TABLE OF CONTENTS

1 INTRODUCTION

1. Purpose

This document provides a comprehensive architectural overview of the new JAVA01 modules, using a number of different architectural views to depict different aspects of the modules. Being intended to capture and convey the significant architectural decisions that have been made about the modules, the document serves as a bridge between the software requirements and the detail design of JAVA01; it will also help software architects to ensure that the modules to be built will meet the needs of the users in term of functionality (selected functional view), of platform and technology (logical views).

2. Scope

From a high level point of view, the document defines the software architecture which addresses JAVA01's requirements in the following areas: functionality, availability, reliability, scalability, maintainability and manageability.

For the logical view, the new JAVA01 modules are best described by the introduction of n-tier architecture, which is reflected in the diagrams of the proposed structure.

3. Definitions, Acronyms and Abbreviations

#	Item	Description
1	JSP	Java Server Page
2	JSTL	Java Standard Tag Library
3	API	Application Programming Interface
4	HTML	Hypertext Mark-up Language
5	WYSIWYG	What You See Is What You Get
6	JVM	Java Virtual Machine
7	HTTP	Hypertext-Transfer Protocol
8	MVC	Model – View – Control
9	J2EE	Java 2 Enterprise Edition - A set of standards for implementing scalable, reliable enterprise solutions, from reusable components.

		These components come with a set of services which handle the behavior of the application.
10	DAO	Data Access Object, this object is responsible for attaching to a system, extracting some information, based on specific requirements, and creating a value object.
11	VO	Value Object
12	SPRING	Spring is a light weight framework to support the IoC dependency injection concept. It is becoming a widely adopted framework and has many books, articles and coding examples online. Spring's main aim is to make J2EE easier to use and promote good programming practice. It does this by enabling a POJO (plain old java object)-based programming model that is applicable in a wide range of environments.
13	HIBERNATE	Hibernate is a powerful, high performance object/relational persistence and query service. Hibernate lets developers to develop persistent classes following object-oriented idiom - including association, inheritance, polymorphism, composition, and collections. Hibernate allows developers to express queries in its own portable SQL extension (HQL), as well as in native SQL, or with an object-oriented Criteria and Example API
14	O-R	Object Relational

4. References

#	Name	Version	Date	Author
1	BRD.doc	1.0	18-Jan-2010	CUSTOMER

5. Overview

The following sections will provide a deeper inner look at the new JAVA01 modules architecture, explaining the functionality that the architecture can provide and also point out the scalability of the architecture to absorb later changes from user requirements or changes in the system interfaces with other third party applications/components. Viewing the architecture from the main angles such as **Functional** and **Logical**, we can ensure the architecture will satisfy all defined requirements and still allow later necessary expansion.

All related technologies that are to be applied for a specific purpose of the software architecture will also be represented here.

2 ARCHITECTURAL REPRESENTATION

System architecture

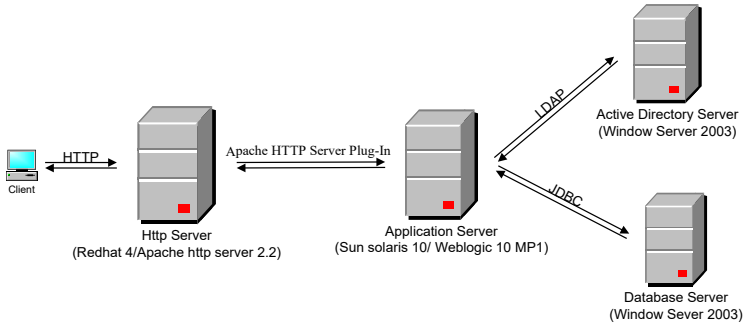

Figure 1: System architecture

N-tier architecture

The following diagram shows the primary tiers in the proposed n-tier architecture. This diagram shows the main layers in this architecture and the vision of how they fit together.

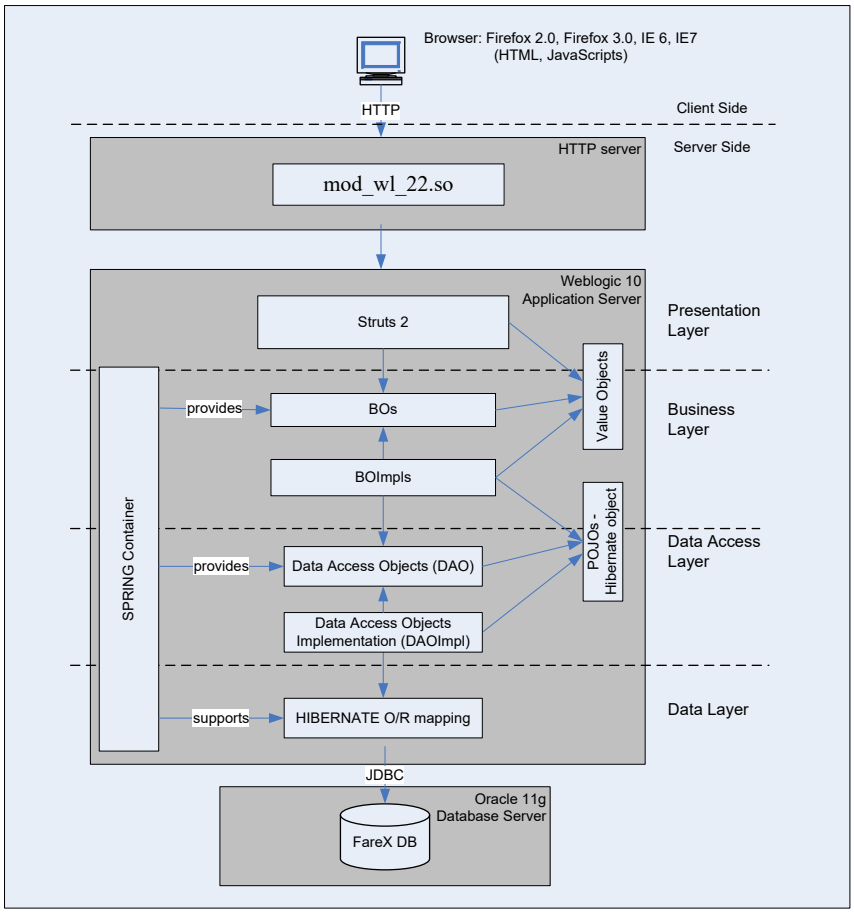

Figure 2 – N-tier architecture of JAVA01

1.2.1 Presentation Layer

This layer controls the display to the end user. For the presentation layer of JAVA01, the struts 2 framework is used. (The Struts 2 architecture is described in more detail in APPENDIX A.1:).

The framework is responsible for:

- Managing requests/responses from/to the clients.
- Controlling display to the end user.
- Assembling a model that can be presented in a view.
- Performing UI validation.
- Providing a controller to delegate calls to business logic and other upstream processes.
- Handling exceptions from other layers.

1.2.2 Business Layer

This layer manages the business processing rules and logic.

- Handling application business logic and business validation.
- Managing transactions.
- Allowing interfaces for interaction with other layers.
- Managing dependencies between business level objects.
- Adding flexibility between the presentation and the persistence layer so they do not directly communicate with each other.
- Exposing a context to the business layer from the presentation layer to obtain business services.
- Managing implementations from the business logic to the persistence layer.

1.2.3 Data Access Layer

This layer manages access to persistent storage. The primary reason to separate data access from the rest of the application is that it is easier to switch data sources and share Data Access Objects (DAOs) between applications.

- This layer manages reading, writing, updating, and deleting stored data.
- The HIBERNATE (as described in detail in APPENDIX A.3) is used as a persistence framework that manages O-R mapping in a seamless manner and enables developers to rapidly build applications that combine the best aspects of object technology and relational databases.

1.2.4 Data Layer

In JAVA01, the storage is managed by a relational database. Oracle 11g Database is used for this layer to provide the management of stored data.

3 LOGICAL VIEW

1. Overview

This section describes the significant packages/components of the architecture of the JAVA01 modules with their architectural design as well as their responsibilities.

2. Architecturally Significant Design Packages/Components

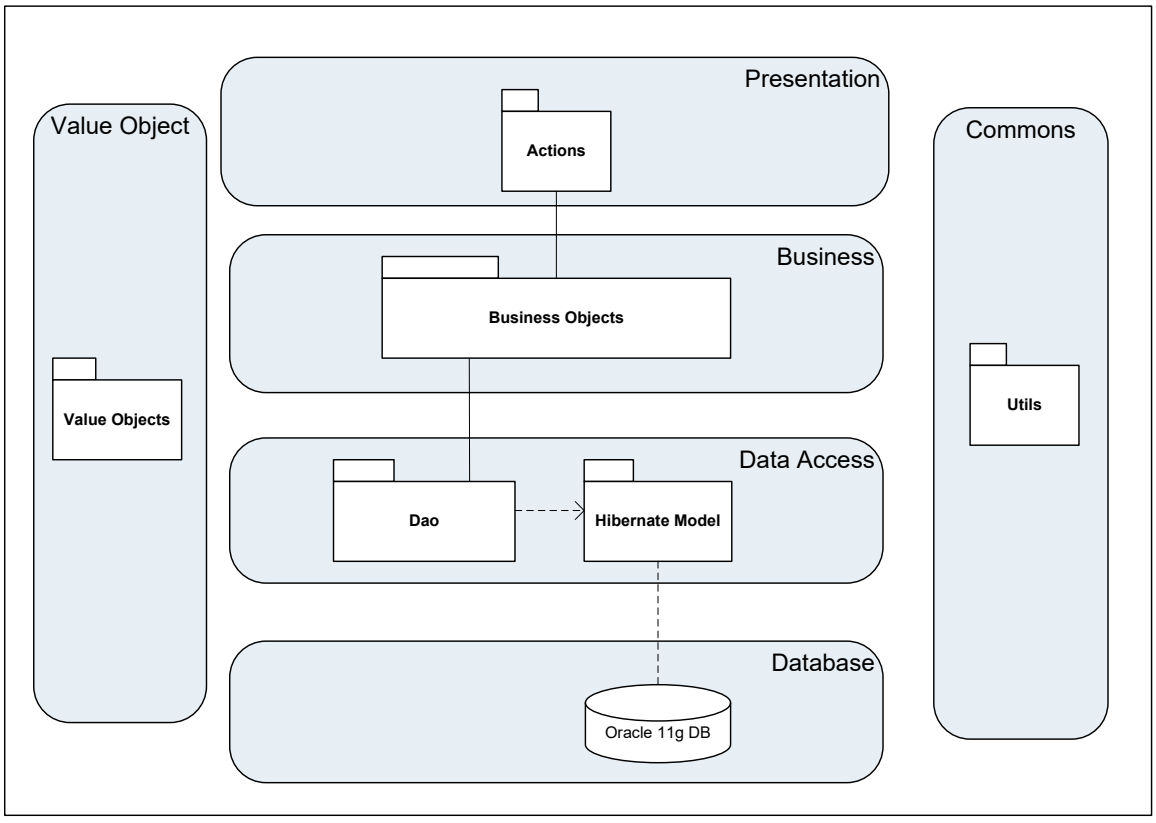

Figure 3 – JAVA01 Architecture packages.

2.3.1 JAVA01 presentation (JSPs, Actions)

Package name: `sg.com.customer.JAVA01.webapp.action`

This package includes the implementation for the Presentation Layer to handle the display to the end user.

2.3.2 Business Object

Package name: sg.com.customer.JAVA01.service

This package includes the implementation of business objects. **Business Object** (BO) layer is used to perform the business operations. The Business Object layer will access the DAO by implementing the Dependency Injector pattern. Transactions should be managed within this business layer.

- **POJOs:** All business objects will be POJOs and created using Spring.
- **Spring Transaction demarcation:** Use Spring light weight declarative transaction management where system wide transaction settings can be declared. i.e. for all business object methods which begin with 'insert' start a transaction but for all business object methods which begin with 'find' do not start a transaction. This is very flexible and will be applied using Hibernate.

2.3.3 Value Objects

Package name: `sg.com.customer.JAVA01.valueobjects`

Value object is Java class, contains lightweight structures for related business information. These are sometimes referred to as data transfer objects. A value object (VO) is a lightweight, serializable object that structures groups of data items into a single logical construct. (Value objects always implement java.io.Serializable).

- A VO is intended to minimize network traffic between enterprise beans and their callers (because each argument passed initiates a network transmission).
- A VO is designed to improve the performance of enterprise beans by minimizing the number of method arguments, and thus network transmissions, needed to call them.
- In addition, VOs are useful in communication among all layers of the application.

2.3.4 DAO

Package name: `sg.com.customer.JAVA01.dao`

This package includes the implementation of DAO Java pattern. Using DAO design pattern here to make the application more flexible to access database. DAO includes basic functions to work with database: *select, insert, update, delete*.

2.3.5 Hibernate Beans

Package name: `sg.com.customer.JAVA01.model`

This package includes the implementation for Hibernate object – which are mapping from relational database to java objects.

Hibernate provides a rich set of features to rapidly build high-performance enterprise applications that are both scalable and maintainable.

2.3.6 Util

Package name: sg.com.customer.JAVA01.util

This package includes all utilities Java classes will be wisely used in the modules.

APPENDIX A: List of Design Patterns and Frameworks Used

This section describes in detail the design patterns and frameworks used in the proposed application architecture for DWA.

APPENDIX A.1: *Struts 2 framework*

Several problems can arise when applications contain a mixture of data access code, business logic code, and presentation code. Such applications are difficult to maintain, because interdependencies between all of the components cause strong ripple effects whenever a change is made anywhere. High coupling makes classes difficult or impossible to reuse because they depend on so many other classes. Adding new data views often requires re-implementing or cutting and pasting business logic code, which then requires maintenance in multiple places. Data access code suffers from the same problem, being cut and pasted among business logic methods. The MVC design pattern solves these problems (Detail about MVC pattern please see http://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller).

The Struts 2 frame work (as shown in below Figure 4) is developed with goal is to cleanly separate the model (application logic that interacts with a database) from the view (HTML pages presented to the client) and the controller (instance that passes information between view and model). Struts provides the controller (a servlet known as ActionServlet) and facilitates the writing of templates for the view or presentation layer (typically in JSP, but XML/XSLT and Velocity are also supported). The web application programmer is responsible for writing the model code, and for creating a central configuration file struts-config.xml that binds together model, view and controller.

Struts 2 is a very elegant and flexible front controller framework based on many standard technologies like Java Filters, Java Beans, ResourceBundles, XML etc.

For the Model, the framework can use any data access technologies like JDBC, EJB, Hibernate etc and for the View, the framework can be integrated with JSP, JTL, JSF, Jakarta Velocity Engine, Templates, PDF, XSLT etc.

For the View, the framework works well with JavaServer Pages, including JSTL and JSF, as well as FreeMarker or Velocity Templates, PDF, XSLT, and other presentation systems.

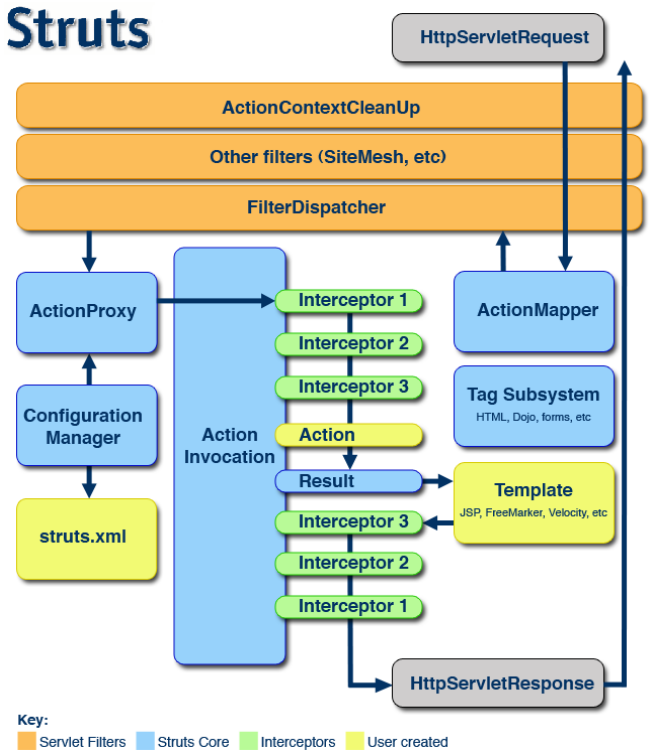

Figure 4: Struts 2 Architecture

APPENDIX A.2: *SPRING Framework*

Inversion of control and dependency injection

The concept behind Inversion of Control (IoC) is often expressed in the Hollywood Principle: "Don't call me, I'll call you.". IoC moves the responsibility for making things happen into the framework, and away from application code.

Dependency Injection (DI) is a form of IoC that removes explicit dependence on container APIs; ordinary Java methods are used to inject dependencies such as collaborating objects or configuration values into application object instances. Where configuration is concerned this means that while in traditional container architectures such as EJB, a component might call the container to say "where's object X, which I need to do my work", with Dependency Injection the container figures out that the component needs an X object, and provides it to it at runtime. The container does this figuring out based on method signatures (usually JavaBean properties or constructors) and, possibly, configuration data such as XML.

Spring framework (http://www.springframework.org/)

Spring is a light weight framework to support the IoC dependency injection concept. The Spring framework was released in March 24, 2004 and is now on version 1.2. It is

becoming a widely adopted framework and has many books, articles and coding examples online.

Spring's main aim is to make J2EE easier to use and promote good programming practice. It does this by enabling a POJO (plain old java object)-based programming model that is applicable in a wide range of environments.
When working with Spring, an application developer can use a large variety of open source tools, without needing to write reams of code and without coupling his application too closely to any particular tool. Spring basically provides the glue between the majority of different java technologies such as Struts, Spring own MVC, Session beans, Entity beans, Toplink, Hibernate, JDBC and the list is end less. If an integration does not exist then we can write our own anyway as spring is light weight and easy to develop with. This provides as loose possible coupling between technologies allowing for a true plug and play web architecture.

N.B. The spring framework is so successful that it has been applied to the .NET framework (http://www.springframework.net/)

Spring Advantages

1. **Reduce glue code:** One of the biggest plus points of dependency injection is its ability to reduce dramatically the amount of code we have to write to glue the different components of our applications together. Often this code is trivial—where creating a dependency involves simply creating a new instance of an object. However, the glue code can get quite complex when you need to access remote resources in a more **component based architecture**. In these cases, DI can really simplify the glue code by providing automatic lookup and automatic proxy of remote resources
2. **Externalize dependencies:** You can externalize the configuration of dependencies, which allows you to reconfigure easily without needing to recompile your application. This gives us two interesting benefits. Spring gives us the ideal mechanism for externalizing all the configuration options of our application for free. Second, this externalization of dependencies makes it much simpler to swap one implementation of a dependency for another. Consider the case where we have a DAO component that performs data operations using JDBC as in the case of MQS Phase 2. Using DI, you can simply reconfigure the appropriate dependency on your business objects to use Hibernate implementation rather than the JDBC one.
3. **Improve testability:** When we design our classes for DI, we make it possible to replace dependencies easily. This comes in especially handy when we are testing our applications. Consider a business object that performs some complex processing; for part

of this, it uses a DAO object to access data stored in a relational database. For our test, we are not interested in testing the DAO; we simply want to test the business object with various sets of data. In a traditional approach, where the business object is responsible for obtaining an instance of the DAO itself, we have a hard time testing this, because we are unable to replace the DAO implementation easily with a mock implementation that returns your test data sets. Instead, we need to make sure that our test database contains the correct data and uses the full DAO implementation for our tests. Using DI, you can create a mock implementation of the DAO object that returns the test data sets and then we can pass this to our business object for testing. This mechanism can be extended for testing any tier of our applications and is especially useful for testing web components where we can create mock implementations of HttpServletRequest and HttpServletResponse.

4. **Foster good application design:** Designing for DI means, in general, designing against interfaces. A typical injection-oriented application is designed so that all major components are defined as interfaces, and then concrete implementations of these interfaces are created and hooked together using the DI container. This kind of design was possible in Java before the advent of DI and DI-based containers such as Spring, but by using Spring, we get a whole host of DI features for free, and we are able to concentrate on building our application logic, not a framework to support it.

APPENDIX A.3: ***HIBERNATE***

Hibernate is well known and supported by SPRING framework, with the following key features:

Hibernate is a powerful, high performance object/relational persistence and query service. Hibernate lets developers to develop persistent classes following object-oriented idiom - including association, inheritance, polymorphism, composition, and collections. Hibernate allows developers to express queries in its own portable SQL extension (HQL), as well as in native SQL, or with an object-oriented Criteria and Example API

Hibernate Core features:

- **Natural programming model** - Hibernate supports natural OO idiom; inheritance, polymorphism, composition and the Java collections framework
- **Support for fine-grained object models** - a rich variety of mappings for collections and dependent objects
- **No build-time byte code enhancement** - there's no extra code generation or byte code processing steps in your build procedure

- **Extreme scalability** - Hibernate is extremely performant, has a dual-layer cache architecture, and may be used in a cluster
- **The query options** - Hibernate addresses both sides of the problem; not only how to get objects into the database, but also how to get them out again
- **Support for "conversations"** - Hibernate supports both long-lived persistence contexts, detach/reattach of objects, and takes care of optimistic locking automatically
- **Free/open source** - Hibernate is licensed under the LGPL (Lesser GNU Public License)