



J2EE/JSF Advantages & A Periscope Case Study

Periscope White Paper
February 2007

Table Of Contents

Introduction: Java Server Faces.....	3
Architecture Overview	3
Technology Advantages of JSF	4
J2EE/JSF At Work - Implementation Case Study at Periscope.....	6
Brief Synopsis:	6
Technology Framework / Components That Were Used:	6
Multi-Tiered Architecture:.....	6
Significant Design Decisions:.....	8
Page Decoration Using Tiles:	8
Security:	9
Multi-Level Menu:	9
Transaction Concurrency:	9
Page Navigation:	9
Performance Consideration:.....	10
Pagination:	11
File Upload & Retrieve:.....	11
User Connection Management from the Application:	11
Custom Validation & Conversion:.....	12
Error Message Customization:.....	12
Browser Session Management:.....	12
Data Loader Utility:	12
Internationalization and Localization:	12

Introduction: Java Server Faces

Java Server Faces (JSF) is a Java based Web application framework that simplifies the development of user interfaces for enterprise Java applications. Inside, JSF uses JavaServer Pages for its display technology.

JSF provides Web application lifecycle management through a controller servlet; and like Swing, JSF provides a rich component model complete with event handling and component rendering. It is based on other Java standards such as **Java Servlets** and **JavaServer Pages**, but it provides a higher-level component layer for UI (user interface) development.

Java Server Faces draws ideas from the many web development frameworks that grew during the past couple of years and provides a natural counterpart to desktop UI toolkits such as a Swing, but for web development.

Architecture Overview

The following diagram shows comparative features of various J2EE frameworks.

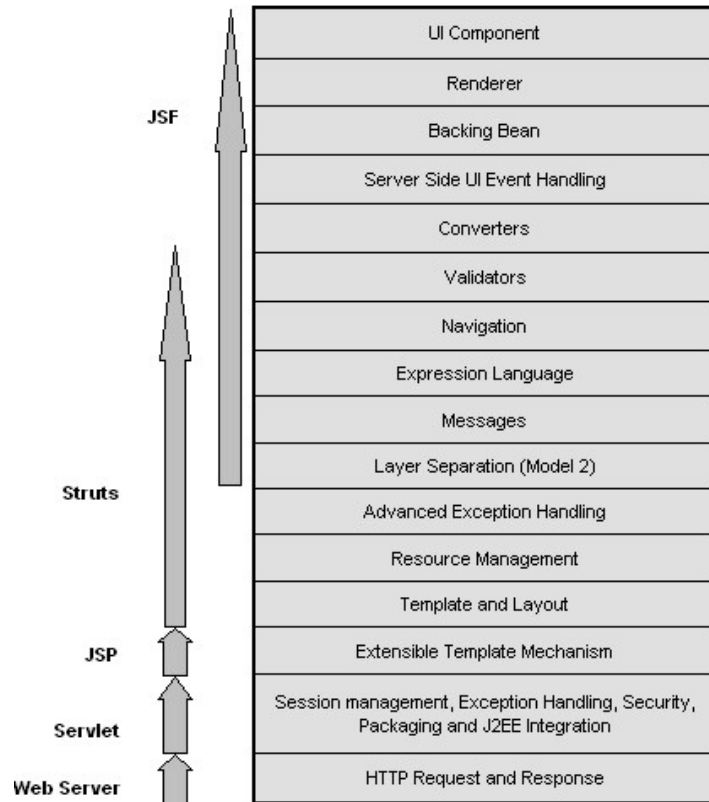


Figure 1: Comparative features of different features of J2EE frameworks

UI Component: Manages the interaction with the end user.

Renderers: Separate classes responsible for displaying UI components. Organized into *render kits* (HTML 4.0.1, WML, etc.). Render kits allow for “skinning” and can be changed on the fly.

Validators: Ensure a component’s values are correct.

Backing beans: Can be associated with UI components *declaratively* via the JSF expression language and Handles UI events.

Converters: Translate an object to and from a string for display.

Events and listeners: Capture the way the user interacts with UI components.

Messages: JSF maintains a list of the current messages generated from its own validation and from user generated message and throws it in the page.

Navigation: JSF uses declarative navigation (like Struts). The Navigation rule is what outcomes are understood and what pages to load based on those outcomes.

Expression language: Associates UI components with backing beans or model objects based on the EL included in JSP 2.0.

The following picture shows the Object Model of JavaServer Faces:

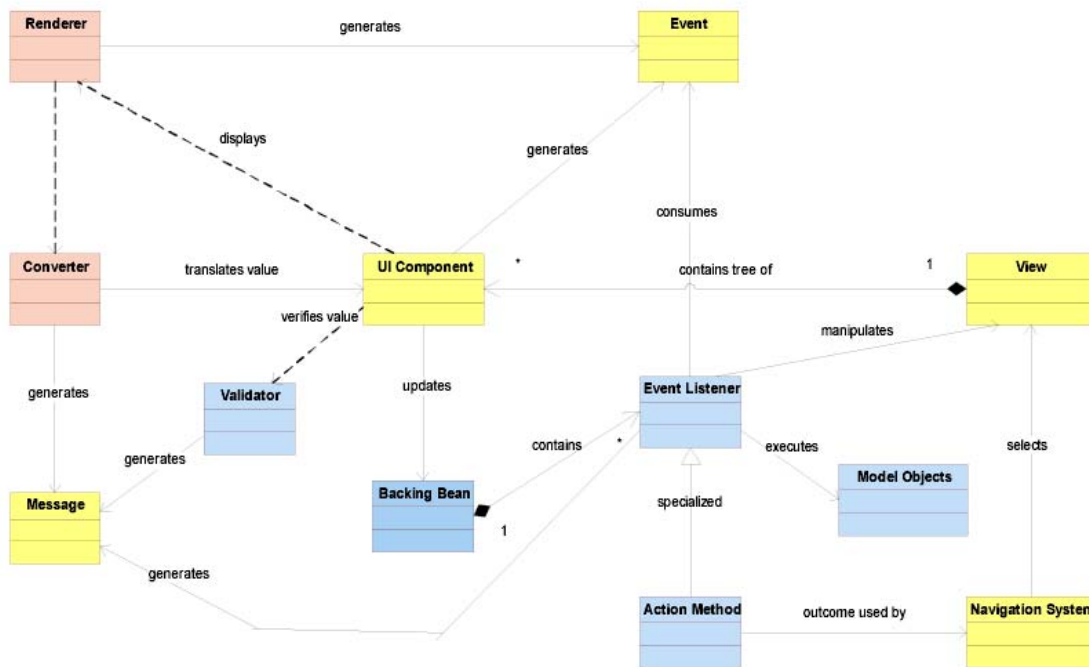


Figure 2: JSF Object Model

Technology Advantages of JSF

There are currently two popular techniques for developing web applications.

- The **rapid development** style, using a visual development environment such as Microsoft ASP.NET.
- The **hard-core coding** style, writing a number of lines of code to support a high-performance back end such as J2EE (the Java 2 Enterprise Edition).

J2EE is an attractive platform, which is highly scalable and portable in multiple platforms. Many vendors support J2EE. The promise of JavaServer Faces is to bring rapid user-interface development to server-side Java. JSF can be considered as "Swing for server-side applications"

The major benefits of JavaServer Faces technology are:

- JavaServer Faces architecture makes it easy for the developers to use. In JavaServer Faces technology, user interfaces can be created easily with its built-in UI component library, which handles most of the complexities of user interface management.
- JavaServer Faces technology offers a clean separation between behavior and presentation.
- JavaServer Faces technology provides a rich architecture for managing component state, processing component data, validating user input, and handling events.
- Robust event handling mechanism.
- Render kit support for different clients
- Highly 'pluggable' - components, view handler, etc

J2EE/JSF At Work - Implementation Case Study at Periscope

Periscope's Java Outsourcing Team was responsible for end-to-end technology implementation for a 'system modernization' effort. The following describes extensive use of JSF and other open source technologies when a legacy application (written in GUPTA) was transformed to J2EE web application.

Brief Synopsis:

Existing GUPTA application source code was rewritten in Java; front-end JSP pages were designed; hibernate mappings were created; required POJOs (Plain Old Java Object) were created and so forth.

One of the major requirements was to maintain "identical" look & feel between the source (desktop application) and target platforms (web based application). This included complexities such as: keeping multi-layer menu items, tracking online users, file upload/download functionality for each user, etc.

Source platform included: GUPTA front-end, SQLBase database, GUPTA business logic layer. Target platform includes: JSP/JSF front-end, Apache MyFaces framework, Apache Tomcat web/application server, Hibernate for relational mapping, Oracle database.

Technology Framework / Components That Were Used:

Frameworks	JDK 1.5 JSF / Apache MyFaces 1.1 Log4j (for detailed and efficient log management) Tiles (for content-formatting)
ORM (Object Relational Mapping)	Hibernate 3.0 (for Object Relational Mapping)
Web/App Server/JSP Container	Apache HTTP Server 2.0, Apache Tomcat 5.5
Database	Oracle 10G Database Server
Reporting Engine	Jasper Reports
IDE (Integrated Development Environment)	Eclipse 3.2

Multi-Tiered Architecture:

A multi-tiered architecture is designed to partition the system into distinct functional units: Client, Presentation, Business-Logic, Integration, and Enterprise Information System (EIS) / Database Management System.

The client tier is where the data model is consumed and presented. Internet Explorer 6.0 and Mozilla Firefox 1.5 browsers were used for development and testing.

Presentation Layer presents Business Rules to the client. Presentation Layer takes user input, validates them and supplies them to Business Logic Layer. It also controls the page navigation and maintains Application's session state. For the Presentation Layer, several tag libraries (Tomahawk, Sandbox, Extension and Core JSF) of JavaServer Faces / MyFaces were used.

Business Logic Layer processes the requests form the presentation layer and interacts with the integration layer. For this layer we used custom Business Agent written in Java

The integration tier handles the data persistence with the relational database Oracle. Different approaches can be used to implement the integration tier:

- Pure JDBC (Java Database Connectivity)
- Entity Beans
- ORM (Object Relational Mapping)

Hibernate 3.0 is used for integration and data persistence in this implementation.

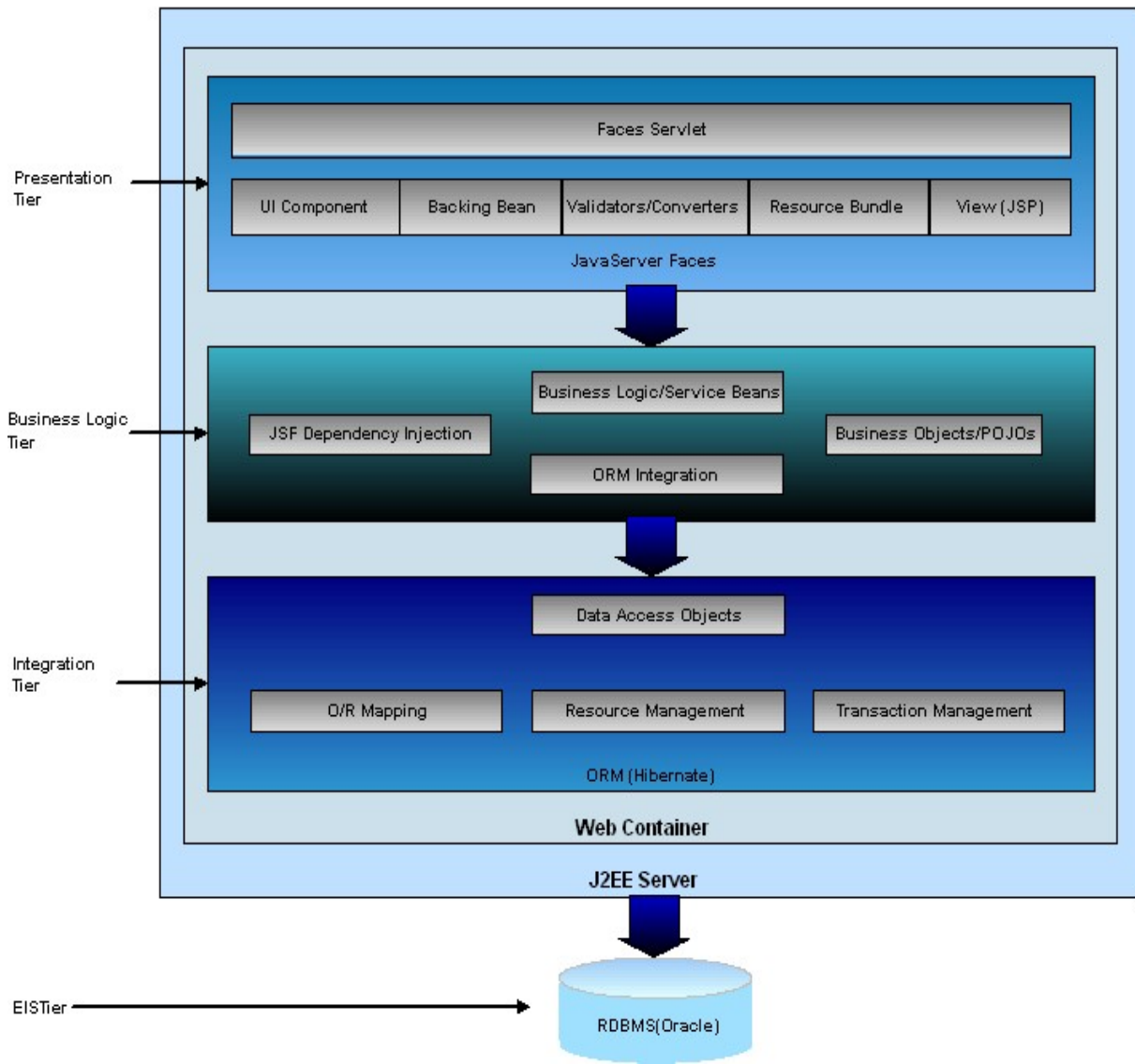


Figure 3: Architecture Diagram of the application based on JSF, Hibernate and Oracle.

Significant Design Decisions:

Page Decoration Using Tiles:

“Tiles” is a templating system. It can be used to create a common look and feel for a web application. Tiles can also be used to create reusable view components. We create a screen by assembling pages (header, footer, menu, body, etc) in our applications. Tiles can be plugged into different frameworks like Struts, JSF etc. Tiles is a combination of the following:

- Screen definitions
- Layouts
- Dynamic page building
- Reuse of Tiles / Components

- Internationalization (i18n)
- Multi-channels

[Read](#) more about Tiles.

Security:

Servlet Filter was used for authentication and authorization throughout the application. In addition to the basic Servlet filter, menu-level security, page-level security, action-level securities were implemented to ensure proper access rights of users depending on their role, etc.

Multi-Level Menu:

Menu is a very important part of any application. We prefer highly customizable menu in our applications. These menus have a desktop look & feel (can be enabled/disabled, grayed out, etc. like in desktop applications) with proper handling of common web menu problems like z-order problem, flickering etc. Also, multiple menus can be rendered in the same page depending on the necessity of the application.

Transaction Concurrency:

Concurrent updates of the same record (example: employee John Doe) by multiple users are handled by implementing **Optimistic Concurrency Control** feature of hibernate with versioning. Adding version column to the required tables and a version tag in the corresponding *.hbm.xml files. The version tag tells Hibernate to use the version filed of the tables for concurrency control. So, if two users load same object, a user updates the object and just after that, another user tries to save same object, the system reloads the object state saved by the first user and prompt second user to enter his/her data again. This is very important for the consistency of the data.

Page Navigation:

Page navigation is important for users to go to different pages depending on the application decisions. We use JSF built in page navigation techniques for this. Here we have to setup navigation rules in faces-config.xml. Inside <navigation-rule> node, there are some child nodes. This is a short summary about these nodes.

```
issuerBinding.setValue(ctx, issuer);
System.out.println("Load issuer");
return "loadIssuer";
}
```

Fig: a user action that returns a <from-outcome> value

Navigation rule: what outcomes are understood and what pages to load based on those outcomes. It generally contains

<from-view-id> which is the page the user is currently working on.

<navigation-case> which makes the decision – what action should be taken. It has two child nodes:

<from-outcome> which is the return value of the user action (method).

<to-view-id> what action should be taken or which page should be called.

```
<navigation-rule>
  <from-view-id>/pages/issuerLookUpForTickler.jsp</from-view-id>
  <navigation-case>
    <from-outcome>loadIssuer</from-outcome>
    <to-view-id>/pages/issuerTic_load.jsf</to-view-id>
  </navigation-case>
  <navigation-case>
    <from-outcome>promptTicAdd</from-outcome>
    <to-view-id>/pages/issuerAccBodyPop.jsf</to-view-id>
  </navigation-case>
  <navigation-case>
    <from-outcome>authFail</from-outcome>
    <to-view-id>/pages/AuthFail.jsf</to-view-id>
  </navigation-case>
</navigation-rule>
```

The JSF-way of handling page navigation is very intelligent and that's why we recommend using JSF navigation rule for page navigation.

Performance Consideration:

The performance of the application can be improved by many techniques. Some of what we follow is below.

Singleton Pattern:

The singleton is a useful design pattern for allowing only one instance of your class. The singleton's purpose is to control object creation, limiting the number to one but allowing the flexibility to create more objects if the situation changes. Since there is only one singleton instance, any instance fields of a singleton will occur only once per class, just like static fields. We use this design pattern wherever we need to reduce memory wastage by creating multiple instances of classes.

Caching:

Generally speaking, anything you can do to minimize traffic between a database and an application server is probably a good thing. In theory, an application ought to be able to maintain a cache containing data already loaded from the database, and only hit the database when information has to be updated. When the database is hit, the changes may invalidate the cache.

We use an open source caching system namely EHCACHE in our web application:

EHCACHE

EHCACHE is one of the most widely use Java caches. See <http://ehcache.sf.net>. It is fast, simple and has minimal dependencies. It comes with memory and disk stores. The disk stores are optionally persistent.

We use it for Hibernate, web page caching and search engine caching in our web applications. It is distributed with Hibernate, so if you are using Hibernate you need nothing more for caching.

Pagination:

Pagination is also very important for performance issue in web application. Displaying a fixed amount of objects at a time instead of all objects user requests for, can improve application performance greatly. We use custom pagination package in our applications. See details below.

Pagination:

The core JSF data grid component (h:dataGrid) automatically does the pagination. This is well with a small result set. But, we have some potential queries in some pages that can return large set data; this data set gets pulled into HttpSession and consumes too much memory. To overcome this issue we use Hibernate pagination instead of JSF pagination.

1 M	02/01/2006	Fees
1 M	02/01/2006	Financial Sta
1 M	02/01/2006	Incumbency

<< < 1/23 > >>

Fig: A screenshot of pagination

File Upload & Retrieve:

Manages the file upload and retrieve activity using the org.apache.myfaces.custom.fileupload.UploadedFile package. And uses the <x:inputFileUpload> tag in the JSP page. User can upload their desired files in the user folders and can retrieve it later when required.

User Connection Management from the Application:

We created a custom tool to manage user connection for a security reason. It creates a list of the users who are connected and if a user logout or if a session expires then the application removes that user from that list. A privileged user can destroy an active user session from the list. Below is the code snippet for User Connection Management:

```
// after successful login, putting userId in loggedInUsers
    Map<String, Long> loggedInUsers = (Map<String, Long>)
httpSession.getServletContext().getAttribute("loggedInUsers");
    if (loggedInUsers == null)
    {
        loggedInUsers = new HashMap<String, Long>();

        httpSession.getServletContext().setAttribute("loggedInUsers",
loggedInUsers);
    }
    loggedInUsers.put(user.getUserId(),
System.currentTimeMillis());

// removing userId from forceLogoutUsers, if exists
```

```
Set<String> forceLogoutUsers = (Set<String>)
httpSession.getServletContext().getAttribute("forceLogoutUsers");
    if (forceLogoutUsers != null)
    {
        forceLogoutUsers.remove(user.getUserId());
    }
```

Custom Validation & Conversion:

To meet our specific business needs we have written custom JSF validation rules and converters. We use custom validation and converters to validate some IDs in string format, date type objects.

Error Message Customization:

Customized JSF validation messages are built to make the messages more meaningful to the user.

Browser Session Management:

In most of our applications we store a user's logon information both in Application Server and in Database. So, User's login information is synchronized in both application server and in database. When a user logs into the application, the system checks his/her user ID in an Application Variable (List type) and a column of an authentication table is updated with "Y" flag, on logout of that user, the application removes him from the Application variable and that column gets updated with "N" flag. This mechanism is used to prevent multiple login with same user ID. We also manage proper logout activity on session timeout. But the logout activity is challenging on closing the browser by clicking (X) button of the browser and from taskbar by right clicking and opting the close option. The option of remembering a user from his machine so that, user don't need to login every time s/he enters into the application was also implemented in some applications. We use Cookies for this.

Data Loader Utility:

We have introduced data loader utility that reads valid CSV file and write the records in the database after validating the records from the CSV and it also read records from the database and export them in a CSV file in most of our applications. As an example, this tool reads account information from a CSV file then it checks for valid account number, account holder's ID, if all the validations return positive result then it inserts that record in the Account table and creates a log with the account numbers which are not eligible.

Internationalization and Localization:

Many web sites need to be adapted to a local or global audience. This can be accomplished by internationalization and localization of the web site. Internationalization consists of technically preparing a web site to display language and country specific (localized) content; localization consists of managing and creating localized data and resources, such as translation strings and formats. Technically, an internationalized web site is not usable unless it is localized to at least one language and country locale.

In summary, both processes consist of:

- Specifying locale settings, including formats and output encoding.
- Placing translation resource keys on pages.
- Creating translated texts.
- Specifying how users will select their language when viewing the web site

In our web applications Internationalization and Localization feature is included. So, it is possible to display the contents according to client's preferred locale(s).

For more information, feedback, and inquiries, please contact info@periscope-inc.com

Also, visit www.periscope-inc.com for our experiences in other technologies and platforms