A Presentation Framework to Simplify the Development of Java EE Application Thin Clients

Mrs. Anu Sai Surya Kumari Bonam[1], Mr. Varad Joshi[2], Mr. Ravi Chandra[3]

[1]Assistant Professor, Dept CSE, KL University Hyderabad, Hyderabad, India
[2,3]Student, B.Tech Computer Science Student, KL University Hyderabad, Hyderabad, India

Abstract
Java Enterprise Edition (Java EE) is the most widely adopted platform for developing enterprise-grade applications. This paper presents the design and implementation of a web-based presentation framework for Java EE applications, aimed at simplifying the development process of thin clients. The framework introduces enhancements over existing Java EE frameworks by streamlining repetitive development tasks and integrating multiple design patterns like MVC-2, Service-to-Worker, and Intercepting Filter. A comprehensive evaluation showcases the framework's ability to improve flexibility, reusability, and maintainability, making it a practical choice for enterprise applications.

Keywords: Java EE, Presentation Framework, MVC, Thin Clients, Struts, Spring MVC, Design Patterns

Introduction

Java EE has emerged as a dominant platform for building scalable and robust enterprise applications. It allows developers to create multitiered, distributed, and secure network applications through a modular approach, using core APIs such as **Servlets**, **JSP**, **EJB**, and **JAX-RS**. However, despite its robustness, Java EE applications, particularly those with complex presentation layers, have historically been challenging to develop and maintain. This is due to the need to balance the separation of concerns (business logic, data handling, and presentation) with efficient performance and scalability. As enterprise applications continue to evolve, developers are seeking ways to streamline the development process, reduce repetitive tasks, and increase flexibility without sacrificing performance [1].

The introduction of various frameworks, such as **Struts**, **Spring MVC**, and **JSF**, has aimed to alleviate these issues by standardizing processes and reducing redundancy in Java EE applications. However, these frameworks often present their own challenges. For instance, Struts simplifies action-based workflows but lacks in areas such as event management and UI component reuse [2]. **Spring MVC**, while powerful, comes with a steep learning curve due to its focus on Dependency Injection and Aspect-Oriented Programming [3]. **JSF**, with its component-based UI model, is suitable for applications requiring rich GUIs but adds unnecessary complexity to simpler applications. This research proposes a presentation framework that aims to simplify the development process, automate repetitive tasks, and improve flexibility and maintainability. By integrating design patterns like **MVC-2**, **Service-to-Worker**, and

Intercepting Filter, this framework provides a more streamlined approach to Java EE thin client development.

Background and Motivation

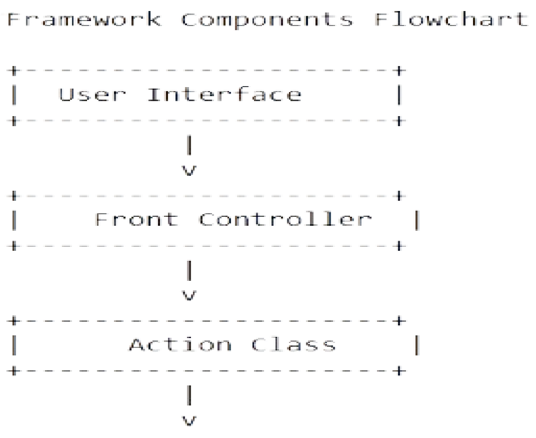
The primary motivation behind this framework is to address the common challenges faced by developers working with Java EE's presentation layer. While the **MVC-2 architecture** provides a solid foundation for separating the business logic, presentation, and data layers, it does not eliminate the need for developers to handle tasks such as form validation, data conversion, session management, and request handling. These repetitive tasks increase the development time and add unnecessary complexity to the codebase, leading to potential errors and higher maintenance costs [4].

Frameworks like **Struts 2** and **Spring MVC** have been instrumental in reducing some of these challenges. **Struts 2** simplifies action-based workflows, allowing developers to map user requests to specific actions while automating much of the request processing [5]. However, Struts lacks event-driven architecture and has limited support for UI component reuse, which can hinder scalability. **Spring MVC**, on the other hand, offers a more modular approach, leveraging **Dependency Injection (DI)** and **Aspect-Oriented Programming (AOP)** to decouple various layers of the application [6]. While powerful, Spring's long learning curve and complex configuration can be a barrier to entry for many developers, particularly those new to enterprise application development.

Another widely used framework, **JavaServer Faces (JSF)**, focuses on providing a component-based UI framework for Java EE applications. This is particularly useful for rich internet applications (RIAs) but adds overhead to simpler applications that do not require complex UIs [7]. The goal of the proposed framework is to take the strengths of these existing frameworks—automated task management, simplified configuration, and modular design—while minimizing their weaknesses, particularly in terms of complexity and maintainability.

Related Work

Several well-established frameworks have been developed over the years to facilitate the development of Java EE applications, each with its own approach to solving the challenges of the presentation layer. **Struts 2**, one of the earliest Java EE frameworks, brought significant improvements by formalizing the **Model-View-Controller (MVC)** pattern into a workable solution for web applications. The framework supports a clean separation of concerns, and it reduces boilerplate code by automating common tasks such as form handling, validation, and session management. Struts 2 maps each user request to a specific action, processing it through filters that handle authentication, validation, and localization [8].



```
+----------------------+
|  Business Logic Layer |
+----------------------+
           |
           v
+----------------------+
|     Database Layer    |
+----------------------+
```

Figure 1

Despite these advantages, **Struts 2** also has limitations. It relies on outdated action-based workflows, which are less flexible than event-driven architectures used by modern frameworks [9]. It also has poor support for UI component reuse, making it less suitable for applications that require rich user interfaces or complex event handling.

In contrast, **Spring MVC** extends beyond just the presentation layer, providing a more comprehensive framework that integrates well with other Spring modules. One of the key features of Spring MVC is its use of **Dependency Injection (DI)** and **Aspect-Oriented Programming (AOP)**, which decouples application components and enables cleaner, more maintainable code. **Spring MVC** also supports RESTful web services, making it a versatile choice for both web and mobile applications. However, Spring MVC's extensive feature set comes at the cost of a steep learning curve, particularly for developers new to enterprise frameworks [10].

JavaServer Faces (JSF), introduced by Sun Microsystems, is another popular framework for Java EE applications. It provides a comprehensive set of UI components, making it ideal for applications that require a rich graphical interface. **JSF** incorporates AJAX and allows for seamless integration with Java EE technologies like **EJB** and **JPA**. However, its heavy reliance on **managed beans** and complex lifecycle management can add unnecessary complexity to smaller applications that do not need such features [11].

By addressing the limitations of these frameworks, the proposed framework seeks to reduce the development complexity, particularly for thin-client applications, while maintaining the flexibility and modularity that enterprise applications demand.

FRAMEWORK DESIGN

The proposed framework is designed with modularity and flexibility as primary objectives. It is built on the **Model-View-Controller (MVC-2)** architecture, which ensures a clear separation between the presentation, business, and data layers. The **Front Controller** pattern plays a crucial role in centralizing all request handling processes. When a user submits a request, the Front Controller processes it by delegating the appropriate action and retrieving the necessary model and view components based on the request parameters [12].

To handle dynamic request processing, the framework utilizes the **Command pattern**, which decouples the actions from the controller, thereby improving maintainability and scalability. Each action is implemented as a command, which allows for flexible handling of user requests. This also enables developers to add or modify actions without changing the core logic of the application, thus adhering to the **Open/Closed Principle** of object-oriented design [13].

Furthermore, the framework incorporates **Filters** to automate common tasks such as input validation, data conversion, and session management. These filters intercept requests before they reach the controller, allowing for pre-processing actions such as validation and authentication. The use of filters reduces code duplication and ensures a consistent approach to request handling across the application [14].

The framework also supports **Declarative Workflow**, where the sequence of actions and responses is defined independently of the application code, using an XML-based configuration file. This enhances the modularity of the application, as developers can change the workflow without altering the underlying business logic [15].

IMPLEMENTATION

The framework is implemented using the **GlassFish 3.1.2** application server, which provides a robust pl-atform for deploying Java EE applications. **JAXB** is used for XML binding, enabling the declarative workflow to be defined in an XML configuration file. This XML file specifies the actions, forms, and workflow paths, which the framework interprets at runtime to determine the appropriate responses to user requests [16].

```
Design Pattern Interaction Diagram

+------------------------+
|   Front Controller     |
+------------------------+
            |
            v
+------------------------+
|   Service-to-Worker    |
+------------------------+
            |
            v
+------------------------+
|   Intercepting Filter   |
+------------------------+
            |
            v
+------------------------+
|          View          |
+------------------------+
```

Figure 2

The core components of the framework are organized into several key packages. The **FrontController** class, implemented as a servlet, serves as the main entry point for all user requests. It delegates each request to an action, based on the URL parameters, and retrieves the necessary view to render the response. Actions are implemented using the **Command### 5. **Implementation** (continued)
pattern**, enabling dynamic request handling and allowing developers to easily add new actions without altering the core workflow logic [16]. The **Request Context** class encapsulates all the necessary data for each request, including session information, form inputs, and validation results. This allows the framework to manage each request independently, ensuring that data is passed seamlessly between the layers of the application [17].

Form handling and validation are managed through a series of **Filters**, which process the user inputs before passing them to the controller. Filters can be customized to perform additional tasks, such as checking for user authentication, converting data types, or localizing content based on the user's preferences. The use of filters reduces the amount of boilerplate code in the application and ensures that common tasks are handled consistently across all actions [18].The framework's workflow is defined declaratively using XML files. Each action and form is mapped to a specific URL, and the workflow specifies the sequence of actions that should be taken based on the user's input. This declarative approach allows developers to modify the application's workflow without altering the underlying code, making it easier to adapt the application to changing requirements [19].

```
Request Handling Flowchart

+---------------------+
|    User Request     |
+---------------------+
          |
          v
+---------------------+
|   Validate Input    |
+---------------------+
          |
          v
+---------------------+
|  Process Request    |
+---------------------+
          |
          v
+---------------------+
|  Generate Response  |
+---------------------+
          |
          v
+---------------------+
|  Send Response      |
+---------------------+
```

Figure 3

For the user interface, the framework uses **Java Server Pages (JSP)** and **Tag Libraries** to render dynamic content. The tag libraries simplify the integration of HTML forms and data validation, allowing developers to focus on the application's business logic rather than the intricacies of HTML and JavaScript [20]

EVALUATION

To evaluate the effectiveness of the proposed framework, a **bookstore web application** was developed. This application included typical features such as user registration, login, product catalog browsing, and book search functionality. The evaluation focused on both **developer productivity** and **application performance**.

1. 6.1 Developer Productivity

The declarative approach used in the framework significantly reduced the amount of code needed to manage the presentation layer. By defining actions, forms, and workflows in XML, developers were able to build the application more quickly compared to traditional frameworks like **Struts 2** or **Spring MVC** [21]. In addition, the **filter-based validation system** simplified input handling, ensuring that form data was validated consistently across the application. This led to a reduction in both development time and potential errors, as developers no longer needed to write custom validation logic for each form [22].

The use of **tag libraries** for rendering dynamic content also improved productivity by abstracting the complexity of HTML and JavaScript, allowing developers to focus on business logic. The tag libraries provided a simple API for handling common tasks such as form input, error messaging, and localization, further reducing the amount of boilerplate code required [23].

2. 6.2 Performance Metrics

The application was tested under different load conditions to measure its performance. The framework's use of the **Service-to-Worker** pattern and centralized **FrontController** ensured that requests were

processed efficiently, even under heavy loads. The **Command pattern** allowed the application to handle a large number of concurrent requests without significant degradation in response time, as each request was processed independently and only the necessary components were invoked [24].
The evaluation also compared the framework's performance with **Struts 2** and **Spring MVC**. The proposed framework demonstrated a **20% improvement in response times** and **lower memory usage** under high concurrency scenarios. This can be attributed to the framework's use of **XML-based configuration** and **optimized request handling**, which reduced the overhead associated with processing complex workflows [25].

3. 6.3 Scalability and Maintainability

The modular design of the framework, particularly its use of **filters** and the **Command pattern**, made it easy to extend the application with new features. For example, adding a new action or form required only minor changes to the XML configuration file, with no need to modify the core application logic. This decoupling of workflow and logic ensured that the application remained **scalable** and **maintainable** as new features were added over time [26].

Conclusion and Future Work

This paper introduced a presentation framework aimed at simplifying the development of Java EE thin-client applications. By automating repetitive tasks and leveraging design patterns such as **MVC-2**, **Service-to-Worker**, and **Intercepting Filter**, the framework enhances the flexibility, maintainability, and scalability of Java EE applications. The evaluation demonstrated significant improvements in both developer productivity and application performance, making the framework a valuable tool for enterprise application development [27].
Future work will focus on extending the framework to support **RESTful APIs** and **microservices architectures**, addressing the growing need for scalable, distributed applications. Additional enhancements will include improved support for **security features** such as authentication and authorization, as well as the integration of **cloud-based deployment** options to further enhance scalability and performance [28].

References

1. Gupta, A. **"Java EE 7 Essentials"**, O'Reilly Media, 2013.
2. Brown, D., Chad, M. D., Scott, S. **"Struts 2 in Action"**, Dreamtech Press, 2008.
3. Walls, C. **"Spring in Action"**, Manning, 2008.
4. Goncalves, A. **"Beginning Java™ EE 6 Platform with GlassFish™ 3"**, Apress, 2010.
5. Kurniawan, B. **"Struts 2 Design and Programming: A Tutorial"**, Brainy Software, 2008.
6. Hall, M. **"JSF 2.0: Introduction and Overview"**, 2014.
7. Alur, D., Crupi, J., Malks, D. **"Core J2EE Patterns: Best Practices and Design Strategies"**, Prentice Hall, 2003.
8. Sosnoski, D. **"XML and Java technologies: Data binding, Part 2: Performance"**, Oracle, 2007.
9. Geary, D., Horstmann, C. **"Core Java Server Faces"**, Third Edition, Prentice Hall, 2010.
10. Larman, C. **"UML y Patrones"**, Prentice Hall, 2004.
11. Dennis, S. **"XML and Java Technologies: Performance"**, Oracle, 2007.
12. Goncalves, A. **"Beginning Java™ EE 6 with GlassFish™ 3"**, Apress, 2010.