IJCTA, 9(23), 2016, pp. 107-112

Behaviour Analysis of J2EE Component at Run Time

Parveen Kumar* and Pradeep Tomar**

Abstract : Making the changes and to analyse the behaviour of dynamic software applications at run time is a complex task. Dynamic parameter passing plays a important role in the software application development. This provides a mechanism to link a component dynamically so that it's functionality could be used. It is very important to understand the dynamic behaviour of the component in an application to fulfil the different real system requirements at run time. This paper presents an efficient approach for dynamic behaviour analysis of software components. This dynamic behaviour of the components is obtained when these components are in execution. The main aim of the paper is to find out runtime behaviour of the components in Java Enterprise Edition (JEE) environment .

Keywords : Dynamic Software System, Dynamic Parameter, Software Component.

1. INTRODUCTION

Application Software systems are the backbone of our daily life and can be used in almost every domain like business, military ,banking, transportation and government organisations. Therefore it is mandatory to understand a software component's behaviour in runtime environment by exploring it's accuracy and correctness .

Dynamic program analysis can be defined as the analysis of execution of different software components involved in real time application scenario. This dynamic analysis will be more effective if the target program is executed with sufficient set of input parameters to generate its different behaviour.

The dynamic analysis needs the execution of the software with a set of relevant input data. With this input parameter set, execution of program behaviour need to be observed. Dynamic analysis can examine the actual and exact runtime behaviour of a program. Thus, dynamic analysis can play an important role in program optimization and understanding .Dynamic analysis leverage run time information to provide better optimisation.

Making the required changes to a running system can affect the commercial web-applications offering professional services. Implementing changes in a running software application is not an easy task. Whole lot of effort goes into maintaining the consistency of the system. A system description requires the details of its runtime behaviour that uses dependencies between instances of components. The aim of dynamic analysis is to incorporate the required changes in component- based systems at the time of execution.

A component is defined by its interface and the services provides to other components to fulfil the system requirements, rather than by its implementation behind the interface.

* School of Information and Communication Technology Gautam Buddha University, Greater Noida-201312 Gautam Budh Nagar, Uttar Pradesh, India psehrawat05@gmail.com

** School of Information and Communication Technology Gautam Buddha University, Greater Noida-201312 Gautam Budh Nagar, Uttar Pradesh, India parry.tomar@gmail.com

2. ENABLING COMPONENTS BASED ON DIFFERENT PARAMETER CALLING AT RUNTIME

Using a Call By Name operation, we call a public program defined in the called application. The called application behaves like a regular component. It will be loaded in a regular manner. Once the application is loaded, the public program will be called. This instant is observed by a software during runtime execution and analysis of the captured information where paths are traced through software components and then aggregated to understand global system behaviour via statistical inference.

A set of parameters passed to the runtime environment for the live components .This parameter passing implies the enabling and disabling of components which reflects corresponding changes in a system interface. Dynamic analysis helps to execute components in a consist running system. It helps to solve a major problem in managing runtime dependencies among the components. Structural changes of the running system can be analysed to manage consistency problems to swap the enabling and disabling of different components at runtime.

Most of the times , incorporating new changes in the system results in the changes in the system interfaces and may lead to enabling and disabling of components in order to obtain a stable system. So, it seems to be an issue to manage the runtime dependencies among the components involved while we are incorporating changes in the system during execution. There are instances in which we wish to load a component dynamically or even conditionally. A certain component can be loaded according to one condition and a different component according to a different condition. It can be required to dynamically call different programs of the same component according to a condition.

3. IMPLEMENTATION CONCEPT OF THE COMPONENTS IN WEB BASED APPLICATION

For implementing dynamic behaviour of component in the web-based application, a single user interface (a web page) has been developed in such a way that it can dynamically alter its behaviour based on certain values or parameters such as who is using it and what purpose it is going to be used for. The dynamic execution of the components always be hidden from users and user can only use functionalities of the dynamically executed components. So a user need not get into the details but need to know only the functionalities provided by the component.

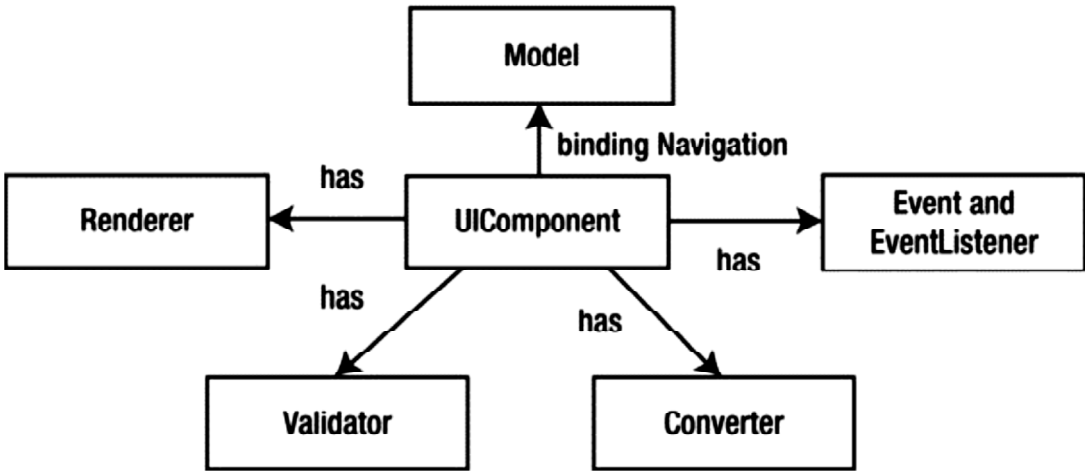

Fig. 1. The JSF component model.

Java Server Faces (JSF) is a Java-based web application framework for developing component based User Interface (UI) for server based applications. The reusable UI components are constructed to develop the web based applications. JSF reduces the complexities and effort in web application development which run on a Java application server and render UI on to a target client. Figure 1 shows the JAVA UI component Model. JSF provides the multiple rendering capabilities in which UI components render themselves differently depending on client type.

Here in the current scenario, the user interface has been designed and developed using prime faces, an open source component library for Java Server faces. Prime faces JSF components provide provision to make rich front end interfacesfor web and mobile applications. The UI components provided in the library are laid down on the JSF/XHTML pages such that some of them are disabled while some are enabled. This enabling and disabling of UI components in dynamic fashion is being controlled through a logic implemented at the backend(database layer). This allows for minimal changes in the application source code as and when changes need to be made in the policies regarding the conditions for enabling /disabling of UI components.

A java class has been designed which contains attributes as data members corresponding to each UI component on the JSF/XHTML page such that each data member in the class is associated with one UI component . This linking is established at the time of design and development of JSF/XHTML page. This java class also contains a pair of methods , Accessors and Mutators (also known as getter and setter methods respectively) corresponding to each data member which is mapped to some UI component on the user interface. Apart from these members functions , the java class also contains methods implementing several business operations .This java class is managed by the web container only in the sense that programmer need not bother about its instantiation and life cycle management . even the values in the data members linked to the UI components is done automatically without writing an Adhoc code for this purpose. Thus this is also popularly known with the name Managed Java Bean. The java bean contains the code for interacting with Middle ware, a transaction monitoring tool (Oracle Tuxedo) which hosts the business logic for data base interaction.

JSF can be seen as a UI (User Interface) standard for Java based web applications and is based on the framework which follows the MVC (Model-View-Controller) design pattern. Due to this fact, the applications developed using this technology are more manageable as the logic for the creation of user interface (View) is separated from the application logic part also known as Model .All the navigation within the application is handled by an inherent servlet code which acts as a Controller.

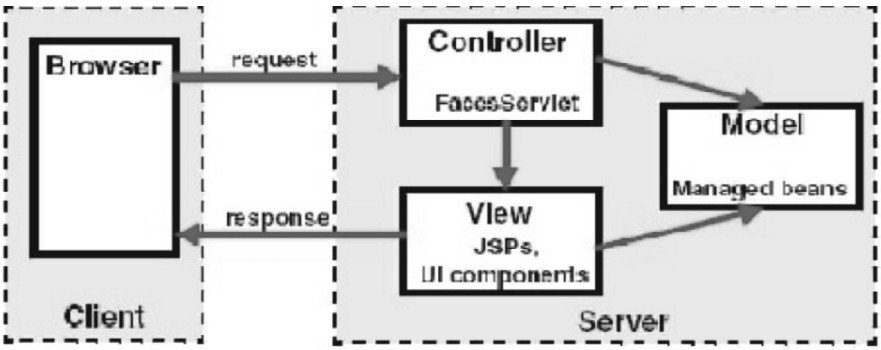

Fig. 2. The JSF Deployment Architecture model.

- **JSF Pages :** These are the front end web pages built using the JSF components. The JSF framework has got corresponding Java classes for the components which execute on the server side.
- **Faces Servlet :** All the user interaction with the application are handled by this servlet provided by the framework itself and ca n be seen as a navigation controller.
- **Configuration files :** These are the XML files (like faces-config.xml etc.)which acts as a listing of several application related information like navigation rules, validators and managed beans
- **Tag libraries :** These libraries are there for representing Event Handlers and validators along with libraries for rendering those UI (User Interface) components.
- **Validators :** These are basic Java classes which are supposed to validate content of JSF components.
- **Managed beans :** These can seen described as Plain Old Java Objects (POJO) which hold data from JSF components though business logic and UI.
- **Events :** These can be seen as Java code which are executed on the server corresponding to the events. Event handling passes managed beans to business logic.

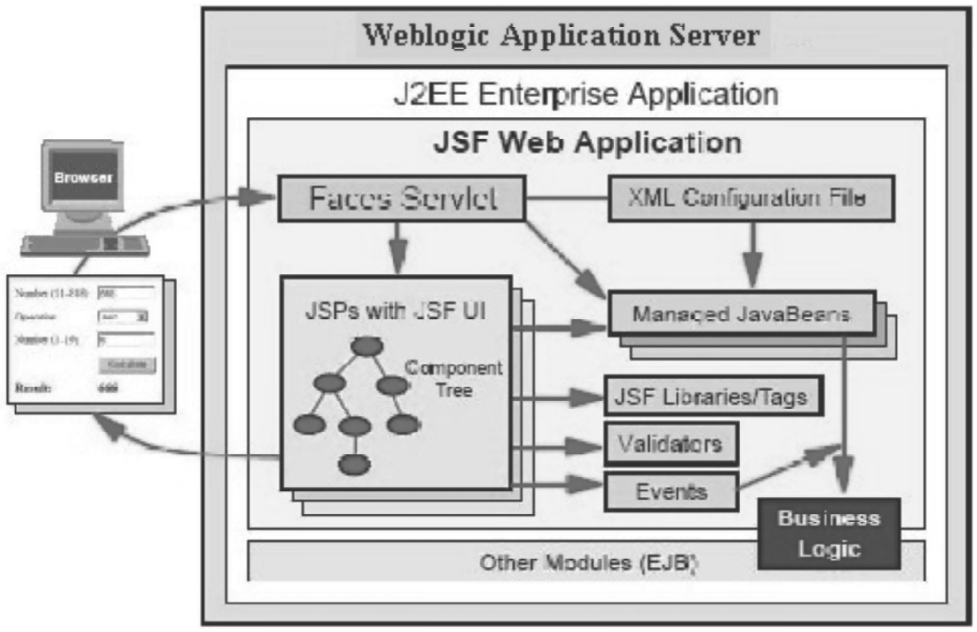

Fig. 3. The JSF application Structure Model.

The component tree is involved in every lifecycle phase and thus potentially traversed several times. This suggests that the size of the component tree has impact on the duration of the JSF lifecycle und thus on the performance of the application.

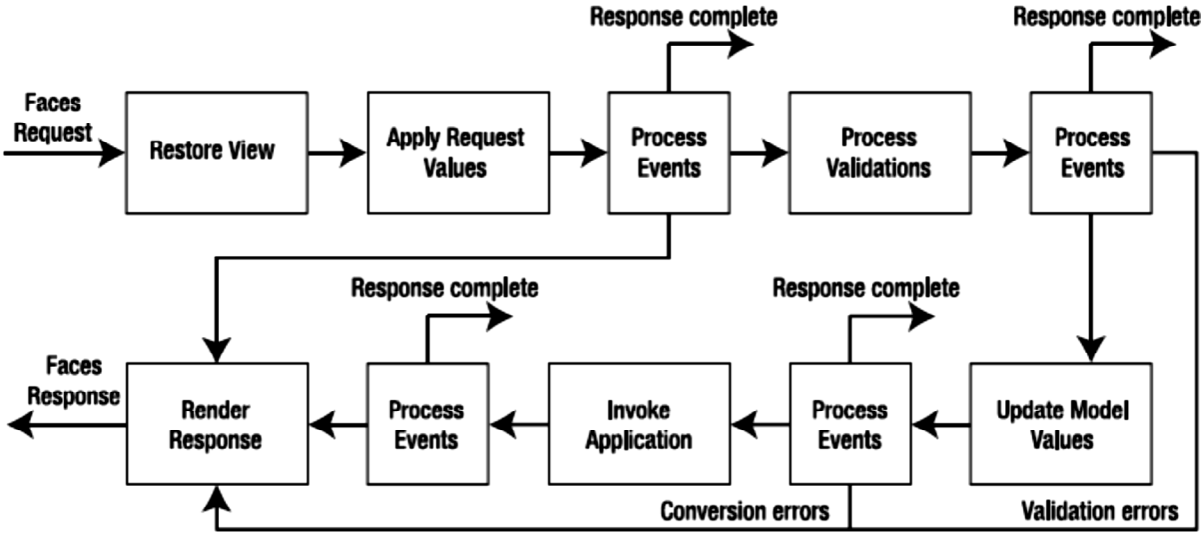

Fig. 4. JSF Component Life Cycle Phases.

Unlike JSP and other front end technologies , JSF in itself is a bit complicated . The reason behind this is due to the fact that MVC being the core concept , the whole user interaction need to be maintained right from clicking till rendering the appropriate components after validation etc.

This whole process goes through a series of steps and is popularly known as the JSF life cycle. Processing of a JSF form basically goes thorough six phases.

Phase -1 Restore View

The moment a user clicks or requests for a JSF page, the Restore View phase comes into action where the actual bootstrapping takes place. It is in this phase only where the view is constructed and the related components like event handlers and validators are linked. In this phase, the view is saved in the FacesContext instance.

Phase-2 Apply Request Values

Once the component tree is restored, each of the component in the tree extracts its value from the request parameters. In case of any error in this process, an error message is queued in the FacesContext instance which gets displayed during render response phase.

Phase-3 Process Validation

It is the phase where all the registered validators are executed for the components present in the component tree. In case of any error, an error message is queued in the FacesContext instance which gets displayed during render response phase.

Phase-4 Update Model Values

Once the JSF has confirmed the validity of the data, the corresponding server side properties for each of the component is update with the local value.

Phase-5 Invoke Application

This phase basically takes care of the execution of the application level events like submission of form and linkage to other forms.

Phase-6 Render Response

In this phase , each of the component in the component tree is traversed and renderer for each component is executed which creates the HTML mark up for each of the requested view.

4. PERFORMANCE AND BEHAVIOUR MEASUREMENT OF COMPONENTS

Performance and behaviour measurement of web applications is very complex task. Figure 3 shows the relative time taken for a user interaction with the components in terms of each phase of the JSF lifecycle. It can be clearly seen that the time elapsed is in proportion with the size of the component tree. Phases 1 and 6 can be considered the most consuming phases in the lifecycle.

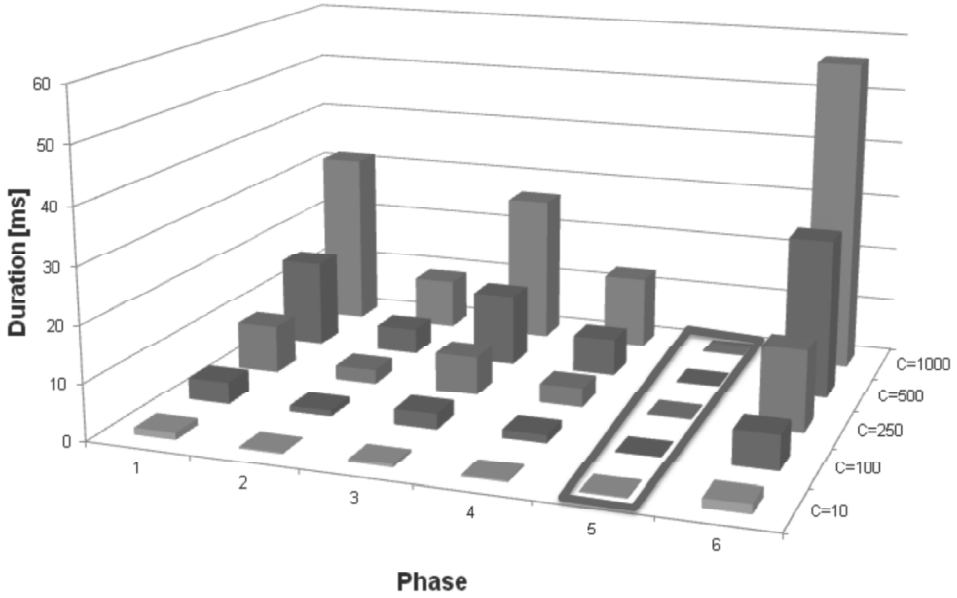

Fig. 5. Execution Time of Lifecycle Phases w.r.t. component tree sizes.

Time taken by the phase 5 is almost zero as the demo application does not contain any business logic. Since the demo application had components which required validation, time consumed by phase 3 is quite significant. The above graph depicts that small size component trees are better for the analysis of the run time behaviour of a JSF application.

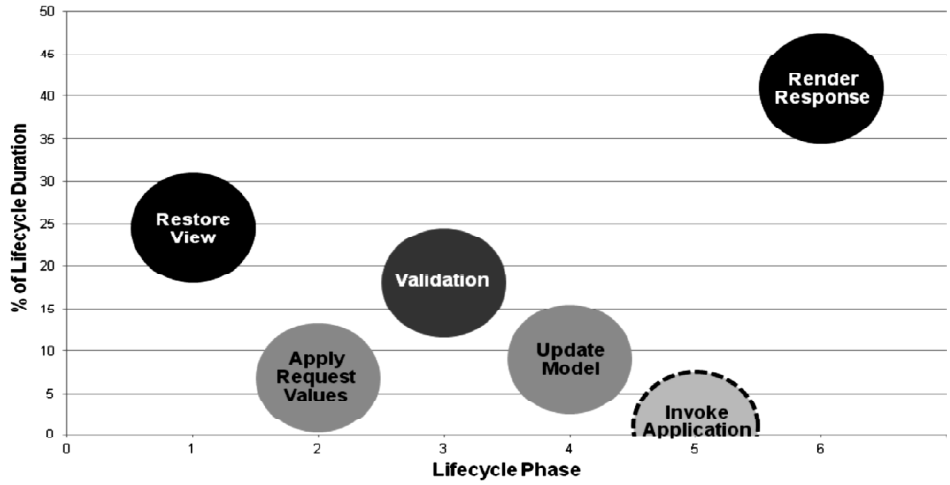

Fig. 6. Total time taken per phase in the lifecycle.

After analysing , it is found that quite often the component tree used in a random view comprises of a good number of components(More than 1500 and up to 4000).These components can turn into an overhead in the rendering of the view as the whole component tree needs to be traversed during execution of the phases of the JSF life cycle. There is a probability that some of the components in the component tree are irrelevant in terms of functional behaviour provided by the UI. Such components in the component tree just add to the response time elapsed in the rendering of the UI.The above figure concludes that reduction in the number of HTTP request can actually optimize the performance of the web application.

5. CONCLUSION

The research work focuses on run time analysis of component-based systems in order to incorporate changes in the dependencies among components present in the system. For the implementation purpose JSF/ TUXEDO/Oracle Technologies are employed. Currently, the work is based on an implementation of a JSF/ TUXEDO system for runtime analysis . A special focus of this paper presents our work that aims to implement the enabling and disabling of different components based on the different parameter involved in the system. Future work includes consideration of simulation methods for study of the behaviour of these components.

6. REFERENCES

1. Welf Lowe “Understanding Software-Static and Dynamic Aspects “ IPD, University of Karlsruhe Germany.
2. Jasminka Matevska-Meyer, Sascha Olliges, Wilhelm Hasselbring. “Runtime Reconfiguration of J2EE Applications**”** Department of Computing Science 2004.
3. Carlo Ghezzi, Andrea Mocci, and Mario Sangiorgio “Runtime Monitoring of Functional Component Changes with Behavior Models”.
4. N. Geethanjali, S.Priyadarshini, Dr.S.Karthik “Detection of unsafe component loading using dyanmic analysis technique” International Journal of Advanced Research in Computer Engineering & Technology (IJARCET) Volume 2 Issue 12, December 2013.
5. Hausi A. M¨uller and Norha M. Villegas “Runtime Evolution of Highly Dynamic Software”.
6. Andreas Bauer, Martin Leucker and Christian Schallhart “Runtime Reflection: Dynamic model-based analysis of component-based distributed embedded systems”.
7. Matevska-Meyer J and Hasselbring W , “Enabling reconfiguration of component-based systems at runtime.”, in J. B. J. van Gurp, editor, Proceedings of Workshop on Software Variability Management, pages 123–125, Groningen, The Netherlands, Feb. 2003. University of Groningen.
8. SEARLS R., J2EE Deployment API Specification, Version 1.1, Sun Microsystems.
9. http://java.sun.com/j2ee/tools/deployment/, Nov. 2003. Retrieved 2004-03-30.
10. Sun Microsystems http://java.sun.com/j2ee, Java 2 Platform Enterprise Edition Specification, Version 1.4, 2003.
11. Book Ian Hlavats “Jsf 1.2 Components”.