



## **Network Activity D – Developing and Maintaining Databases**

Report D3.1.1

### **User interface : analysis and modularization study**

Anna Gadré  
Régine Vignes Lebbe

Université Pierre et Marie Curie, Paris

August 20th - 2005

The development of biodiversity databases, their interconnection and improved access through Internet will open collection data as well as taxonomic data for a wider public. In future, non-taxonomic users will become more and more important so that the User Interface will have to be adopted (UI) for these new requirements.

In order to build upon progress of BioCASE UI (Biological Collection Access Service for Europe) implementations it was necessary to obtain User feed back on the facilities available so far. Starting from this feedback and the experience gained during the BioCASE project we display in this document an analyse of different types of UI improvements and technical solutions to obtain a new modular and adaptable UI.

### **Plan**

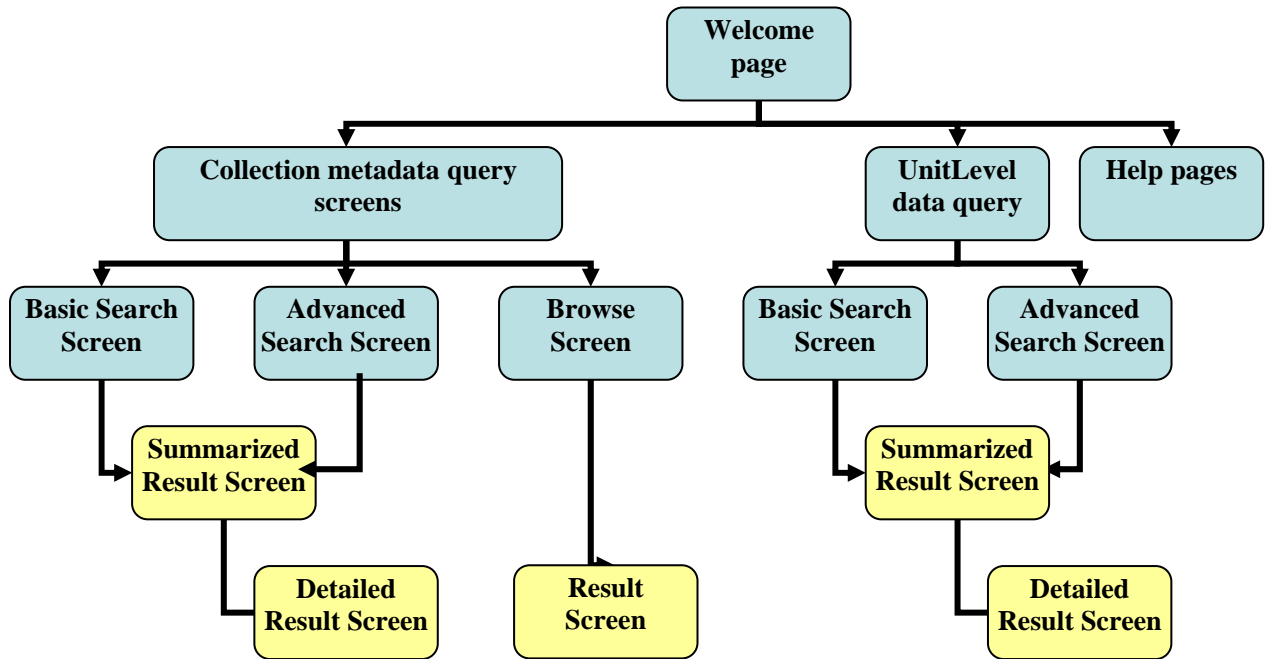
1	BioCase User interface .....	3
2	Modification of UI during the BioCASE project development .....	5
3	Customizing the user interface .....	5
4	Internationalization issues .....	8
5	Overview of technologies.....	10

# 1 BioCase User interface

BioCASE UI gives an access to two different kinds of data:

- Collection metadata providing from CORM database
- Unitlevel data providing from providers datababases

The general sitemap can be represented at follow:



Users can switch explicitly to different query screens with the navigation menu.

There are two types of query screens:

- typical form screens using standard HTML controls as text field or select list (Basic Search Screen and Advanced Search Screen)
- screens permitting data exploration without data input into text fields

Query results can be explored in two steps: first, a summary list is displayed, the Users can select from this list items and access detailed information describing these data.

The BioCASE UI is developed in Java which means

- crossplatform support
- largely server side processing so that additional software is not required on the client side.
- many libraries available (for example useful for XML processing or indexing)

Some UI screens are JSP files; others are directly generated by Java servlets.

Java beans associated to JSP files are classified into 2 groups:

- common beans

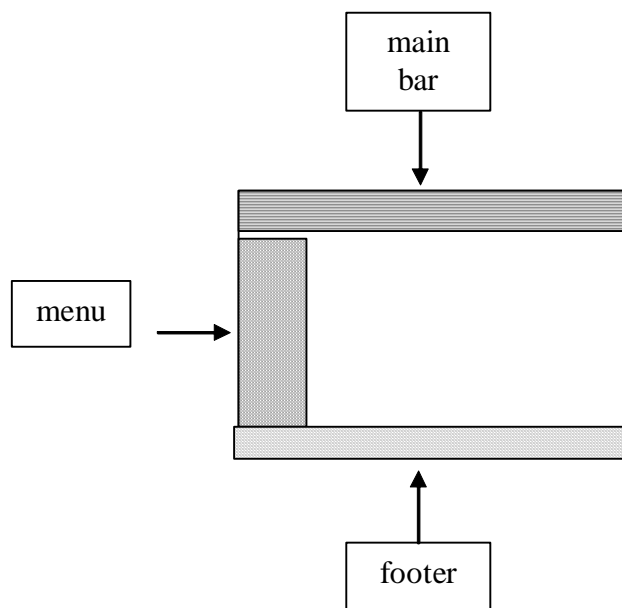
This group contains some packages used by all screens, for example the beans used for JDBC connection, an access to the index, the general site navigation as well as xml processin.

- specific beans: collections and specimens beans

This group contains the beans used respectively by Collection Metadata, or unit level data screens. In details, each group is sub divided in input, output, and servlet packages.

JSP also includes mechanisms which reduce code redundancy and promote reusability. To include an HTML or JSP fragment in a JSP file, the `include` directive is used. The effect of this directive is equivalent to the replacement of the directive with the contents of the included file and an insertion of the text contained in another file: either static content or another JSP page, in the including JSP page. This means that an included fragment can access any objects defined in the main JSP page. The `jsp:include` element is processed when a JSP page is executed (translated into a servlet class).

The include mechanism is used for situations when text, such as ASCII or HTML, needs to be included in multiple JSPs especially for banner content or copyright information re-used in the entire website. The figure below shows some elements of the BioCASE UI inserted with include mechanism:



UI installation includes a set of setting files used for specify some parameters like database connection properties, files path and so on. All SQL queries are also extracted into a special file.

The present version of the UI defines the style of pages and their components in a CSS file. Cascading Style Sheets (CSS) technology helps to separate content and structure from presentation and layout. The documents created with CSS load faster and are easier to maintain and update. Their content can be changed independently of formatting because presentation and layout are handled in CSS. This means that it is possible to change presentation and layout on many pages at once by changing the external style sheet to which those pages are linked without having to modify the source document. For example, if we decide to propose different screen presentations for different users groups, the CSS technology will permit very quickly integrate this modification.

## 2 UI modifications during the BioCASE project

During several BioCASE workshops different versions of BioCASE UI were presented and commented. The screens were reviewed according to the criticism and comments. These modifications can be classified in two categories: ergonomics and functionality.

### 1) UI ergonomics and ease-of-use

- screen organization

We received many comments about the position of menus, use of tabsheets, and button locations. Some screens were considered as too complex (metadata advanced query screen) but also some others screens were merged and become more complex by regrouping more functionalities (organization/network/collection sections screens became one common result screen)

- type of form elements

One example is a request for drop down lists instead of text fields to facilitate selections when entering query data.

- labels and textual messages

Some titles, command button messages, and field labels were considered as too generic or poorly written; and they were replaced by others, more clear, concise, and written in an understandable way. This usually means to use no technical jargon or system-oriented information and to provide full and clear descriptions.

- graphic presentation

Users asked for modifications such as contrast between the text color and the background color, the possibility to choose text size according to the User's preferred font size or font type

### 2) UI functionalities

- **modification of functionalities (example modification of format to export results)**
- **deletion of functionalities (example map ...)**
- **adding new functionalities (sort of results, time limit for query, ...)**

- adding/removing new functionalities

Some functionalities were considered as not useful for different reasons

- the character of the data doesn't allow to use correctly this functionality

(ex-selection of location or display of the result with use of GIS parameters)

- users proposed a solution better adapted for their needs (for example reports in text format instead of PDF file)

## 3 Customizing the User interface

### 3.1 *On-line feed back of the users*

BioCASE and BioCISE had listed User types for collection data. These different User categories were described and contacted to test the UI (see RGE WP). A feedback form has been linked to the BioCASE UI (see fig).



Feedback given by 21 Users has been compiled (August 18<sup>th</sup> 2005) by the RGBE team. These comments have to be analyzed in order to propose concrete improvement of UI.

### 3.2 Profiles of the users

The profiles of BioCASE UI Users can be very different and therefore the UI should be able to satisfy very different needs. In general, these differences can concern two domains:

- **different levels of Internet skills**, browser program software and computer techniques use.  
Some users; more experienced with internet navigation rules, will more easily and quickly fulfill their requirements.

These users will for example be interested to work with some special data formats like XML, not appropriate for not experienced users. They can also understand more technical vocabulary (e.g. concerning internet issues and database content).

- **different levels of biodiversity and taxonomy knowledge**

These Users, according to their profile needs a customization of UI concerning:

- different query criteria and displayed result
- different order of mentioned elements on the screen
- different vocabulary

### 3.3 *Multiple specific UI versus one UI*

Ideally, the UI design should be so intuitive that both the beginner and expert use the same interface. But in reality the two needs have to be balanced.

It is not exceedingly common to see expert/novice modes in user interfaces. By expert/novice mode we mean having (at least) two different levels of accessibility to the user, one tailored to the beginner, and another to a user who has familiarity with the system. Normally, though, expert/novice distinctions can be confusing for the user because the interface changes in some respects. Sometimes having beginner and expert levels never allows the beginner to transition. A beginner level is often a wizard which bears little resemblance to the expert level. Most of the time the interface for the wizard is completely separate from the expert level which makes the transition difficult. Strive for consistency of presentation and input between the different levels.

For these reasons specialist of UI development claimed that "a well-designed and humane interface doesn't have to be split into beginner and expert subsystems." (Raskin, Jef. *The Humane Interface*. (Stoughton, MA: ACM Press, 2000) Providing several methods to do a task will help a user transit from beginner to expert. Expert novice modes can be explicit settings for the user to choose, or they can evolve implicitly with the user over time. An explicit setting requires some way for the user to specify that they are ready for an expert mode.

So; according to UI development recommendation ; this vision of an expert/novice interface is actually a system with three or more levels of expertise and which changes very slightly as the user becomes more and more used to the system. This system, able to assist users; which simply begins to tell users about advanced functionality as they become more and more comfortable with basic commands. It is obvious that a program's role is to assist in the user's task toward achieving a goal. A program should be a guide or a tutor for users. Therefore, a program has to have the ability to help users. For example, a program should provide proper tools, demonstrate a user the proper way to navigate a program, indicate to users that there are functions in the program, and give a guide to performing the task and the flow of functions.

While a simpler user interface, based on keyword search, is enough for expert users, the user interface for non expert users must be significantly more complex, since it must guide them step-by-step from the general concepts to the detailed information in the database.

The criteria for developing a user interface for non-expert users were studied by a working group composed of computer engineers, experts of disability in education and experts of cognitive psychology. The group defined a set of requirements to guarantee an effective communication, where the most important is *circularity* of information: the user selects some items among the information presented by the machine, and the machine proposes new information based on the user selections, in an endless circular process. The navigation model is based on a gradual approach to the topic, starting from general information and going towards the documents. The user never has to type any text, and the interface proposes possible choices based on the current information available to the IUI. All specific terms are *contextualized*, i.e., never appear alone (as in a list of keywords) but are always parts of a complete sentence. This interface is aimed at users that have little or no knowledge of the specific terms and keywords used by experts in the field, and allows an easy navigation without requiring to type any query string. While the user navigates, the Interface infers a model of his/her interests, and activates suitable searches on behalf of the user.

## 4 Internationalization issues

Internationalization is the process of adapting application for use abroad. Internationalization requirements vary from one country to another, depending on language, formatting conventions, and also cultural values. It can concern:

- Support for national language and corresponding character set.
- Internationally acceptable use of images and colours. Some images may be interpreted differently depending on culture.
- Support for national format conventions for dates, time, decimals, currency, names, and addresses. Formats may include or omit certain fields, and require different separators.

In some cases, an application must be able to display different character sets to support languages such as:

- Japanese
- Chinese
- Arabic
- Russian (Cyrillic)
- Greek

Applications must also support bi-directional (BiDi) languages, such as Arabic and Hebrew, which read text right to left, but read numbers left to right (within the same body of information). This support is provided through style sheet settings, so it is essential not to hard code any element that must be translated into a BiDi language.

BiDi languages require that all horizontal directional UI images, such as arrows, be flipped to match the reading direction.

Presently, BioCASE UI has a very simple mechanism permitting to change the language of application. The text of the GUI (all text labels, buttons values, titles, menus labels, messages) is outputted from HTML /JSP code and separated in the special file.

This is a simple text file containing couples of values:

*Identifier of text to be displayed= Text displayed in specific language*

Actually UI uses English version of this file, the name of this file is indicated in system setting files. If we would like use another language for UI, this text file should be translated entirely in this specified language. It is possible to use a mechanism of browser recognition of linguistic environment of user, so the browser would be able send this information to the system. In this case, the application will be able to choose a text file designated for this language and automatically reload adapted UI version displayed in the current language.

There is no translation system for the dataset results extracted from CORM database or providers databases.

The current system should be improved. Indeed, internationalisation can involve many specific problems:

- Page layouts must allow translated text to expand by an average of 100 percent, and up to 150 percent for short words (less than 5 characters long) and narrow columns.
  - Words in some languages, such as German and Finnish, are typically longer than their English equivalents. When placed in narrow columns, these longer terms cause additional line breaks, leading to major increases in vertical space, and added horizontal space, which causes extra scrolling.
  - Some languages may not have equivalent terms to those used in the original English version, so those terms need to be translated as multiple words.
  - Sometimes references to national organizations, telephone numbers, and addresses must be modified, requiring different fields or a change in field order.

The layout and styles of UI should be also be revisited to take account the internationalization issues: sometimes they need be changed in translated screens:

- **Font Families:** Some hard-coded fonts should be not use anymore but use CSS style classes instead. The style sheet can then be customized by locale.
- **Font Sizes:** Certain font sizes are not available or legible in all languages. For instance, Asian fonts must be rendered in larger point sizes.
- **Alignment:** To ensure proper translation to all languages, including bi-directional languages, applications must use alignments respectively for textual content and other alignment type for strictly numeric content.
- **Horizontal and Vertical Space:** Avoid any precise layouts, for instance filling a screen with a dense amount of information that must line up correctly. Horizontal and vertical spacing can dramatically change when an application is translated to a different language, especially when that language is character based, and/or bi-directional.
- **Embedded Images and Fields:** Applications cannot embed images or fields for variables within sentences, or concatenate fields and labels in a sentence-like structure
  - Languages have different grammars which cause major rearrangements of translated sentences, thus making it impossible to predict where to place the images or fields in relation to the labels.
  - Some languages, especially Romance languages such as French and Spanish, have both masculine and feminine forms of articles, adjectives, and pronouns, and these parts of speech must agree with the gender of related nouns. If a field displays a word, it is impossible to predict its gender when translating associated label text.
- **Fields and Field Order:** Depending on locale settings or country selection, some fields in forms and tables may be added, omitted, or change order:
  - Name and address fields may change fields and field order to match national requirements.
  - Some languages, such as Japanese, include additional phonetic fields (a.k.a. "Alternate" fields as aids to pronunciation).

These changes should be implemented by individual development teams based on the localization needs of their customers.

Also; many format conventions vary from one country to another. Some of these, such as date, time, currency, and numeric formats, are controlled by user preference and locale

settings. Locale settings modify the display of data to match language, and national and regional requirements. For example, many European locale settings substitute commas and periods in numeric formats. User preferences include choices between a minus sign and angle brackets to display negative numbers, choices between 24-hour and 12-hour time formats, and comma or period for numeric formats.

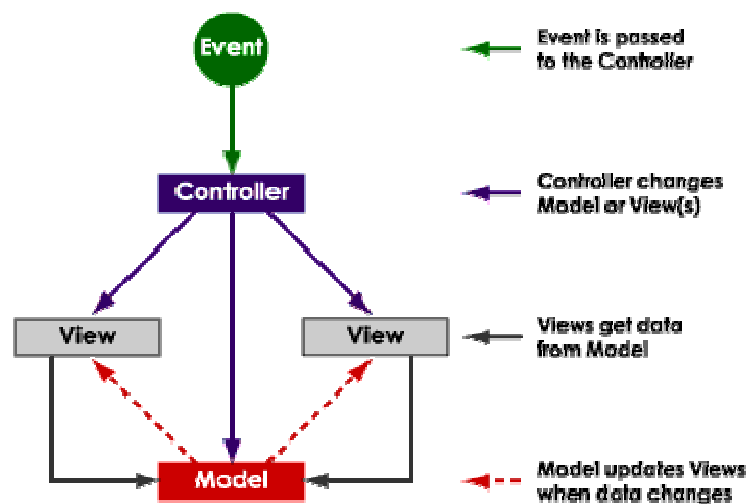
## 5 Overview of technologies

### 5.1 Model-View-Controller (MVC)

Model-View-Controller (MVC) is a classic design pattern often used by applications that need the ability to maintain multiple views of the same data. The MVC pattern hinges on a clean separation of objects into one of three categories - models for maintaining data, views for displaying all or a portion of the data, and controllers for handling events that affect the model or view(s).

Because of this separation, multiple views and controllers can interface with the same model. Even new types of views and controllers that never existed before can interface with a model without forcing a change in the model design.

The MVC abstraction can be graphically represented as follows.



**Fig. 1 MVC organizes an interactive application into three separate modules: one for the application model with its data representation and business logic, the second for views that provide data presentation.**

Events typically cause a controller to change a model, or view, or both. Whenever a controller changes a model's data or properties, all dependent views are automatically updated.

Similarly, whenever a controller changes a view, for example, by revealing areas that were previously hidden, the view gets data from the underlying model to refresh itself.

The literature on Web-tier technology in the J2EE platform frequently uses the terms "Model 1" and "Model 2". Model 1 and Model 2 simply refer to the absence or presence of a controller servlet that dispatches requests from the client tier and selects views.

A Model 1 (Page-centric architecture) consists of a Web browser directly accessing Web-tier JSP pages. The JSP pages access Web-tier JavaBeans that represent the application model, and the next view to display (JSP page, servlet, HTML page, and so on) is determined either by hyperlinks selected in the source document or by request parameters. A Model 1 application control is decentralized, because the current page being displayed determines the next page to display. In addition, each JSP page or servlet processes its own inputs

(parameters from GET or POST). In some Model 1 architectures, choosing the next page to display occurs in scriptlet code, but this usage is considered poor form.

A Model 2 (Servlet-Centric Architecture) architecture introduces a controller servlet between the browser and the JSP pages or servlet content being delivered. The controller centralizes the logic for dispatching requests to the next view based on the request URL, input parameters, and application state. The controller also handles view selection, which decouples JSP pages and servlets from one another. Model 2 applications are easier to maintain and extend, because views do not refer to each other directly. The Model 2 controller servlet provides a single point of control for security and logging, and often encapsulates incoming data into a form usable by the back-end MVC model. For these reasons, the Model 2 architecture is recommended for most interactive applications.

A Model 1 architecture is best when the page navigation is simple and fixed, and when a simple directory structure can represent the structure of the pages in the application. Such applications usually embed the page flow information in the links between the pages. The presence of `forward` in a JSP page implies that logic embedded in the page is making a decision about the next page to display.

JSP pages in a Model 1 application that use scripting elements, custom tags, or JavaScript to forward requests should be refactored to Model 2.

Most Web-tier application frameworks use some variation of the MVC design pattern and share common set of functionality

- Dispatching HTTP requests
- Invoking model methods
- Selecting and assembling views
- Provide classes and interfaces that can be used/extended by developers

In a more concrete terms, these are the benefits of using a common Web application frameworks. Framework decouples presentation tier from business logic, which would be useful for maintenance and reusability of the code. It provides a central point of control.

The features of frameworks are the following

- Decoupling of presentation tier and business logic into
- separate components
- Provides a central point of control
- Provides rich set of features
- Facilitates unit-testing and maintenance
- Availability of compatible tools
- Provides stability
- Enjoys community-supports
- Simplifies internationalization
- Simplifies input validation

## **5.2 Frameworks-Models MVC2**

There is currently a certain number of more or less advanced project aiming to the creation of framework MVC2. These projects, all based on the servlets, progress very quickly. Here is a short presentation:

### 5.2.1 Barracuda

The Barracuda Presentation framework (<http://barracuda.enhydra.org/>) is built as a series of layers that depend on the Servlet 2.2+ API, each of which can be used independently of one another.

The key features of Barracuda are summarized here:

- **Client Capabilities** - Automatically identifies client capabilities (browser type, target locale, etc).
- **Event Model** - Maps client requests to first class event objects; dispatches events to all interested listeners; guarantees that an HTTP Response is generated.
- **Form Mapping & Validation** - Converts HTTP Form parameters into first class Java objects and provides a powerful validation mechanism to verify them.
- **XMLC** - Compiles HTML, WML, and XML documents into DOM template objects that can be manipulated programmatically.
- **Localization** - Automatically creates localized versions of the DOM templates for additional locales. These templates can be loaded based on a target client locale.
- **Component Model** - Uses familiar UI Widgets with strongly typed Swing-style MVC interfaces to dynamically populate the DOM templates with data. the DOM can then be rendered and returned to the client browser.

### 5.2.2 Tapestry

Tapestry (<http://sourceforge.net/projects/tapestry>) is an open-source framework for creating dynamic, robust, highly scalable web applications in Java. Tapestry complements and builds upon the standard Java Servlet API, and so it works in any servlet container or application server (. released under the Apache Software Licence 2.0).

Tapestry divides a web application into a set of pages, each constructed from components.

This provides a consistent structure, allowing the Tapestry framework to assume responsibility for key concerns such as URL construction and dispatch, persistent state storage on the client or on the server, user input validation, localization/internationalization, and exception reporting. Developing Tapestry applications involves creating HTML templates using plain HTML, and combining the templates with small amounts of Java code using (optional) XML descriptor files. In Tapestry, an application is created in terms of objects, and the methods and properties of those objects -- and specifically *not* in terms of URLs and query parameters. Tapestry brings true object oriented development to Java web applications. Tapestry is specifically designed to make creating new components very easy, as this is a routine approach when building applications. The distribution includes over fifty components, ranging from simple output components all the way up to complex data grids and tree navigators.

Tapestry is architected to scale from tiny applications all the way up to massive applications consisting of hundreds of individual pages, developed by large, diverse teams.

### 5.2.3 Struts

Struts (<http://jakarta.apache.org/struts>) is a very popular web application framework. Struts consists of the following architectural pieces:

1. **Controller** - Struts provides a Controller servlet to give the developer a default Model 2 flow control implementation. This servlet acts as a kind of "central

nervous system" to ensure that requests gets routed, handled, and rendered according to this pattern.

2. **Model** - A common task in most applications is getting data from the HTTP Request into the Model layer (which acts as a facade to underlying business objects). Struts provides ActionForm and Action classes to streamline this process.

An Action class acts as an "adapter" between the web layer (HttpServletRequest) and the business logic (JavaBeans, EJBs, whatever). An ActionForm is really just a server-side representation of the current contents of an input form. Typically a developer writes Java code that extends these classes to provide the implementation details specific to a particular action.

This layer also provides a mechanism for developers to validate the data and redirect flow if an error occurs.

3. **View** - Of course, applications also need to generate responses to client requests. In Struts, this occurs in the View portion of the framework. Views are typically rendered with JSPs, which use Java Taglibs to pull data out of the Model and insert it into the JSP page. Once the page has been populated with data, it is ready to be returned to the client.
4. **Utilities** - Struts offers several utilities aimed at specific aspects of the development process- for example Database support - Struts allows to define JDBC data sources from a config file and provides a JDBC connection pooling mechanism

Struts makes part of the Apache project. Knowing that the quality of the projects Open Source is brought by the number of users, it's possible to think that Struts will be of better quality than other projects MVC2-Since some frameworks was abandoned, whereas Struts continues its development in an unquestionable way.

#### 5.2.4 Java Servlet Faces (JSF)

JavaServer Faces (JSF) is a user interface framework for building web applications that run on the server side and render the user interface back to the client. It lets develop tools that simplify coding web-based Java applications and the construction of the presentation layer of Web applications. JSR 127, which defines the JSF framework, comes with a reference implementation that provides basic UI components, such as input fields and buttons. It is possible assemble reusable UI components to create Web pages, bind these components to the application data source, and process client-side events with server-side event handlers.

JSF provides a clear separation between application logic and GUI presentation, improved maintenance capability for Web applications, and a framework for the development and reuse of Web UI components.

Users of JSF-based web applications appreciate the wide range of user actions made available by JSF controls. It offers more features than a standard HTML front end.

JSF provides the following main features:

- Clean separation of behaviour and presentation
- Page navigation specification
- Standard user interface components like input fields, buttons, and links
- User input validation
- Easy error handling
- Java bean management
- Event handling
- Internationalization support and Standardization
- Device Independence

A typical JSF application consists of the following parts:

- JavaBeans components for managing application state and behaviour
- Event-driven development (via listeners as in traditional GUI development)
- Pages that represent MVC-style views; pages reference view roots via the JSF component tree

JSP has the ability to be extended with custom tags. A custom tag is a special XML element backed by Java code that can be used in addition to standard JSP elements or HTML elements. A custom tag can do almost anything: display the value of variables, parse XML, conditionally display parts of a page, access a database, and so on. Their main purpose is to keep Java code out of the pages and allow frontend developers to use simple, familiar tags instead. A group of related custom tags forms a tag library. JSF is integrated with JSP using custom tags.

The navigation rule for this application is described in the **faces-config.xml** file. This file already exists in the skeleton directory structure. The JSF Navigation Framework provides navigation rules that allows to define navigation from view to view (mostly JSP pages) in a Web application.

JSF provides tools for internationalizing Web applications; it supports number, currency, time, and date formatting, as well as allowing to externalize UI strings.

JSF technology comes with predefined messages for all sorts of situations, namely validation and conversion. However, these predefined messages suffer from two severe problems:

- They are general and do not provide the user with enough information to correct the error. For example, the conversion error does not display the pattern that the user input should comply with.
- They are not localized, i.e. they are only available for English.

JSF uses the standard Java internationalization (I18N) mechanism, which means that we can provide other messages, and for other locales, than those already available

With the simple, well-defined programming model that JSF technology provides, developers of varying skill levels can quickly and easily build Web applications by: assembling reusable UI components in a page, connecting these components to an application data source, and wiring client-generated events to server-side event handlers. With the power of JavaServer Faces technology, these web applications handle all of the complexity of managing the user interface on the server, allowing the application developer to focus on their application code.

Sun and other members of the JSF expert group -- which includes Borland, IBM, Macromedia, and Oracle, along with many other companies and individuals -- are evaluating ways to incorporate JSF technology into a new generation of tools that simplify the development of multi tier web-based applications. One of these tools is Sun's Java Studio Creator.

### **5.2.5 Comparison JavaServer Faces technology and Struts**

Struts is an open-source Java web application framework whose architecture is based on the MVC design pattern in which requests are routed through a controller that provides overall application management and dispatches the requests to application components. JSF technology is a user-interface framework for Java web applications. It is focussed on the view tier of an MVC-based architecture. The Struts and JSF technology frameworks do have some overlapping functionality; however each framework has its advantages and developers can use certain features of both frameworks in a single application.

The primary advantages of Struts as compared to JSF are as follows:

- Struts has a more sophisticated controller architecture than does JSF. It is more sophisticated partly because the application developer can access the controller by creating an Action object that can integrate with the controller, whereas JSF technology does not allow to access to the controller. In addition, the Struts controller can do things like access control on each Action based on user roles. This functionality is not provided by JSF technology.
- Struts includes a powerful layout management framework, called Tiles, which allows to create templates reusable across multiple pages, thus enabling to establish an overall look-and-feel for an application.
- The Struts validation framework includes a larger set of standard validators, which automatically generate both server-side and client-side validation code based on a set of rules in a configuration file. It is also possible create custom validators and easily include them in application by adding definitions of them in configuration file.

The greatest advantage that JSF technology has over Struts is its flexible, extensible UI component model, which includes:

- A standard component API for specifying the state and behaviour of a wide range of components, including simple components, such as input fields, and more complex components, such as scrollable data tables. Developers can also create their own components based on these APIs, and many third parties have already done so and have made their component libraries publicly available.
- A separate rendering model that defines how to render the components in various ways. For example, a component used for selecting an item from a list can be rendered as a menu or a set of radio buttons.
- An event and listener model that defines how to handle events generated by activating a component, such as what to do when a user clicks a button.
- Conversion and validation models for converting and validating component data.

Because the JSF technology architecture separates the definition of a component from its rendering it is possible render all components in different ways or even to different clients, such as a WML client. Moreover, the extensible component APIs of JSF technology allows

extending the standard set of components and creating entirely new components. None of this is possible with Struts. In fact, Struts has no notion of server-side components, which also means that it has no event model for responding to component events and no facility for saving and restoring component state. While Struts does have a useful tag library for rendering components on the page, these components have no object representation on the server and they can only be rendered to an HTML client.

Another distinct advantage of JSF technology is that it is standard, which means that it has been developed through the JCP process and has been designed to allow easy integration into tools. As a result, JSF technology already has wide industry support and is being leveraged by several web application development IDEs.

Because both JSF Faces technology and Struts contribute such valuable features, developers might want to be able to use both of them in a single application. Developers might want to integrate the flexible component model of JSF Faces technology into their existing Struts applications while continuing to use the Struts controller architecture. Similarly, developers who have JSF Faces technology applications might want to integrate the more powerful client-side validation mechanism and Tiles layout framework found in the Struts architecture into their applications.

### **5.2.6 JavaServer Faces technology and JavaServer Pages (JSP) technology relation**

JSF technology, version 1.0 relies on JSP 1.2. Since JSP 2.0 is a superset of JSP 1.2, it is possible to use JSF technology, version 1.0 with JSP 2.0. Future versions of the JSF specification will be able to take better advantage of JSP 2.0. So, an integration of JSF in BioCASE UI architecture (developed in JSP) is not problematic. Some features of JSF, especially reusable components and internationalization solutions can bring many advantages. If we decide to use JSF, the maintenance of UI will be simplest, but for this maintenance we will need a developer who is familiar with this technology. For actual UI version, since most JSP pages are simply HTML documents, any Java programmer who is familiar with HTML should be able to maintain it without a problem. Otherwise, actually even as maintenance doesn't need any special computer knowledge, as the presentation and logic code coexist in JSP pages, and are coupled, this raises development and maintenance costs.