line allows to show specified message in logo window during program
    startup when system loads this action */

    putValue(AbstractAction.NAME, ui.getString("fina2.helloWorld"));
}
```

- ❑ dlgs = new HelloWorldDialog(main.getMainFrame(), true);

```
❑ ui.loadIcon("fina2.helloworld.icon", "icon.gif");
    putValue(
        AbstractAction.SMALL_ICON,
        ui.getIcon("fina2.helloworld.icon")
    );
```

- ```

 ui.addAction("fina2.helloworld.HelloWorld", this);
}

```
- Write action handler code in actionPerformed method
  - Create an instance of the action, for example “new fina2.helloworld.HelloWorldAction();” in fina2.ui.UIManager.initTopActions method

Full source code and compiled classes of this example can be found in folder DOC\helloworld\_action.zip.

An Action must register itself in the system by calling UIManager.addAction(String key, javax.swing.Action action) method. The first parameter is a unique key of the action. The action can be found and accessed from any part of the system with help of UIManager.getAction(key) method and by using this key. It is recommended to use a full class name of the action as key. The second parameter is an instance of the action.

The logical sequence for registering an Action is presented on *Figure 4.4.1, Action Registration*.

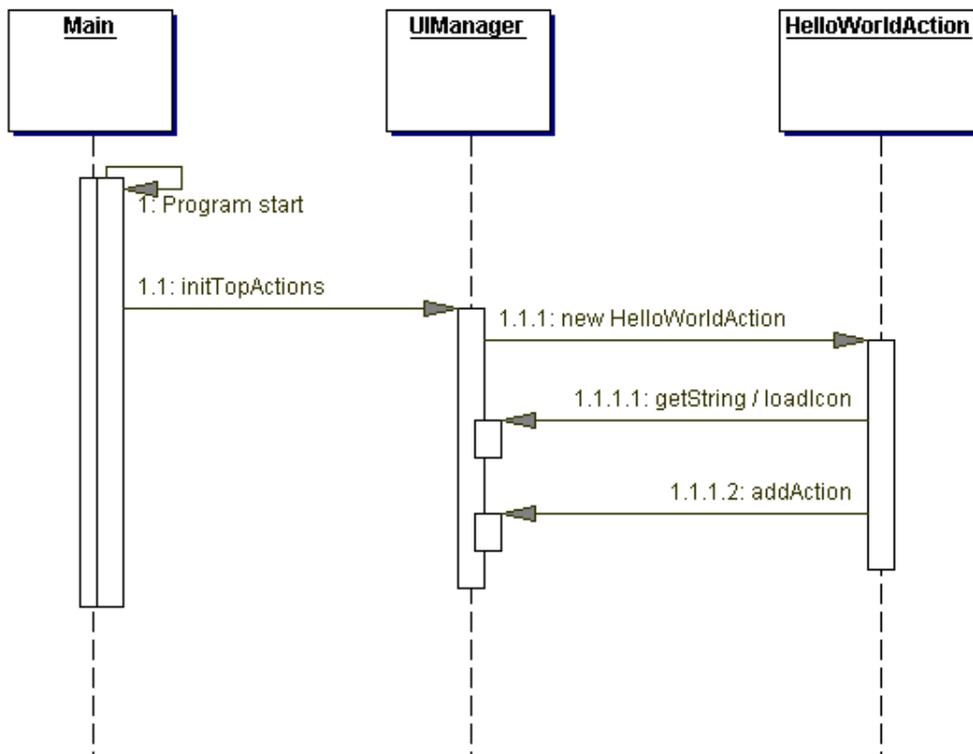


FIGURE 4.4.1, ACTION REGISTRATION

**Note:** *The best way to create new action is to get an existing one and edit or replace constructor and handler with your code because most actions are similar.*

#### 4.5. Message Bundle

According to current language settings UIManager loads the appropriate message bundle from a file located on the client computer and makes it available for the client application.

Message Bundle file is located in FINA\_INSTALL\_DIR/conf directory.

Also UIManager.getFont() method returns default font with default size according to current language settings.

#### 4.6. Configuration

UIManager's putConfigValue(key) and getConfigValue(key) methods provide the ability for different UI components (such as windows, dialog boxes etc.) to store their own configuration parameters. For example, a window must store position and dimension values. The Return Manager window must store settings of filter bar.

Any Java object implementing java.io.Serializable interface may be stored as a configuration value. They will be serialized and saved into the configuration file at program finish and restored at program start time.

#### 4.7. Business Logic

##### 4.7.1. Overview

Figure 4.7.1, FinA Business Logic shows the Business Logic Diagram of FinA, which outlines the functions performed on the Client, and those functions performed on the Application server and the Database server.

#### FinA International- Business Logic Diagram

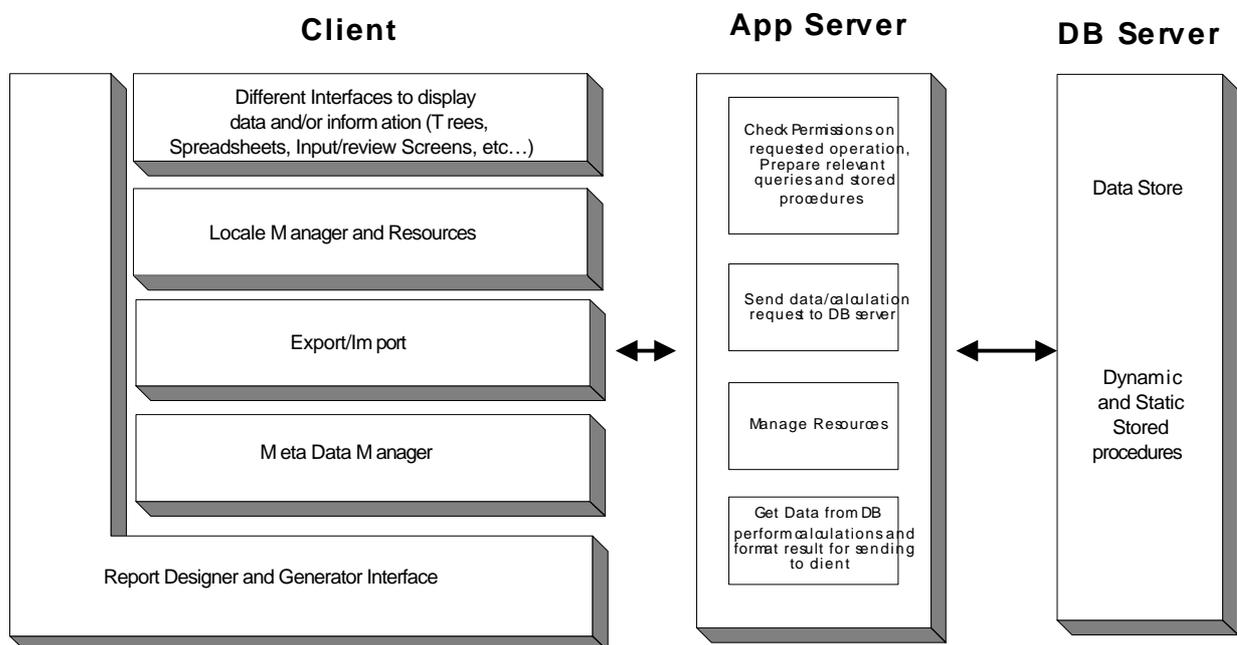


FIGURE 4.7.1, FINA BUSINESS LOGIC

#### 4.8. FinA Enterprise Java Beans (EJBs)

The following types of EJBs are used in FinA:

- ❑ Entity Beans with Bean Managed Persistence (BMP)
- ❑ Stateless Session Beans

All Beans that have class names that end with “Bean” are the Entity Beans. Beans which have names that end with “SessionBean” accordingly are Session Beans.

Since Bean Managed Persistence is used in FinA, all SQL queries are embedded in the Beans’ code and are not generated automatically by EJB container. This approach was chosen because the main part of queries contains complex relations between the tables and therefore could not be generated automatically. For example, for getting information about separate Metadata Tree Nodes, it is necessary to select information about the node from the IN\_MDT\_NODES table and its name from SYS\_STRINGS table, which contains text strings based on the selected language.

The following table contains a list of EJBs used in FinA. See also class diagrams [/DOC/ClassDiagram\\_JavaDoc/index.html](/DOC/ClassDiagram_JavaDoc/index.html)

| Name                           | Type              | Description                                                                                                                                                 |
|--------------------------------|-------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------|
| fina2.i18n.LanguageBean        | Entity            | Allows creating, finding, and accessing locales’ properties.                                                                                                |
| fina2.i18n.LanguageSessionBean | Stateless Session | Contains methods to manage tables of system locales see 5.3.3. EJBTable pattern.                                                                            |
| fina2.security.UserBean        | Entity            | Allows creating, finding, and accessing users’ properties. Check permissions & banks/reports access rights.                                                 |
| fina2.security.RoleBean        | Entity            | Allows creating, finding, and accessing user roles’ properties. Check roles’ permissions.                                                                   |
| fina2.security.UserSessionBean | Stateless Session | Contains methods to manage tree of users/roles see 5.3.2. EJBTree pattern.                                                                                  |
| fina2.ui.menu.MenuBean         | Entity            | Allows creating, finding, and accessing menu items’ properties.                                                                                             |
| fina2.ui.menu.MenuSessionBean  | Stateless Session | Contains methods to manage menu tree see 5.3.2. EJBTree pattern.                                                                                            |
| fina2.metadata.MDTNodeBean     | Entity            | Allows creating, finding, and accessing metadata nodes’ properties.                                                                                         |
| fina2.metadata.MDTSessionBean  | Stateless Session | Contains methods to manage Metadata tree see 5.3.2. EJBTree pattern. Find dependencies between nodes. Find Return Definitions where specified node is used. |
| fina2.bank.BankTypeBean        | Entity            | Allows creating, finding, and accessing bank types’ properties.                                                                                             |

|                                    |                   |                                                                                                                                                                                         |
|------------------------------------|-------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| fina2.bank.BankSessionBean         | Stateless Session | Contains methods to manage tables & hierarchy of banks, bank types, bank groups, bank licenses, branches see 5.3.2. EJBTree pattern & 5.3.3. EJBTable pattern.                          |
| fina2.bank.BankBean                | Entity            | Allows creating, finding, and accessing banks' properties.                                                                                                                              |
| fina2.bank.LicenceBean             | Entity            | Allows creating, finding, and accessing bank licenses' properties.                                                                                                                      |
| fina2.bank.BranchBean              | Entity            | Allows creating, finding, and accessing bank branches' properties.                                                                                                                      |
| fina2.bank.BankGroupBean           | Entity            | Allows creating, finding, and accessing bank groups' properties.                                                                                                                        |
| fina2.bank.LicenceTypeBean         | Entity            | Allows creating, finding, and accessing bank license types' properties.                                                                                                                 |
| fina2.period.PeriodTypeBean        | Entity            | Allows creating, finding, and accessing period types' properties.                                                                                                                       |
| fina2.period.PeriodBean            | Entity            | Allows creating, finding, and accessing periods' properties.                                                                                                                            |
| fina2.period.PeriodSessionBean     | Stateless Session | Contains methods to manage tables of periods and period types see 5.3.3. EJBTable pattern.                                                                                              |
| fina2.returns.ReturnTypeBean       | Entity            | Allows creating, finding, and accessing return types' properties.                                                                                                                       |
| fina2.returns.ReturnDefinitionBean | Entity            | Allows creating, finding, and accessing return definitions' and tables' properties.                                                                                                     |
| fina2.returns.ReturnSessionBean    | Stateless Session | Contains methods to manage tables of returns, return definitions, schedules, return types (See 5.3.3. EJBTable pattern). Change statuses of returns and get history of return statuses. |
| fina2.returns.ProcessSessionBean   | Stateless Session | Contains methods for return processing, import, file robot.                                                                                                                             |
| fina2.returns.ScheduleBean         | Entity            | Allows creating, finding, and accessing schedules'                                                                                                                                      |

|                                                 |                   |                                                                                                        |
|-------------------------------------------------|-------------------|--------------------------------------------------------------------------------------------------------|
|                                                 |                   | properties.                                                                                            |
| fina2.reportoo.server.ReportBean                | Entity            | Allows creating, finding, and accessing reports' properties.                                           |
| fina2.reportoo.server.OOReportSessionBean       | Stateless Session | Contains methods to manage reports tree. (See 5.3.2. EJBTree pattern). Generate and store reports.     |
| fina2.reportoo.repository.RepositorySessionBean | Stateless Session | Contains methods to manage hierarchy of formulas in reporting repository (See 5.3.2. EJBTree pattern). |

#### 4.9. Transactions' Types

Transaction management in FinA is carried out by the Application Server. The following types of transactions are used in FinA:

1. Bean Managed Transactions
2. User Transactions
3. Container Managed Transactions

1. Bean Managed Transactions are used for operations that require more time to perform, such as Return Processing and Report Generation. In these cases selection of source data and calculation of results occurs before the transaction takes place. The transaction is initiated inside of Session Bean method before the results of the calculation must be stored in the database.

For example, the fina2.returns.ProcessSessionBean performs return processing by collecting the values of the nodes involved in the processing of a given return without actually performing the transaction and therefore does not lock the database for other users. After collection of source data and calculations are finished, it initiates transaction in the following way:

```
javax.transaction.UserTransaction tran =
 jndiContext.getUserTransaction();

tran.begin();
try {
 /* perform database update with calculated values */
 ...
 tran.commit();
} catch(Exception e) {
 tran.rollback();
}
```

2. User Transactions are used in cases where the transaction must be performed by the Client application. For example, when Amend Dialog has to store its fields by using several methods of Entity Bean, if an error occurs while using one of these methods, actions which were already performed by the other methods, must be cancelled. This type of transaction is only used in the EJBTree & EJBTable patterns.

3. Container Managed Transactions are used in all other cases and have the “Required” attribute.

Information about which types of transactions are used in Beans can be found in EJB application file - fina2\_server.jar/META-INT/ejb-jar.xml

#### **4.10. Security**

Authorization and users’ rights control is realized with features that are part of the FinA application and do not use the features of the Application Server or the operating system.

All classes, which are associated with security, are located in the package fina2.security.

##### **4.10.1. Entity Beans**

Entity Bean – fina2.security.UserBean is the fundamental component of the FinA security system. Any part of the program can get a handle (an object that identifies an Enterprise Bean) to the copy of the UserBean of a currently logged in user and check its rights through the assistance of UserBean.hasPermission(), UserBean.canAccessBank(), etc.

##### **4.10.2. Client**

The Client Application checks the user’s permissions only when it has to disable or enable different buttons and correctly display the user interface. This operation happens through UserBean.hasPermission() method.

##### **4.10.3. Server**

In other cases the control of users rights is performed by the server and the client only has to catch fina2.PermissionDeniedException to show the appropriate message dialog. For example fina2.returns.ReturnSessionBean contains the method getReturnDefinitionsRows() which returns the list of return definitions. This method checks who has user rights to review or amend return definitions through the assistance of the UserBean.hasPermission(“fina2.returns.definition.amend”) & UserBean.hasPermission(“fina2.returns.definition.review”). If the user does not have these permissions and tries to open the return definitions dialog window which calls getReturnDefinitionsRows(), the server will throw the message PermissionDeniedException to the client and the client application must catch it and show dialog window with appropriate error message.

Usage of UserBean and hasPermission() method is eliminated in EJBTREE & EJBTABLE patterns.

The diagram in *Figure 4.10.3, Logic Sequence* shows user login sequence.

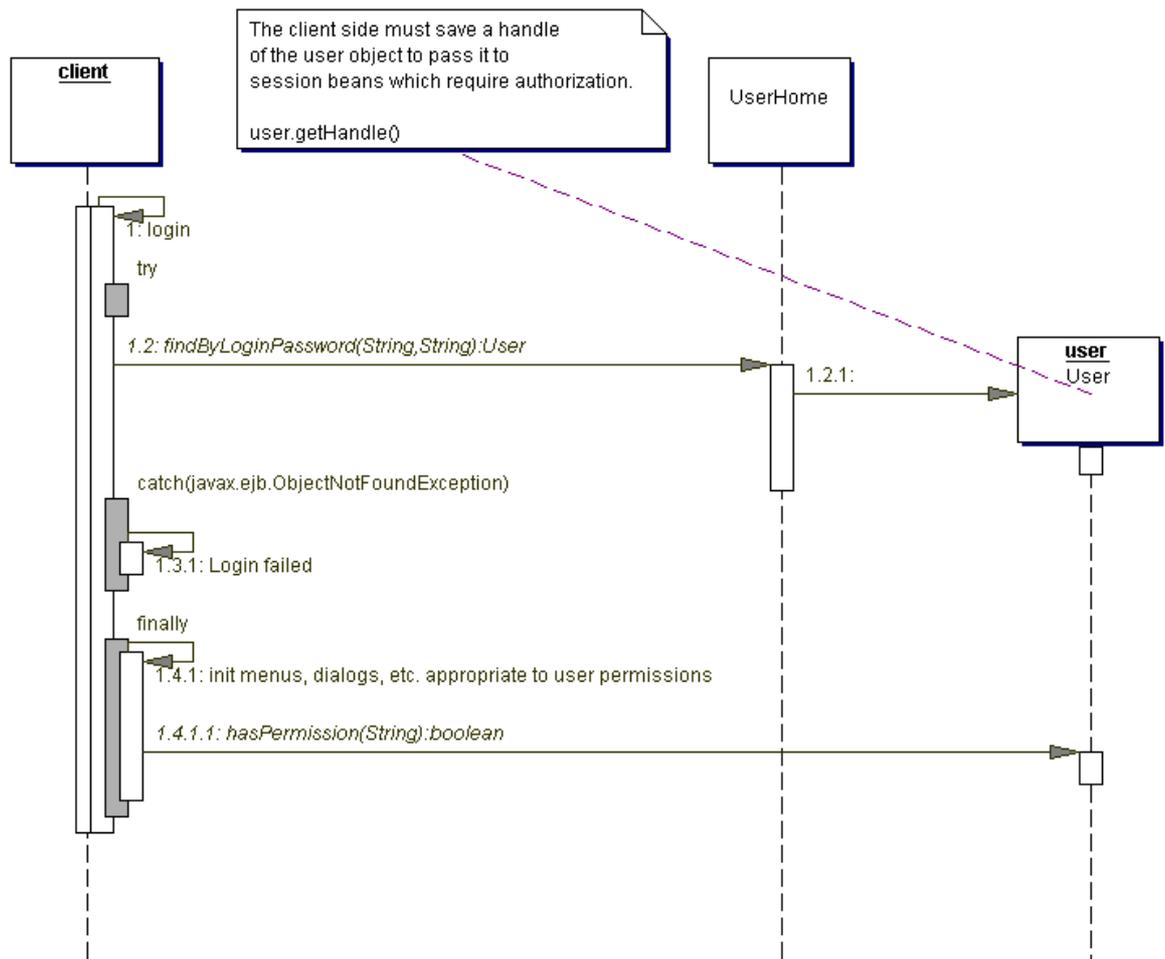


FIGURE 4.10.3, LOGIC SEQUENCE

The list of permissions available in the system is stored in table SYS\_PERMISSIONS in the database and is accessible through UserSessionBean.

The names of the permissions are based on the name of the package to which they are related. For example, the right to edit the Metadata Tree will have the key “fina2.metadata.amend”.

#### 4.11. Return Processing

Return processing happens with the assistance of the fina2.ui.ProcessSessionBean. This Stateless Session Bean has following functionality:

- ❑ Calculates the values of “variable” nodes
- ❑ Controls and updates the statuses of the returns
- ❑ Controls versions of returns
- ❑ Checks dependencies between returns and performs necessary calculations of related variables

See the Sequence Diagram in *Figure 4.11.1*.

As shown on the diagram "Processing" consists of following steps:

- ❑ Checks if the current user has permission to process returns or not
- ❑ Finds returns and all its variable nodes in the database
- ❑ Finds nodes from other returns on which current return depends
- ❑ Executes scripts of variable nodes for each return that is involved

- Updates the status and version of each modified return
- Updates the database with calculated values

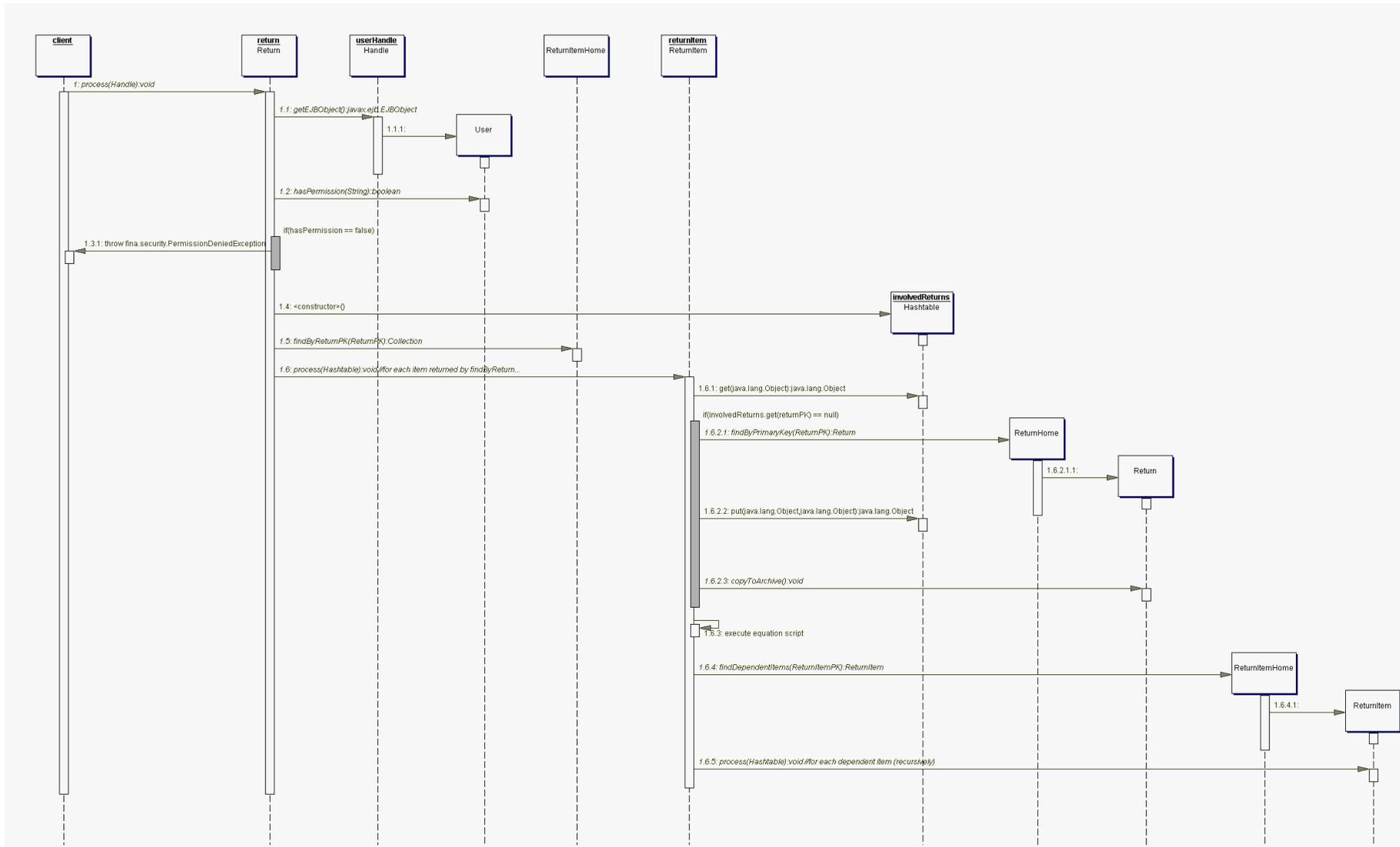


FIGURE 4.11.1, SEQUENCE DIAGRAM

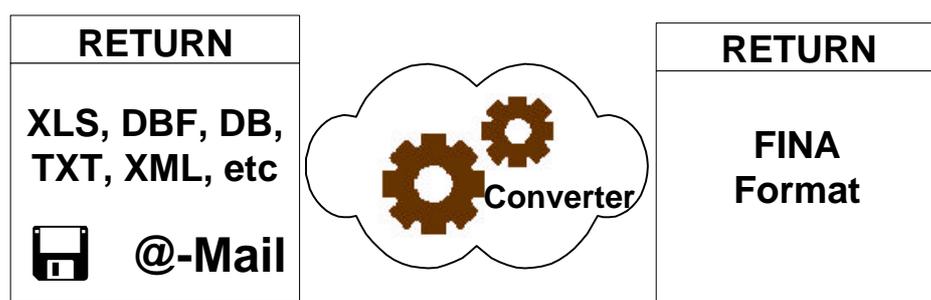
Netscape's Java Script interpreter, Rhino, is used for variables calculations during processing. Its source code, binaries, and documentation are available at <http://www.mozilla.org/>.

Scripts can be executed by using the fina2.script.Engine class which contains static methods for that purpose. This class was developed in order that the system would be able to use other Java Script interpreters. It is possible to replace Rhino with another interpreter.

## 4.12. Database

Financial information from banks (returns) are submitted in different types of files (Excel, text, etc.) depending on the regulations in each country. These files must be converted to an XML file that is compatible with FinA before it can be processed by the system and placed in the database. The DTD of the FinA version of XML can be found in **DOC/XML\_Return.dtd** file. An example of a converter from an XLS file to FinA XML file can be found in **DOC/Xls2Xml.zip**. The converter maps data from existing format (XLS, DBF, TXT, etc.) to the FinA structure described as Metadata.

Data is stored in the following table: In\_Return\_Items.



### 4.12.1. Tables Structure Overview

The FinA Database contains three types of tables. Each of these types of tables begins with different prefixes SYS, IN, and OUT:

- SYS tables contain system information such as menu structure, users' rights, locale descriptions, and text strings in different languages.
- IN tables contain the meta-structure of input data, information about banks, periods, and input data.
- OUT tables describe output data of the system – reports and all related information.

Three main types of data stored in the FinA database are:

- structure of the input forms (Returns)
- data from returns
- output forms (Reports)

The structure of Returns consists of the metadata tree, return definitions, and return types. This structure must be created based on the specific banking regulations in each country. The Metadata Tree is stored in the tables, which have names that start with IN\_MDT\_...

Return definitions are stored in IN\_RETURN\_DEFINITIONS and IN\_DEFINITION\_TABLES tables.

Return types are stored in the IN\_RETURN\_TYPES table.

Returns<sup>1</sup> and their data are stored in IN\_RETURNS and IN\_RETURN\_ITEMS tables.

Definition of output reports is stored in table OUT\_REPORTS.

The IN\_RETURN\_ITEMS table is the largest in the system. It has numerous indices and requires special attention during the composition of queries. This table continuously collects data from the returns and its size increases over time more than any of the other tables.

IN\_RETURNS contains the list of returns. This table contains only the headers and the values are in the IN\_RETURN\_ITEMS table.

IN\_RETURN\_STATUSES contains the history of modifications to the returns.

IN\_RETURN\_TYPES contains the list of return types.

IN\_SCHEDULES contains the list of schedules.

IN\_MDT\_NODES contains the Metadata tree hierarchy.

IN\_MDT\_DEPENDENT\_NODES contains the dependencies between nodes in the Metadata tree.

IN\_MDT\_COMPARISON contains comparison rules for the Metadata tree nodes.

IN\_RETURN\_DEFINITIONS contains return definitions.

IN\_DEFINITION\_TABLES contains a description of the return definition tables.

IN\_BANKS contains the list of banks and information on these banks.

IN\_BANK\_GROUPS contains the list of bank peer groups.

IN\_BANK\_TYPES List of bank types.

IN\_BANK\_BRANCHES contains information about bank branches.

IN\_PERIODS contains a list of the periods for the bank returns.

IN\_PERIOD\_TYPES contains the list of period types.

---

<sup>1</sup> The data in the returns is submitted by the commercial banks in the format that is specified by the banking regulations in that country. This data may be received in a number of different formats and it must be converted to the FinA version of XML in order for it to be processed by FinA and placed in the database. DTD of this XML can be found in DOC/XML\_Return.dtd file.

The SYS\_STRINGS table is also important because it has relations with many other tables. It contains text strings that connected to concrete objects and has versions in different languages.

The SYS\_LANGUAGES table describes the languages found in the system.

The SYS\_MENUS table contains the menu structure.

The SYS\_PERMISSIONS table contains the list of permissions that are available in the system.

The SYS\_ROLES table contains the list of the user roles.

The SYS\_ROLE\_PERMISSIONS tables contain the permissions for each user role.

The SYS\_USERS table contains user information.

The SYS\_USER\_BANKS table contains banks access rights for each user.

The SYS\_USER\_MENUS table contains the menus that are accessible for each user.

The SYS\_USER\_PERMISSIONS table contains the permissions for each user.

The SYS\_USER\_REPORTS table contains the access rights for reports for each user.

The SYS\_USER\_ROLES table contains the roles assigned to each user.

*(See database diagram. /DOC/ DataBaseDiagram/index.html)*

#### 4.12.2. Stored Procedures

Stored Procedures are used in the two parts of FinA that process large volumes of information and are performance critical:

- Return Processing
- Reporting

Procedures involved in Return Processing:

|                          |                                                                                                                                                                            |
|--------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| prepare_process_items    | This procedure creates a temporary table IN_PROCESS_ITEMS and fills it with values from a specified return. This table will be used by other procedures during processing. |
| get_all                  | This procedure selects values from the Return and sends them to a result set.                                                                                              |
| get_dependents           | This procedure selects the values of the nodes from other Returns on which variables from the specified Return depend.                                                     |
| get_process_items        | This procedure selects the whole IN_PROCESS_ITEMS table.                                                                                                                   |
| calculate_node           | This procedure calculates the sum, average, max, min, or count of sub-nodes according to node's "evaluation type" and updates the IN_RETURN_ITEMS with calculated values.  |
| var_count                | This procedure calculates the number of variables in the specified Return.                                                                                                 |
| deallocate_process_items | This procedure saves a temporary table to IN_PROCESS_ITEMS.                                                                                                                |

### **4.13. Procedures involved in Reporting**

The procedures for Reporting are not full-fledged stored procedures. The procedures that are dynamically generated during report generation are not permanently stored in the database. All calculations and data groupings performed by these procedures are done by the SQL server. The results of these calculations are placed in a temporary table called “rep\_result”.

The procedures for Reporting perform the following operations as part of a single transaction:

1. Calculate periods’ offsets and recognize which periods are involved in the report
2. Sort recognized periods and place their IDs and sequence numbers in temporary table “per\_offsets”
3. Select values of return items and place them in temporary table “middle”
4. Perform data grouping and calculations of PEER/ALL BANKS/PCT aggregates according to banks, peer groups, periods, etc., with help of cursors and data collected in tables “middle” and “per\_offsets”
5. Place results of the in “rep\_result” table
6. Delete temporary tables by selecting the whole “rep\_result” table and deleting it from the database.

### **4.14. Configuration**

The database connection configuration takes place in the JBoss configuration files and is described in the “FinA Site Customization Manual”.

FinA uses JDBC driver from Microsoft for MS SQL Server, which is available at <http://www.microsoft.com/>.

JBoss has built-in connection pool capabilities. The size of the pool and the maximum number of connections that can be opened at the same time can be configured by the application server tools. The maximum number of simultaneous connections is also related to type and license of SQL server that is used.

### **4.15. Patterns**

#### **4.15.1. Overview**

In order to unify the overall development process, similar functions in the system follow the same patterns. These patterns were developed based on an analysis of the system's functionality.

The user interface of FinA is based on three main types of windows, which display following types of information:

- Tree
- Table
- List

Appropriate patterns are designed for each of these types of windows.

### 4.15.2. EJBTree Pattern

The EJBTree Pattern describes collaboration between the Client side GUI and server side EJBs. All data in the system that must have a tree representation should be processed using this pattern. A chart is presented on *Figure 4.15.2.1*.

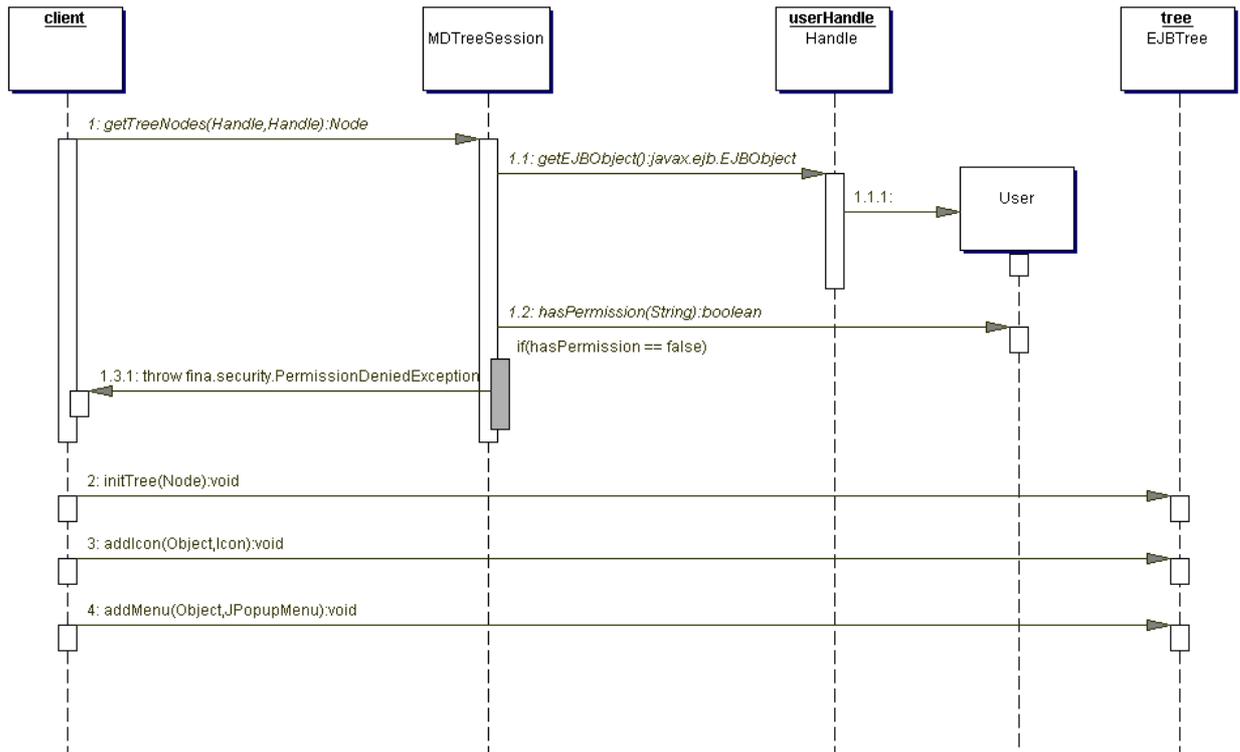


FIGURE 4.15.2.1, EJB TREE PATTERN

For each data type (such Metadata Tree, Menu Tree, Users/Roles Tree, etc.) an appropriate Session Bean exists. For example the diagrams in this chapter illustrate the `fina2.metadata.MDTSessionBean` for Metadata Tree.

These Stateless Session Beans have the `getTreeNodes()` method which allows for the retrieval of the structure of a given tree from the database and the returns of its root node. The root node is then passed as a parameter to `fina2.ui.tree.EJBTree.initTree()` and a tree is displayed.

The `fina2.ui.tree.Node` class contains primary key and type properties. It is possible to specify different icons and pop-up menus for different types of nodes in the EJBTree.

For example, after an amend action is performed in a pop-up menu or button, the action listener should trigger the following operations:

- obtain selected node from the tree
- find appropriate Entity Bean by node's primary key
- show Amend dialog box for the Entity Bean and get/set its fields

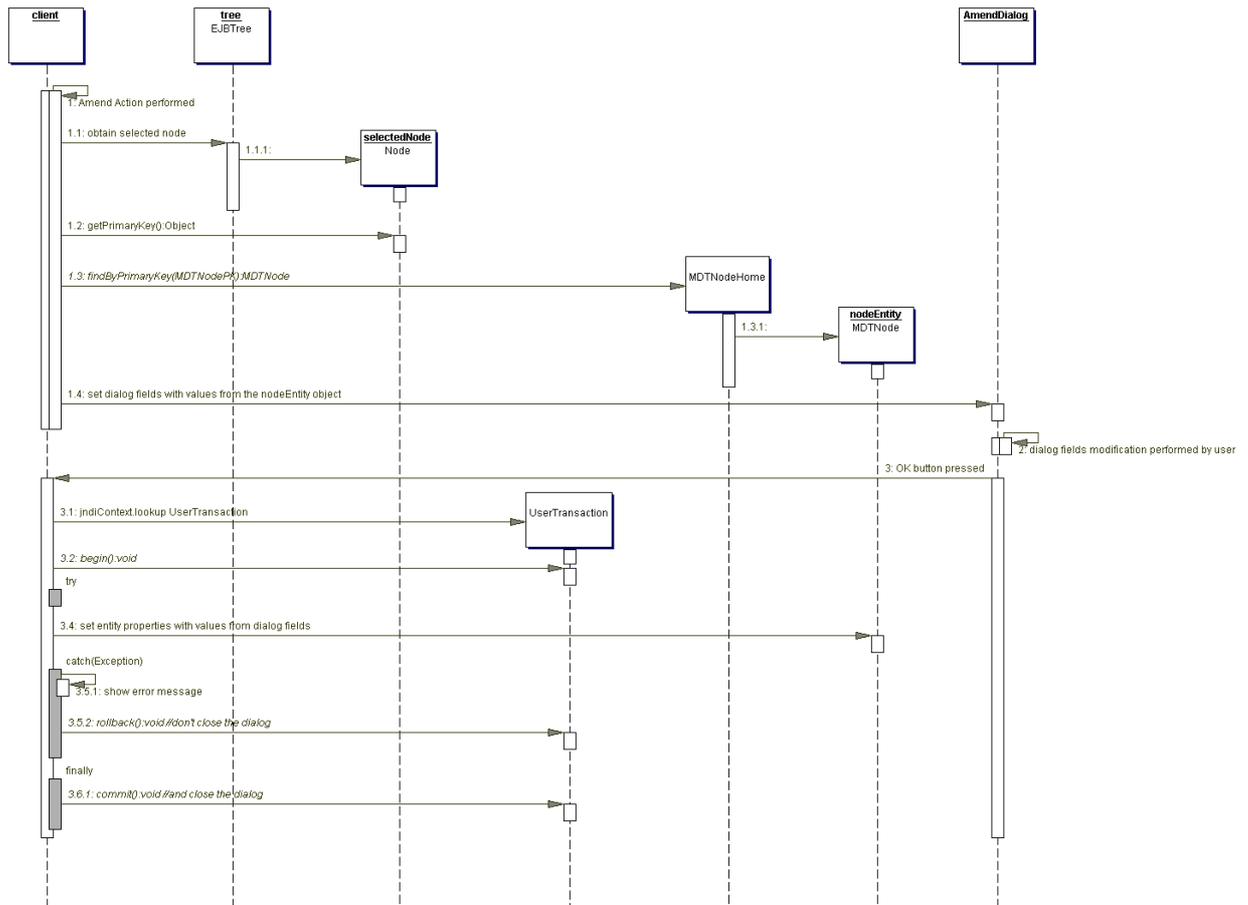


FIGURE 4.15.2.2, AMEND ACTION LOGIC

Entity Beans perform validation of values passed to them using “set” method and in case the value is not acceptable they throw java.beans.PropertyVetoException with description message.

Parts where EJB Tree pattern is used:

| Name           | Package        | Client classes                                                                       | Server classes                  |
|----------------|----------------|--------------------------------------------------------------------------------------|---------------------------------|
| Metadata Tree  | fina2.metadata | MDTAmendFrame<br>MDTInputAmendDialog<br>MDTNodeAmendDialog<br>MDTVariableAmendDialog | MDTSessionBean<br>MDTNodeBean   |
| Banks          | fina2.bank     | BanksFrame<br>BankAmendDialog                                                        | BankSessionBean<br>BankBean     |
| User Manager   | fina2.security | UsersFrame<br>UserAmendDialog<br>RoleAmendDialog                                     | UsersSessionBean<br>UsersBean   |
| Report Manager | fina2.reportoo | ReportManagerFrame<br>Designer<br>Generator<br>FolderAmendDialog                     | ReportSessionBean<br>ReportBean |

### 4.15.3. EJBTable Pattern

This pattern should be applied to data which must be represented as rows of a table or list.

Similar to the EJBTree Pattern, for each data type (such as Return Definitions, Returns, Periods, etc.) a specific Session Bean exists which retrieves the table's data as collection of rows.

The collection of  `fina2.ui.table.TableRow`  objects returned by Session Bean should be passed to  `fina2.ui.table.EJBTable.initTable()`  method.

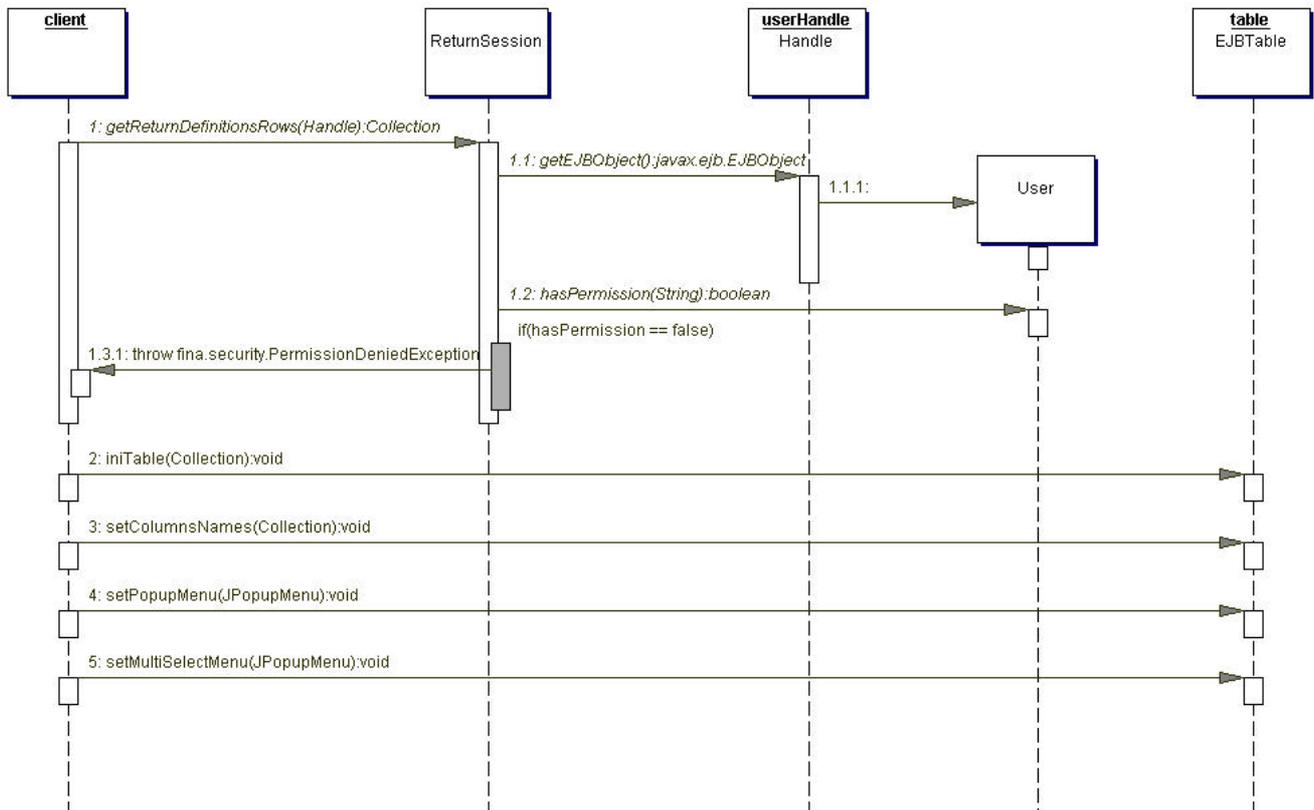


FIGURE 4.15.3.1, EJB TABLE PATTERN

Pop-up menus for table rows may be set by EJBTable.setPop-upMenu() method.

Processing of actions in this pattern happens in same manner as in EJBTree Pattern.

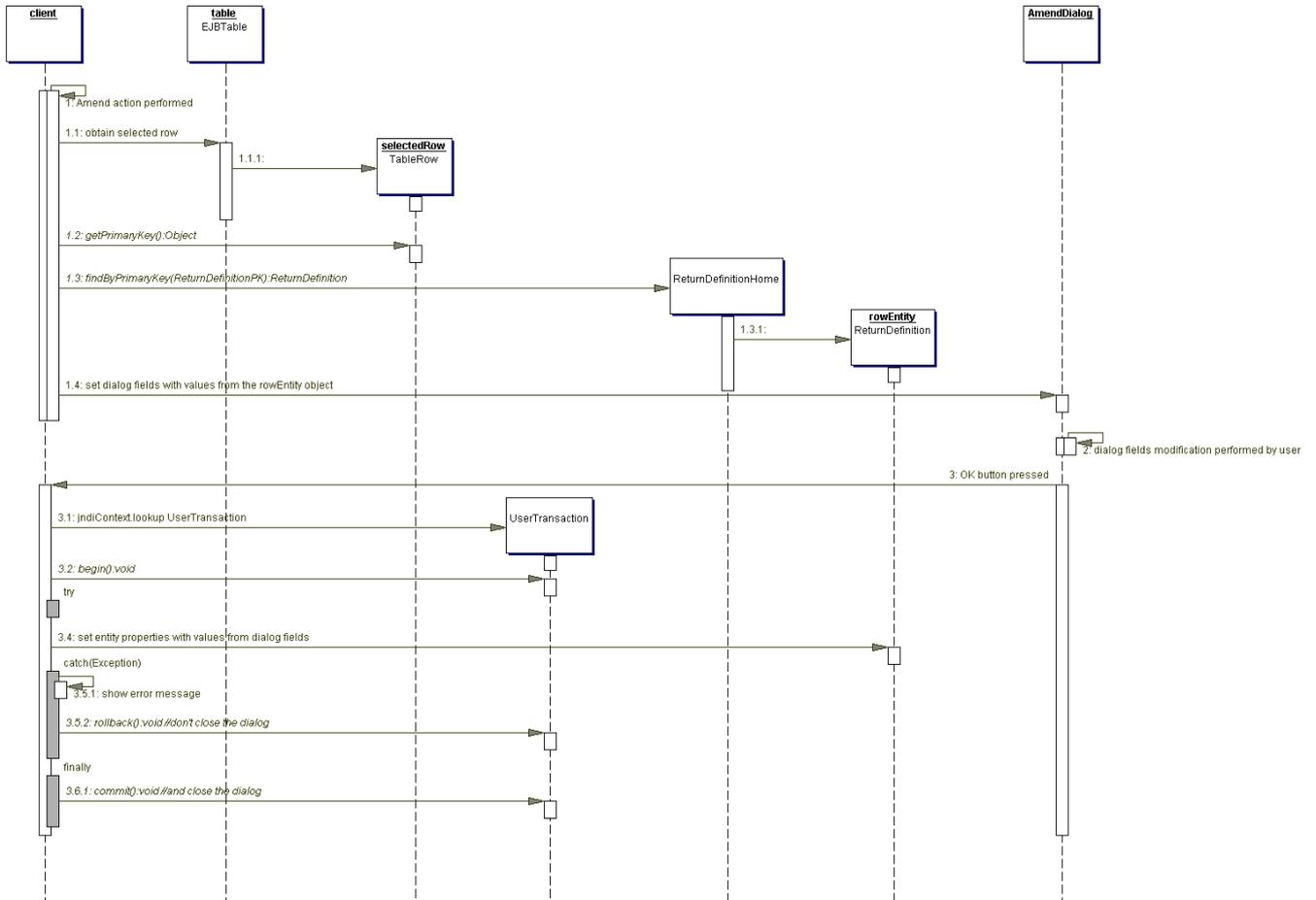


FIGURE 4.15.3.2, POP-UP MENU PATTERN

Functions of FinA that use the EJBTable pattern:

| Name               | Package        | Client classes                                                                       | Server classes                            |
|--------------------|----------------|--------------------------------------------------------------------------------------|-------------------------------------------|
| Periods            | Fina2.period   | PeriodFrame<br>PeriodAmendFrame                                                      | PeriodSessionBean<br>PeriodBean           |
| Locales            | Fina2.i18n     | LanguagesFrame<br>LanguageAmendDialog                                                | LanguageSessionBean<br>LanguageBean       |
| Comparison Rules   | fina2.metadata | ComparisonsAmendFrame<br>NodeComparisonAmendDialog                                   | MDTSessionBean                            |
| Return Definitions | fina2.returns  | ReturnDefinitionsFrame<br>ReturnDefinitionAmendDialog<br>ReturnDefinitionTableDialog | ReturnSessionBean<br>ReturnDefinitionBean |

|                |               |                                                                    |                                   |
|----------------|---------------|--------------------------------------------------------------------|-----------------------------------|
| Schedules      | fina2.returns | SchedulesFrame<br>SchedulesAutoInsertFrame<br>SchedulesAmendDialog | ReturnSessionBean<br>ScheduleBean |
| Return Manager | fina2.returns | ReturnManagerFrame<br>ReturnAmendDialog                            | ReturnSessionBean<br>ReturnBean   |
|                |               |                                                                    |                                   |

#### 4.16. Reporting

Reporting is performed by OpenOffice Calc. The entire report generation process takes place on the server and is placed in the fina2.reportoo package.

The Report generation process contains following steps:

- ❑ Insert necessary number of columns and rows for iterators
- ❑ Replace all CURNODE(), CURBANK() and other CUR...() functions with appropriate values from iterators or parameters
- ❑ Execute calculation of spreadsheet formulas to collect information about all combinations of nodes/banks/peer groups/periods/functions which are used in report
- ❑ Generate and execute SQL procedures based on information collected during the previous step which selects source data from database
- ❑ Perform final formulas calculations with existing source data
- ❑ Return generated spreadsheet to client

The main classes of reporting used on client are fina2.reportoo.Designer and fina2.reportoo.Generator.

The fina2.reportoo.ReportSessionBean is used on server for report management and generation purposes.

##### 4.16.1. OpenOffice.org CALC

OpenOffice.org is the open source project through which Sun Microsystems has released the technology for the StarOffice[tm] Productivity Suite. OpenOffice.org CALC is fully integrated with the FinA client. Therefore the user will not need to install OpenOffice Calc. separately. The installation and start-up will be part of the FinA client installation.

OpenOffice.org has been built for Linux (RedHat 6.2), Solaris (Sparc 2.6) and Windows platforms Languages for the OpenOffice-API: OpenOffice implements the API with UNO (Universal Network Objects). Currently there are language bindings for Java and C++. See [www.openoffice.org](http://www.openoffice.org) for further information on Open Office.org Calc.

OpenOffice Calc is used by the Client and Server side of FinA in different ways.

The FinA Client uses OpenOffice Calc for data presentation such Review, Print, Return, Amend, etc., and also for report design.

GUI is not necessary on server side. The server uses OpenOffice Calc for calculations at report generation time.

OpenOffice Calc provides two ways to access its functionality from an external application with help of UNO API by two ways:

1. Using native pipes of Operational System
2. Using TCP/IP sockets

See <http://www.openoffice.org/>

The first way allows for the manipulation of OpenOffice GUI and the embedding of its GUI into the external application's GUI. The second way does not allow access to the GUI but permits using spreadsheet's data manipulation functionality.

#### ❑ Client

The Client uses the Openofficebean to access the OpenOffice API (<http://whiteboard.openoffice.org/OOBean/>). The Openofficebean contains two parts: Java and the native libraries for all platforms supported by OpenOffice. This simplified the development of the interaction between FinA and OpenOffice as the native part of code was already written for the use of the OpenOffice GUI in FinA.

#### ❑ Server

The Server accesses OpenOffice with help of the UNO API and uses sockets for this purpose. FinA uses the Java class library to access Open Office Calc services through the UNO API. This library contains Java interfaces and its specifications are available at <http://www.openoffice.org/>. The JAR file containing this library is placed in the server's deploy directory and in client's lib/ext directory.

OpenOffice does not allow running more than one instance on a single machine and can use only one method of interaction (native pipes or sockets) in one instance. Therefore, with the current version of OpenOffice Calc, it is not possible to run client and server together on a single machine.

FinA uses “driver”-based architecture (FinA – driver – spreadsheet). The same driver is used by client and server. (*See driver specifications in [Doc/Driver\\_JavaDoc/index.html](#)*).

### **4.17. Application Server**

The JBoss 3.0 Application Server is in the FinA system. It is possible to switch to other Java 2 Enterprise Edition compatible Application Server without changes in the FinA source code.

To replace JBoss:

1. Build fina2\_server.jar with help of jar build tool provided with the specific App Server
2. Configure database connection “Fina2DS” with appropriate tools and/or configuration files (depends on App Server)
3. Deploy fina2\_server.jar into server (also depends on App Server)
4. Modify configuration file (conf/jndi.properties) on client workstation
5. Place client class libraries provided by App Server vendor to lib/ext folder on client workstation

This process completely depends on App Server and not on FinA code.

Currently fina2\_server.jar contains the FinA application server and is located in FinA\_Install\_Dir/server/jboss/server/default/deploy folder. The database connection configuration file mssql-service.xml is also located in this folder.

The Server application starts automatically after JBoss is started on command line using the following batch file:

FinA\_Install\_Dir/server/jboss/bin/run.bat.

*For additional information please see sourcecode.zip*

#### **4.18. Configuration Properties**

##### ❑ Client properties

All configuration files of client workstation are placed in the conf/ folder.

- ❑ jndi.properties
- ❑ messages\_xx\_XX.properties
- ❑ fina2.conf

jndi.properties file contains parameters of JNDI connection which are used to make the connection to the server. These parameters are not specific to FinA. They are part of JNDI specification ( <http://java.sun.com/products/jndi/> ).

messages\_xx\_XX.properties is a message bundle. It can be modified according to the “FinA Installation & Configuration Manual 5.4.2”.

fina2.conf contains configurations parameters such windows' positions, dimensions, and other values which are stored by different parts of FinA. This file is binary and contains serialized classes.

##### ❑ Server Properties

There are no FinA specific configuration files on the server. The configuration of the application Server configuration must be performed according to its manual.

#### **4.19. Build (version) Management**

This section outlines the build procedures used in the development of the FinA software.

#### **4.20. Development Environment**

- ❑ The application has been written entirely in Java using Forte 4 on a Windows 2000 platform against a Microsoft SQL Server 7/2000 database
- ❑ The Java JDBC drivers are used to interface with the database engine
- ❑ Swing components are used throughout the UI
- ❑ The "Install anywhere" software was used to build the installation package
- ❑ Visio was used as database modeling and structure control tool

#### **4.21. Directories**

The following are directories under C:\FinA\current live build:

For Client

- ❑ C:\Program Files\fina2client\
- ❑ C:\Program Files\fina2client\conf - For Configuration Files
- ❑ C:\Program Files\fina2client\fina2 - Main class directory
- ❑ C:\Program Files\fina2client\import - for imported files
- ❑ C:\Program Files\fina2client\lib - Library folder

For Server

- ❑ C:\Program Files\FinA2Client\
- ❑ C:\Program Files\FinA2Client\server
- ❑ C:\Program Files\FinA2Client\server\audit
- ❑ C:\Program Files\FinA2Client\server\jre
- ❑ C:\Program Files\FinA2Client\server\jboss
- ❑ C:\Program Files\FinA2Client\server\jboss\bin
- ❑ C:\Program Files\FinA2Client\server\jboss\catalina
- ❑ C:\Program Files\FinA2Client\server\jboss\client
- ❑ C:\Program Files\FinA2Client\server\jboss\lib
- ❑ C:\Program Files\FinA2Client\server\jboss\ooconf
- ❑ C:\Program Files\FinA2Client\server\jboss\server
- ❑ C:\Program Files\FinA2Client\server\jboss\server\all
- ❑ C:\Program Files\FinA2Client\server\jboss\server\minimal
- ❑ C:\Program Files\FinA2Client\server\jboss\server\default
- ❑ C:\Program Files\FinA2Client\server\jboss\server\default\conf
- ❑ C:\Program Files\FinA2Client\server\jboss\server\default\db
- ❑ C:\Program Files\FinA2Client\server\jboss\server\default\deploy
- ❑ C:\Program Files\FinA2Client\server\jboss\server\default\lib
- ❑ C:\Program Files\FinA2Client\server\jboss\server\default\log
- ❑ C:\Program Files\FinA2Client\server\jboss\server\default\tmp

There will also be a directory C:\FinA~~XXX~~ and C:\FinAServer~~XXX~~, where ~~XXX~~ is the current live build number. For instance "C:\FinA0018b4", C:\FinAServer0018b4" or "C:\FinA0018c5", "C:\FinAServer0018c5" etc. This directory is constructed from the live build directories and contains all the files and directories on the CD for the specific release of FinA.

#### **4.22. The Build Procedure**

The build procedures are as follows:

1. Before copying the new build backup (copy) c:\fina folder to the c:\FinA~~xxx~~ folder where ~~xxx~~ is build number.
2. Copy the new source code received from the software engineers into the C:\FinA\ directory and the new version is compiled completely with Forte (see Forte 4.0 documentation ).
3. Once the build manager is satisfied that the new software does not compromise the system, then the C:\FinA\ source directory is backed up by creating a zip file of that directory. This file is called ~~xxx~~Source.zip , where ~~xxx~~ stands for the build version number. This is then copied to the C:\FinA\ builds directory.
4. Run Forte and compile source code (see Forte 4.0 Documentation for directions).
5. Run c:\FinA\deploy.bat. It will create fina\_server.jar file, which must be copied to C:\FinAServer\deploy directory. FinA Server is ready to run or to build installation package (FinA~~sXXXXXXXX~~.exe where ~~XXXXXXXX~~ is build number).

### **5. Installation Package Build**

---

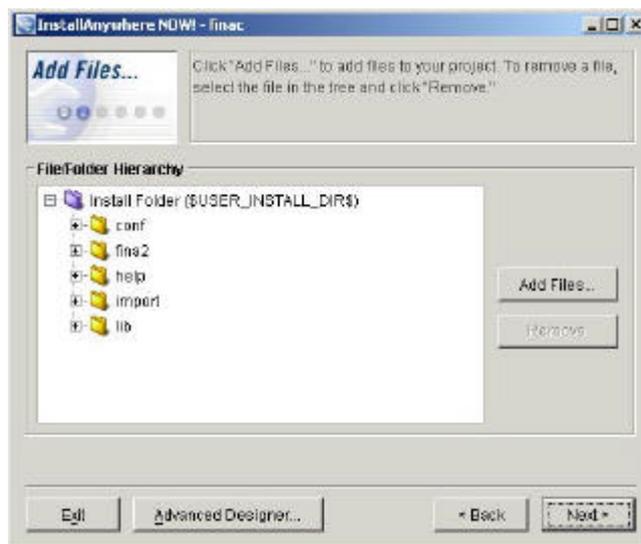
The folder c:\FinAinst is needed to build the installation package for FinA Server (fina~~sXXXXXXXX~~.exe) and the FinA Client (fina~~cXXXXXXXX~~.exe). It contains the following folders:

- C:\FinAinst\finac for FinA Client
  - C:\FinAinst\finas for FinA Server
1. Backup existing version of installation directory to c:\FinAinst~~XXXXXXXX~~ where ~~XXXXXXXX~~ is build number.
  2. Copy all files except the source code from c:\FinA to C:\FinAinst\finac and C:\FinAServer\ to C:\FinAinst\finas.

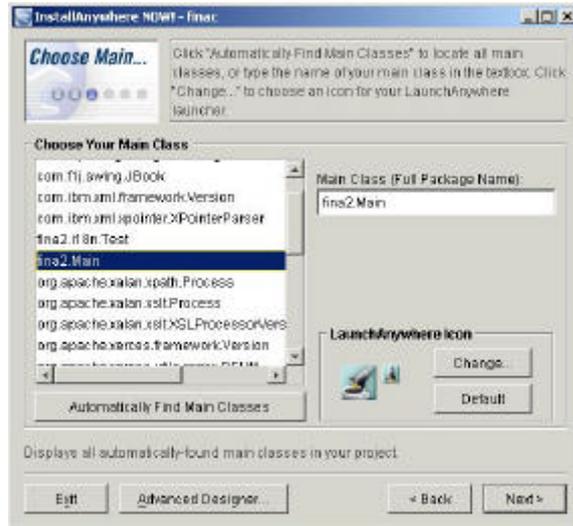
3. Run Install Anywhere, open project finac.



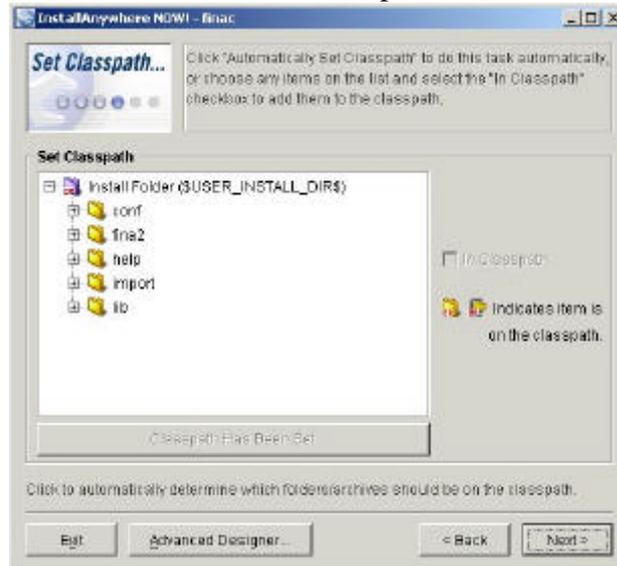
4. Update it with new files. You can click on the "Add Files" button, find the relevant directory, and then add the necessary files or all of the files.



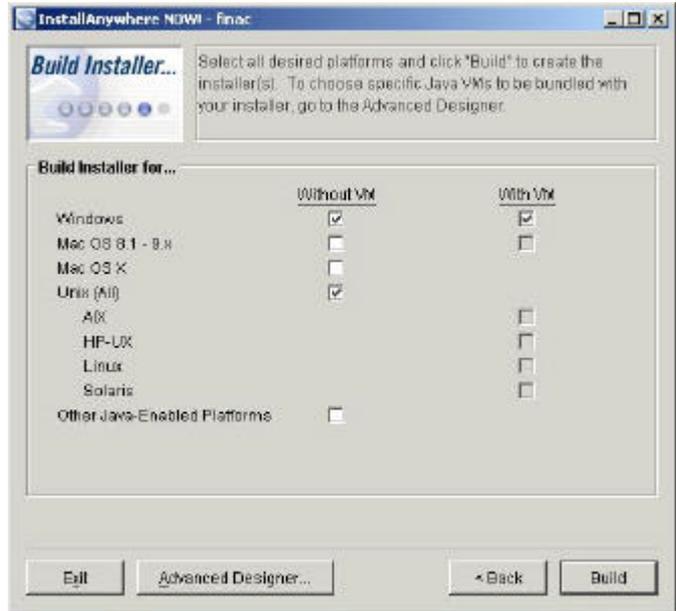
Click **Next**. The next step is to define the main classes. Put *fina2.main* in the Main class field of InstallAnywhere.



Click on the "Next" button and define the classpath.



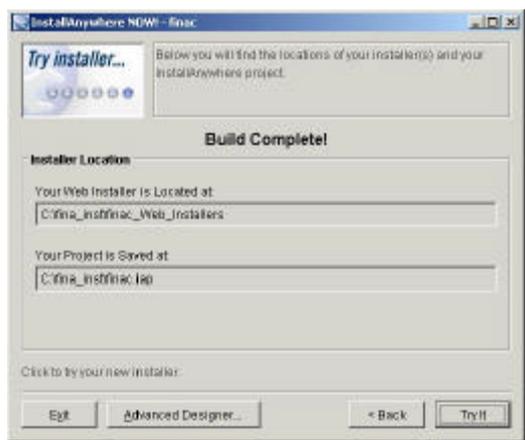
Click on the "Next" button. Choose the platform and JVM for package.



Click on the **Build** button and a progress bar will appear:



After the build procedure is finished, click on the **[Try it]** button and test the installation package.



5. Repeat the same procedure with FinAS for FinA Server

Installation packages (files: finac.exe and finas.exe) will be created into folder c:\FinAinst.

## 6. Database Modification Procedure

---

On the FinA CD that contains the new build, the user can find the scripts for updating the database. The names of the scripts have following construction:

dbupdNNXN\_NNXNN.sql where NNXN is build numbers; for instance:

dbupd18b5\_18c6.sql is a script which will update database version 0018b5 to version 0018c6. You have to run that script through Query Analyzer. (See MS SQL Server 2000 manual)

## 7. Coding Standards

---

In the development of FinA, Java Coding Standards as put forth by Doug Lea were used. See

<http://gee.cs.oswego.edu/dl> for a full copy of these standards.

### Structure and Documentation

**Packages:** Create a new java package for each self-contained project or group of related functionality. Create and use directories in accord with java package conventions. Consider writing an index.html file in each directory briefly outlining the purpose and structure of the package.

**Program Files:** Place each class in a separate file. This applies even to non-public classes (which are allowed by the Java compiler to be placed in the same file as the main class using them) except in the case of one-shot usages where the non-public class cannot conceivably be used outside of its context.

**Classes and Interfaces:** Write all `/** ... */` comments using *javadoc* conventions. (Even though not required by javadoc, end each `/**` comment with `*/` to make it easier to read and check.)

Preface each class with a `/** ... */` comment describing the purpose of the class, guaranteed invariants, usage instructions, and/or usage examples. Also include any reminders or disclaimers about required or desired improvements. Use HTML format, with added tags: *author-name*, *version number of class*, *string*, *URL*  
*classname#methodname*

**Methods:** Use *javadoc* conventions to describe nature, purpose, preconditions, effects, algorithmic notes, usage instructions, reminders, etc.

Use Running `//` comments to clarify non-obvious code, but try to make the code as obvious as possible.

## 8. Naming Conventions

---

Packages - lowercase.

Classes: CapitalizedWithInternalWordsAlsoCapitalized

Exception class: ClassNameEndsWithException.

Interface. When necessary to distinguish from similarly named classes:

`InterfaceNameEndsWithIfc.`

Class. When necessary to distinguish from similarly named interfaces:

`ClassNameEndsWithImpl` OR

`ClassNameEndsWithObject`

constants (finals):

`UPPER_CASE_WITH_UNDERSCORES`

private or protected: (pick one!)

`firstWordLowerCaseButInternalWordsCapitalized` OR

`trailingUnderscore_`, OR

`thisVar` (i.e. prefix with *this*), OR

`myVar` (i.e. prefix with *my*), OR

`fVar` (i.e. prefix with *f*)

static private or protected:

`firstWordLowerCaseButInternalWordsCapitalized` OR

`twoTrailingUnderscores__`

local variables:

`firstWordLowerCaseButInternalWordsCapitalized` OR

`lower_case_with_underscores`

methods:

`firstWordLowerCaseButInternalWordsCapitalized()`

factory method for objects of type X:

`newX`

converter method that returns objects of type X:

`toX`

method that reports an attribute x of type X:

`X x()` or `X getX()`.

method that changes an attribute x of type X:

`void x(X value)` or `void setX(X value)`.

See <http://gee.cs.oswego.edu/dl> for more details.

## 9. Code Security

---

FinA is designed using Java Code Security Guidelines. These guidelines are aimed at assisting programmers to write code so that it will not be vulnerable to security attacks. However, a chain only is as strong as its weakest link, and when new system code is added, so also is a new link to the security chain.

Java Code Security Guidelines: The Programmer should use following guidelines: Static fields; Reducing scope; Public methods and fields; Protecting packages; The equals methods; Make objects immutable if possible; Never return a reference to an internal array that contains sensitive data; Never store user-supplied arrays directly; Serialization; Native methods; Clear sensitive information; see <http://java.sun.com/security/seccodeguide.html#gcg2>.

For more details see /DOC/ProgGuide\_Apendif2\_iso15408-3.pdf.

## **10. Reference**

---

1. Forte 4.0 documentation <http://www.sun.com>
2. Install Anywhere manual <http://www.zerog.com/goto/iamanuals>
3. SQL Server documentation <http://www.microsoft.com>
4. JBoss Documentation <http://www.jboss.org>
5. Security Code Guidelines <http://java.sun.com/security/seccodeguide.html#gcg2>