

# Modular Web Framework: Software Automation

Tuan Nguyen and Dr. Paul Wagner (Faculty Mentor)

Department of Computer Science  
University of Wisconsin - Eau Claire

Eau Claire, WI 54701

[nguyenta@uwec.edu](mailto:nguyenta@uwec.edu), [wagnerpj@uwec.edu](mailto:wagnerpj@uwec.edu)

## Abstract

The use of software is an integral part of any organization. Software applications, including web applications, must evolve quickly. By dividing a web application into independent components, we are developing a framework that gives developers a set of generic functionalities which can be integrated to fulfill specific requirements. Our research involves both developing the framework and comparing the similarities and differences with other well-known frameworks that support the development of a web-application.

Modular Web Framework (MWF) is a framework which also supports extension of its functionality for future requirements. MWF enables developers to automate three components of a web application: 1) a web-based interface, 2) a specification-based process, and 3) database manipulation.

# 1 Introduction

The use of software is an integral part of any organization. Software applications, including web applications, must evolve quickly. By dividing a web application into independent components, we are developing a framework that gives developers a set of generic functionalities which can be integrated to fulfill specific requirements. Our research involves both developing the framework and in the future, comparing the differences in time and costs in design and development between using our framework and other currently used methods.

Currently, there are several frameworks that support dynamic generation of interface, data structures and database manipulation. These frameworks include Java Server Faces, Spring, and Hibernate. However, there are still many issues in using these existing frameworks. Firstly is the integration between frameworks. Although all three frameworks are Java-based, each of them utilizes different technology and structures. So it is a challenge of to understand each one of the frameworks, use them efficiently, and configure them to work as one system. Secondly, none of the frameworks provides a very user-friendly interface. It is understood that each of the frameworks is provided for the programmers, and not to the end user. However, the lack of an interface increases the trial-and-errors period, and leaves space for undesirable mistakes.

Modular Web Framework (MWF) is a framework which also supports extension of its functionality for future requirements. MWF enables developers to automate three components of a web application: 1) a web-based interface, 2) a specification-based process, and 3) model-based database automation. MWF also provides a user-friendly interface for the users and programmers. Not only the interface helps the programmers to learn the framework faster and better, bringing understanding by examples, but also provide functionalities for users without requiring much of time, effort and knowledge of the system

# 2 Features

The framework consists of three components. The three components of the framework are correlated to other frameworks, so it is beneficial to present the features of each component in light of the other well-known frameworks. This helps in relating the similarities and also distinguishing and emphasizing the differences of each of the two.

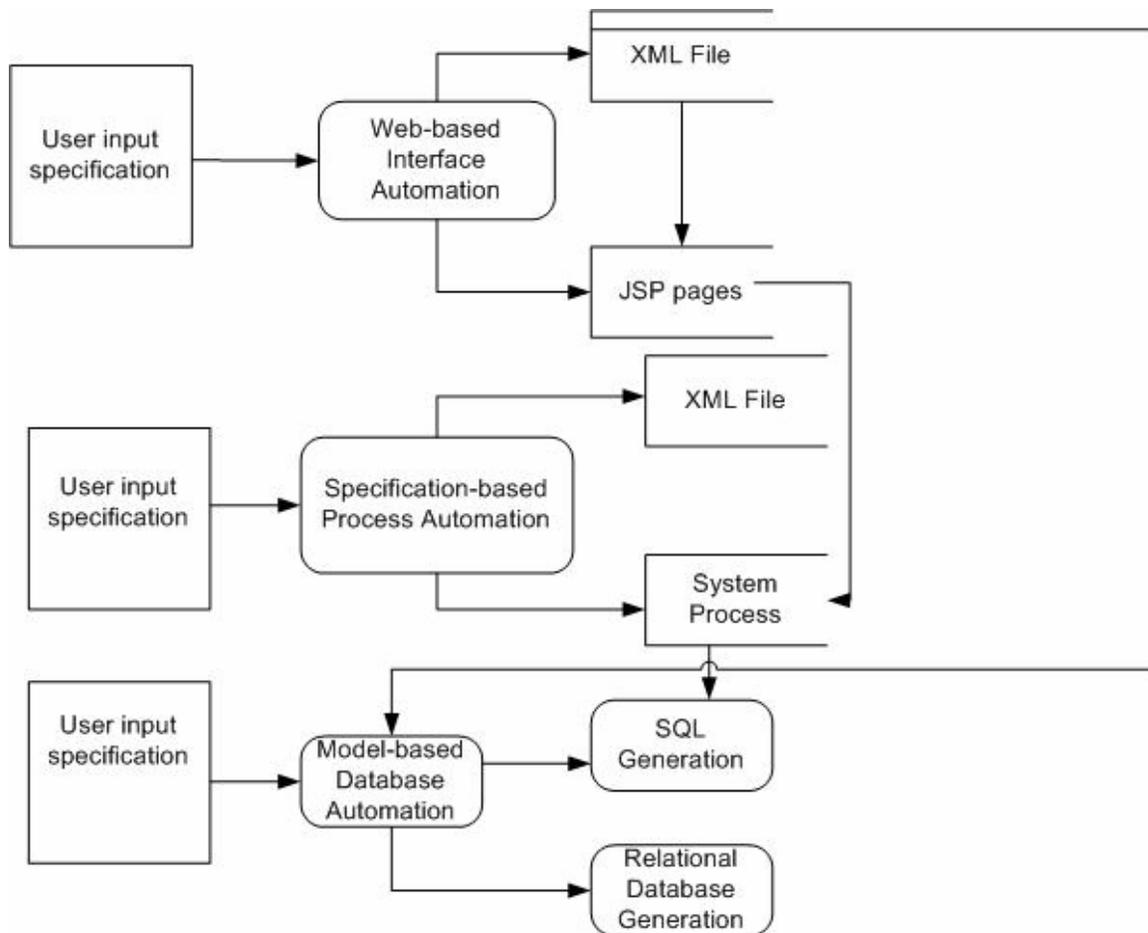


Figure 1 Modular Web Framework general structure

## 2.1 Web-based Interface Automation

### 2.1.1 Introduction

The web is becoming the interface for many services and applications. We can see the direction in which Google and many other big corporations are heading. There are many advantages with providing the interface over the web.

Firstly, it can be utilized across the internet or other type of network without any distribution of the client side application. That also eliminates the need to update and upgrade the application. This work can be done on the server side, not requiring any user to do any of the work.

Secondly, it would decrease the running time on the client side since everything would be done on the server. Even an older system can achieve better running time if its load is reduced.

Thirdly, the web interface increases the usability of the application. For many companies, such as Ebay, Amazon.com, Dell, and IBM, it is crucial to have a web-interface over the applications that they provide.

The Web-based Interface Automation (WIA) component automates the generation of the web interface for the application. The use of HTML limits the user to the static web page. So with the coming of JSP/Servlet technology, the interface on the web now could have a dynamic content and layout.

The interface component automates development of a web-based interface for the application. The component generates output display and input forms from a specification of data structures used by the other components

### **2.1.2 Comparison to Java Server Faces**

Java Server Faces (JSF) is a framework that simplifies the process of building user interfaces for Java Server applications. It was developed through the Java Community Process and now is a product of Sun Developer Network [4]. The WIA component of MWF was developed independently of JSF, with this author having no knowledge at the time of the existence of the JSF framework. However, since the two frameworks try to achieve one same ultimate goal of application automation, there are some similarities.

The first similarity is that Java Server Faces and the WIA both have dynamic web interface generation. Instead of having each page include the HTML or make part of the application visible, like a JSP or an ASP page, JSF and WIA both shield those sensitive structures and information [5]. All the processing of input data, server-side manipulation, and HTML output generation are hidden from the page, and this work is done on the server, leaving the page with a set of JSP custom tags.

The use of custom tags is the next thing that JSF and WIA have in common. The two frameworks rely on a tag library. Since there is no programming involved, the interface can be generated quickly and be more readable for the users. For expert users, the page could be written using the tags without any other support [5].

However, WIA has some distinguishing features that separate it from JSF. The first unique feature of WIA is the generation of the page tag specification could be done based upon the system model structures. Since many of the pages, ranging from those getting input to those displaying output results, correlate to the data structure or bean type, it is sometimes convenient to generate the layout of the page based on the bean structure. The feature also provides a user-interface so that the user can pull out a bean's structure, specify the tag used for each bean, and a page can then be generated from the specification. This eliminates many minor yet vital mistakes such as the misspelling of field names, or the misspelling or misuse of tag and tag options if every page required the user to specify by typing in the text.

The second unique feature of WIA is that all the page specifications are saved also under the XML format. In WIA, from the user specification, a JSP page is created and another XML specification of the file may also be saved. That is useful for later editing, regenerating the pages from the specification, or getting rid of pages that is irrelevant to the system.

Third, WIA offers an option to display the data structures in tree view. Though there is a specific requirement for the data structures in order for the tree view tag to work, it is a convenient, organized and visually pleasant to display a hierarchic data structure.

Last but not least, WIA has a distinct validation mechanism. Instead of having a custom validator for each page, or form like JSF, WIA offers a generic validator that validates the inputs from page and bean XML specification. Since the page and bean specification have the constraint for each field, besides the field type, it is possible for the validator to utilize the extra information there to validate. At this point in time, the validator is not completely implemented and will be available in the future revision of the software.

The WIA component of the framework provides some advantages over the use of pure JSPs or even JSF in some aspects. First of all, it is more secure than a conventional JSP page. With the use of WIA, the page will be visible with a set of tags, instead of revealing any structure of the input/output data, or containing any sensitive information of the server and host. Many times during the development process, the server was down, and the user at that time could view the JSP page as plain text, thereby disclosing information about the JSP and related components.

The WIA component provides a lighter weight solution to the view in the MVC model in comparison with JSF. Having the state and validation handler shielded on the server side, and just including the tags, WIA reduce the weight and complexity of the component.

However, the WIA component poses some disadvantages as well. Utilizing the web as the interface, WIA limits itself and the display types to the limit of the HTML specification. There are a limited number of controls that could be made visible on the web and the component has a ceiling on that capability.

## **2.2 Specification-based Process Automation**

### **2.2.1 Introduction**

The Specification-based Process Automation (SPA) component of the framework provides a means to automate the process or controller of the system. Instead of having a controller written specifically for a flow of work, a set of units will be tied together to accomplish the process. Each unit utilized does a specific operation, ranging from low level work to higher level work.

### 2.2.2 Comparison of features with Spring

Spring is a full-stack Java/J2EE application framework. It delivers significant benefits for many projects, reducing development effort and costs while improving test coverage and quality [1].

Rod Johnson states that the Spring Framework allows users to specify and utilize the automation of bean generation. Like Spring Framework, SPA provides the bean specification to be saved into a XML, and then the system would generate the bean upon the XML at run time. There would be no concrete bean or data model class in the system. All the beans can be generated from the XML specification and used throughout the system at runtime [2].

Like the Spring Framework, the SPA component of the framework is modular and comprehensive. The SPA framework is sufficient with the currently-provided functionality, and the users can change the order in which those modules can be put together [2].

Unlike the Spring Framework, SPA allows a user to specify a process using the XML specification. The specification is a description of what concrete individual units would be put together and how they would be constructed. With each unit responsible for an individual, complete, yet unsophisticated task, when put together those units can accomplish much more complex work. An example of the units that we can look at is DBSelect, which is a unit that takes in inputs either from the user or from the specification, and performs a query against the database. The result is in the form of a generic data structure, so no more manipulation regarding the database data structure is required.

A process can be used by the system according to the XML specification without any further programming. So instead of having to write code for a controller for each flow of logic work, the user can just specify the flow in a XML specification file and the component will load the required units to complete the task.

Since there is no coding involved in managing the process, SPA allows the user to modify each flow of work easier and faster. Instead of recoding, debugging, recompiling, and redeploying the system, with SPA the user could then use the utility included to re-specify the process specification. At runtime, the process would then be loaded from the specification without any change in the code or compiled classes.

Java is a dynamic class loading language [7]. Unlike, C++ or C, which link the compiled source code into an executable program, at runtime the Java Virtual Machine (JVM) loads the initial required classes, and load the classes the system needs on the run. So by having a map or specification of classes that would be required for the system, SPA would not affect the performance of the system significantly. At runtime, a processor, in the role of a mediator, will load the specification, then load the required classes, one by

one, to accomplish the work. The mechanism that MWF is currently utilizing is consistent with the language, so it provides the flexibility but will not subtract from the performance of the system.

However, flexibility does come with a cost. Since the process specifications would be loaded onto the system, even if one process is running, or even none, the memory is still allocated for the specifications. In MWF, the specifications are loaded and put into ServletContext, so that every page can access it on the scope of the application. Normally, the specification would not change through out the life cycle of the system after it is loaded, so it remains to be read-only. However, if the system allows users to modify the specification when it is already loaded and running, some possible deadlock or inconsistency might occur. At this time, the specification is loaded once in the initializing phase, and will not change throughout. Any change made to the specification is saved onto the XML and is loaded in the next system initialization.

Another issue with the process automation component is that the specification is loaded onto the ServletContext and could be accessed by users. Though it requires each level of user to access a certain specification, but it could be vulnerable to being misused. It is especially important for a sensitive workflow of an organization where how a task is performed matters to the competitors. Some encoding or hashing of the specification could provide another security layer, yet it would slow down the performance.

In conclusion, the SPA component of the MWF is another way to look at process automation. It provides flexibility for the programmer, and for users of the system. It also encourages modularity for application development. The component is also easily extended with each unit only required to implement an appropriate interface and comply with the XML format for specification.

## **2.3 Model-based Database Automation**

### **2.3.1 Introduction**

In many systems today, the database also evolves as the application changes. Migration and/or creation of a new database is a lengthy process including designing, understanding the system model structure, and having the knowledge of the DBMS. The Model-based Database Automation (MDA) component shields the user from many of the complications of SQL commands, separates the database generation from the model structures and provides the users with an interface to design the models. The relational database will be generated from the model specification.

The database component provides several services regarding database manipulation. First, it generates relational tables for databases from the model, mapping the system data structures to the database tables. Second, it provides connectivity for several database management systems (DBMSs) including Oracle, MySQL, and SQL Server. Third, it generates SQL commands from the process model. This component shields the details of

any specific DBMS, and requires only the general knowledge of relational database systems.

### **2.3.2 Comparison of features with Hibernate**

The MDA component has many features in common with Hibernate. As described above, MDA provides generation of database and SQL commands from model structures. MDB, like Hibernate shields all the complexity of DBMS engines and SQL structures and leave the user to focus more on the logic of the application [3]. Based upon the data structure specification, the database will be generated. The constraints of each field in the data structures would also be reflected on the relational database. It also references foreign keys of related fields of tables.

At this point, a subset of SQL commands can be generated by MDA. It supports basic SQL commands such as CREATE TABLE, UPDATE, SELECT, DROP, DELETE, TRUNCATE, and INNER JOIN.

However, MDA has some features that are not included in Hibernate. First, from the bean specification and constraint in XML form, MDA can generate the relational database. In the MWF framework, there is no concrete data structure, only the specification and “soft” data structures are used in the system. With the generation of database from the XML, MDA supports the framework better than if we had to have concrete bean classes.

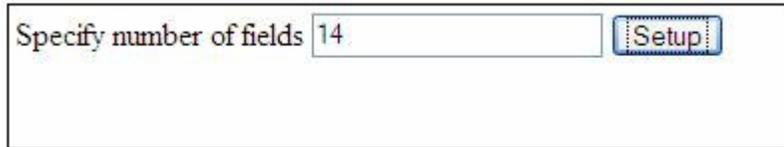
As a query service and for relational persistence, the MDA component offers many of the same features as Hibernate. It is not as fully developed as Hibernate, but it provides the necessary functionalities for the MWF system. It also supports the generation of the database from the soft data model. The author did not want to re-invent the wheel and at the point of creation was focused on providing a functional database manipulation mechanism. MDA certainly has more room for improvement, but also could be replaced or integrated with Hibernate for better SQL command generation.

## **3 The MWF Life Cycle**

At the beginning, each application that utilizes MWF would have the same number of components and units. However, the user can configure the units and components to their needs and also extend the frameworks. Like the description above, the framework has no concrete data structure or model. As we can see from Figure 1, the web interface and database generation and the process all utilize the model specified by the user. So at the start of its life cycle, MWF requires user to specify the data structure that would be used within the system, then a relational database will be generated, and lastly, the processes that utilize these elements will also be created.

Here is a walk-through example using the framework to generate the bean specification, the database, and the process. Firstly, we create a bean for a Customer and the data associate with the entity.

The system would first prompt for number of fields for each model. Figure 2 is a snapshot of the page



Specify number of fields

Figure 2 Prompt for number of fields of bean

Then it would ask the user to specify each field, name the field, and specify a type and any constraint associated with the field. The constraints would be strictly used for the database generation but it will also be used to validate the input later.



bean-name	field-name-1	field-type-1	field-constraint-1
Customer	cust_id	primeInteger	PRIMARY KEY
	cust_fname	String	
	cust_lname	String	
	cust_mi	String	
	cust_phone	String	
	cust_email	String	
	cust_username	String	
	cust_password	String	
	cust_address	String	
	cust_city	String	
	cust_state	String	
	cust_country	String	
	cust_zip	String	

Figure 3 Bean field specification

After that the system will display the beans with their specification. The page will also provide the functionality for user to Edit or Delete the bean and to generate the database script from the current beans.

With the input for the Customer structure, the system will generate a script to create a table named Customer with the column specification that correlates to the bean fields.

```

CREATE TABLE Customer(
cust_id INT(10) AUTO_INCREMENT,
cust_fname VARCHAR(30),
cust_lname VARCHAR(30),
cust_mi VARCHAR(30),
cust_phone VARCHAR(30),
cust_email VARCHAR(30),
cust_username VARCHAR(30),
cust_password VARCHAR(30),
cust_address VARCHAR(30),
cust_city VARCHAR(30),
cust_state VARCHAR(30),
cust_country VARCHAR(30),
cust_zip VARCHAR(30),CONSTRAINT CUSTOMER_CUST_ID_PK PRIMARY KEY
(cust_id));

```

The interface generation will rely on the beans in the system. So after having the beans needed for the system, the user could then specify the individual page for the purposes needed.

The interface component will first prompt for the page name and the main tag for the page. There are three choices for the main tag. The user could use the AutomateTag, which automatically presents the result or data structure in plain text form. The second option is to use TreeTag, which would represent the output in a tree view. The user could also use MainTag, which encloses other tags to enrich the display options to the needs of the user. For the purpose of this example we will go through the use of MainTag, which we have utilized more than the other two. Figure 3 shows the page to input the page name and the main tag that the page would utilize. At this point, a XML Document will be created in the back end, and each field with the display options will be attached onto the Document.

The screenshot shows a web form with two input fields and two buttons. The first field is labeled 'page\_name' and is an empty text box. The second field is labeled 'tag-name' and is a dropdown menu with 'AutomateTag' selected. Below the fields are two buttons labeled 'Setup' and 'Reset'.

Figure 3

Then the user can specify the options of the MainTag. MainTag act as an outer most tag, having some of the options that correlate to the table or form for all the nested tags inside. The system creates a XML Document associated with the input and displays the specification options for each field, as shown in Figure 4. The options in the MainTag mainly control the setting of the table and form that the fields belong to. For example, to set true to showHeader, and then the header or name of field for each field will be shown.

In Figure 5, the system would allow the user to specify the tag that would be used for each field of the data structure and the page. If a field is not specified, then it will not be displayed later in the generated result JSP page.

tag-name	MainTag
page_name	signup.jsp
bean-type	Customer
doField	boolean
beanName	Customer
action	signup.do
form	
actionValue	Signup
actionParameter	method
tableProperties	
pageSize	
horizontal	false
displayPage	false
displayForm	true
includeUpload	
showHeader	true
<input type="button" value="Create"/> <input type="button" value="Reset"/>	

Figure 4 MainTag options specification

columnName	value		
cust_id	AutomateTag	▼	<input type="button" value="Setup"/> <input type="button" value="Reset"/>
cust_fname	TextField	▼	<input type="button" value="Setup"/> <input type="button" value="Reset"/>
cust_lname	TextField	▼	<input type="button" value="Setup"/> <input type="button" value="Reset"/>
cust_mi	TextField	▼	<input type="button" value="Setup"/> <input type="button" value="Reset"/>
cust_phone	TextField	▼	<input type="button" value="Setup"/> <input type="button" value="Reset"/>
cust_email	TextField	▼	<input type="button" value="Setup"/> <input type="button" value="Reset"/>
cust_username	TextField	▼	<input type="button" value="Setup"/> <input type="button" value="Reset"/>
cust_password	PasswordField	▼	<input type="button" value="Setup"/> <input type="button" value="Reset"/>
cust_address	TextField	▼	<input type="button" value="Setup"/> <input type="button" value="Reset"/>
cust_city	TextField	▼	<input type="button" value="Setup"/> <input type="button" value="Reset"/>
cust_state	TextField	▼	<input type="button" value="Setup"/> <input type="button" value="Reset"/>
cust_country	TextField	▼	<input type="button" value="Setup"/> <input type="button" value="Reset"/>
cust_zip	TextField	▼	<input type="button" value="Setup"/> <input type="button" value="Reset"/>

[Save XML](#) [Gen JSP](#)

Figure 5 Field specification

Figure 6 shows a typical page which allows the user to specify field display options.

tag-name	<input type="text" value="TextField"/>
column-name	<input type="text" value="cust_fname"/>
column	<input type="text"/>
displayData	<input type="text"/>
all	<input type="text"/>
<input type="button" value="Insert"/> <input type="button" value="Reset"/>	
Pages [1]	

Figure 6 Field specification

After finishing with the specification for each field, the system allows the user to save the specification, now in the XML Document, into a XML file and generate the result page from it. Below in Figure 7 is the result page resulting from the process.

cust_fname	<input type="text" value="String"/>
cust_lname	<input type="text" value="String"/>
cust_mi	<input type="text" value="String"/>
cust_phone	<input type="text" value="String"/>
cust_email	<input type="text" value="String"/>
cust_username	<input type="text" value="String"/>
cust_password	<input type="text" value="*****"/>
cust_address	<input type="text" value="String"/>
cust_city	<input type="text" value="String"/>
cust_state	<input type="text" value="String"/>
cust_country	<input type="text" value="String"/>
cust_zip	<input type="text" value="String"/>
<input type="button" value="Signup"/> <input type="button" value="Reset"/>	

Figure 7 Result JSP

Process generation is the next step. After having the database, and the input and output forms, the user now can specify the process that controls all of the elements. The structure of the process specification is that it contains many functions. Each function is

a set of units, where each unit is a Java class that will accomplish one independent, complete and simple task. Each unit would have a set of conditions, which could be considered as the parameters of the functions, and a set of properties, which provides any value needed to initialize or to specify the unit.

Figure 8 shows the page which sets up the function. Figure 9 shows how the unit is added onto a function specified above.

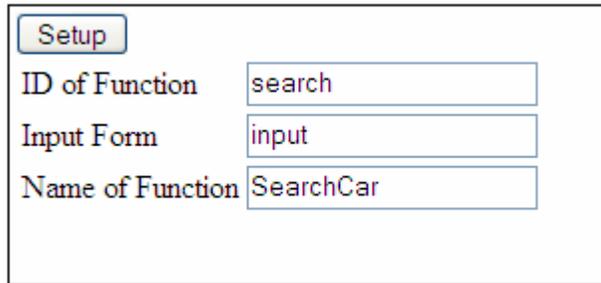


Figure 8 is a web form titled "Process Specification". It features a "Setup" button at the top left. Below the button are three input fields: "ID of Function" with the value "search", "Input Form" with the value "input", and "Name of Function" with the value "SearchCar".

Figure 8 Process Specification

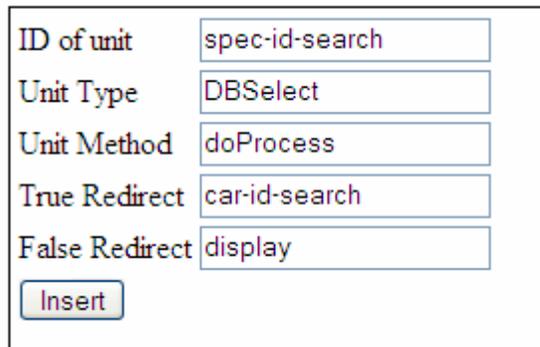


Figure 9 is a web form titled "Process Unit Specifications". It contains five input fields: "ID of unit" (spec-id-search), "Unit Type" (DBSelect), "Unit Method" (doProcess), "True Redirect" (car-id-search), and "False Redirect" (display). An "Insert" button is located at the bottom left of the form.

Figure 9 Process Unit Specifications

Then the property and condition will be added to the unit as shown in Figures 10 and 11



Figure 10 is a web form titled "Unit property". It has a single input field labeled "Property Value" and an "Edit" button positioned below it.

Figure 10 Unit property

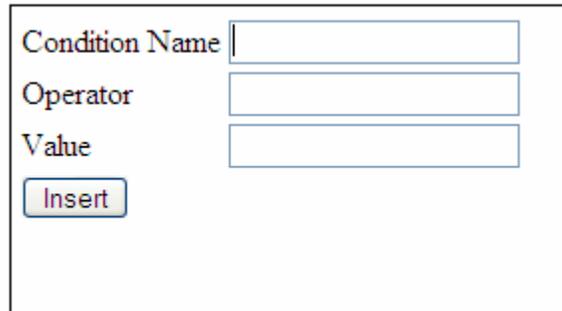


Figure 11 is a web form titled "Unit Condition". It contains three input fields: "Condition Name", "Operator", and "Value". An "Insert" button is located at the bottom left of the form.

Figure 11 Unit Condition

The result is the XML document which specifies the process or function. After this, the system will be able to load the function, and utilize it when requested.

```
<unit type="vn.automate.controller.unit.AutoDBCColumnCheck" method="doCheck"
      id="DBCheck" ifTrue="DBSelect" ifFalse="relogin">
  <condition conName="Users.userName"
    operator="=">inputForm.userName</condition>
  <condition conName="Users.password"
    operator="=">inputForm.password</condition>
  <property name="relogin">failure</property>
</unit>
```

## 4 Summary

### 4.1 Advantages of the whole system

The Modular Web Framework is a different approach to web application development. MWF provides automation in the three aspects of the development process, which is interface generation, process automation, and database manipulation. It not only gives the user a complete set of tools for a web application development, but also it provides the flexibility and convenience to maintain and update the system later.

Using the three standard frameworks compared above (Java Server Faces, Spring, and Hibernate), in place of the three MWF components, leaves the user facing several problems. Firstly, the user is required to have a sufficient knowledge of the three frameworks. Though the three frameworks are Java-based, each of them is independently developed. Each framework also utilizes different technology, so each requires a separate learning curve. Secondly, to have three frameworks to work as one system, the user needs to integrate them. Using MWF, all components are developed cohesively, thus no integration is required for the components to work together. Thirdly, MWF includes a user interface for system configuration. A user of MWF could configure the system without an in-depth understanding of the underneath technology and mechanism.

### 4.2 Drawbacks of the system

Flexibility and convenience come with a cost. Firstly, since everything relies on the file system, XML specification files, and other utility maps, the initializing requires more time and CPU consumption. Space is another issue, because instead of having the data structure or controller compiled, the system requires a module for each of the data structures or processes to be specified in an XML file and loaded only at runtime. In the scope of the framework, all the structures of bean and process are stored onto the ServletContext, which consumes the web container memory allocation as well.

Secondly, the system can demand more CPU consumption and in effect might slow down performance. All the data structures in the system are “soft” data structures, meaning, there is no concrete definition of them anywhere in the system. They are specified and stored on a map and are only utilized at runtime. The same thing happens for the process automation. Each process is a set of concrete units, but which units are utilized and how they are put together are loaded and utilized at runtime.

Thirdly, since the processes are specification based, MWF might pose some security issues. Many organizations have a strong security needs, and the way each company does certain work might be crucial and vital to their existence. By having the process on the specification files instead of being enclosed inside of the code, application information might be more easily compromised.

## 5 Glossary

**Hibernate:** A framework that support relational persistence and SQL command services

**JSF:** Java Server Faces, a framework developed through the Java Community Process which support the automatic generation of web-based interface

**MDA:** Model-based Database Automation, one component of MWF, provides automation of SQL commands generation, and database creation base on the bean/data structure

**MWF:** Modular Web Framework, developed by Tuan Nguyen, is a framework to develop a web application, and it provides automation in aspects of interface, process, and database manipulation.

**Process:** A system work flow or function.

**SPA:** Specification-based Process Automation, one component of MWF, automates the processes of the system based upon the user specification

**Spring:** A Java/J2EE framework that provides support for modularity of components and automation of beans.

**Tag:** A custom JSP tag, provided as part of Web-based Interface Automation component.

**Tag options:** Attributes of the custom JSP tag, provides functional options that could be decided by user

**Unit:** A concrete class or object that can accomplish a complete yet simple task

**WIA:** Web-based Interface Automation, one component of MWF, assists in the automation of the web interface, based on the system data structure (or bean type)

**XML file:** A specification of a bean, a process or an interface. It is generated from user’s inputs through the help of a user interface included in the system. It is also used to specify the according object in the system.

## References

[1] Spring Framework. <http://www.springframework.org/>. March 2, 2006

- [2] Johnson, Rod. Introduction to the Spring Framework. May 2005  
<http://www.theserverside.com/articles/article.tss?l=SpringFramework>
- [3] Hibernate. <http://www.hibernate.org/>. March 1, 2006
- [4] Java Server Faces Technology. <http://java.sun.com/javase/javaxserverfaces/>. March 7, 2006.
- [5] Mahmoud, Qusay H. Developing Web Applications With Java Server Faces.  
<http://java.sun.com/developer/technicalArticles/GUI/JavaServerFaces/>
- [6] Geary, David. A first look at Java Server Faces  
<http://www.javaworld.com/javaworld/jw-11-2002/jw-1129-jsf.html>
- [7] Sosnoski, Dennis. Java Programming Dynamics, Part 1: Classes and class loading.  
<http://www-128.ibm.com/developerworks/library/j-dyn0429/>. 29<sup>th</sup> April 2003