2014 Eighth International Conference on Complex, Intelligent and Software Intensive Systems

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

Santi Caballé[1], Jose-Arturo Ortega[1], Josep-Maria Camps[1], Leonard Barolli[2], Elis Kulla[2], Evjola Spaho[2]

[1]Department of Computer Science, Multimedia, and Telecommunication
Open University of Catalonia, Barcelona, Spain
{scaballe, jortegaruf, jcampsri}@uoc.edu
[2] Fukuoka Institute of Technology, Department of Information and Communication Engineering
Fukuoka, Japan
barolli@fit.ac.jp, eliskulla@gmail.com, evjolaspaho@hotmail.com

***Abstract*—The main aim of this paper is to report on the building of a software framework that greatly simplifies the development of the presentation layer for Java EE applications with thin clients. One of the design principles of the presentation layer in Java EE applications is to separate the business layer flow control and calls from the presentation layer. To this end, Java EE applications usually use the Model-2 architecture, which includes the well-known Model-View-Controller (MVC) design pattern. The advantages of the MVC pattern are a clear separation of concerns, resulting in more flexible applications, which are eventually easier to manage and update. However, the use of the MVC pattern bears many repetitive tasks that all supported applications must perform, making the development work tedious and complex. In order to overcome these deficiencies and limitations, in this paper we propose a web framework for developing Java EE applications that simplifies many development aspects and tasks, resulting in web applications that are more flexible, reusable and maintainable. The starting point of this contribution is to survey and analyze the most common existing Java EE frameworks with similar purposes in terms of functionality, applicability and pros-cons comparison among them. This study allows us to go deep into the knowledge of the presentation layer of Java EE applications and collect the appropriate requirements for our framework with the focus on easy of use. The framework is then designed and implemented with Java EE technologies and finally evaluated by an ad hoc testing approach.**

Keywords- Java EE applications; presentation framework; design patterns; MVC; Struts

I. INTRODUCTION

Java Enterprise Edition (Java EE) is the de facto Java computing platform for developing enterprise applications [1]. The platform provides an API and runtime environment for developing and running enterprise software, including network and web services, and other large-scale, multi-tiered, scalable, reliable, and secure network applications. Java EE is currently in the version 7 and software is primarily developed in the Java programming, which extends the Java Platform, Standard Edition (Java SE), providing an API for object-relational mapping, distributed and multi-tier architectures, and web services. The platform incorporates a design based largely on modular components running on an application server [3].

On the other hand, software frameworks and components are normally developed for the construction of complex software systems alleviating developers the hard work when meeting both functional and non-functional requirements, and with increasingly recognition of its strategic importance in terms of productivity, quality and cost. To this end, many web frameworks have appeared to support the development of complex enterprise software with Java EE [2][3][4][6].

However, whilst Java EE frameworks claim to simplify the development of web applications, many of them achieve the opposite by having to take on sharp learning curves. In fact, failing to meet the "easy of use" requirement and increasing certain repetitive and tedious tasks are the main issues that developers complain. In order to overcome these limitations, we propose a Java EE web framework that take great advantage of design patterns to provide developers with many advantages: (i) simplify and standardize the validation of the input parameters; (ii) decouple the presentation from business layers in separated components; (iii) centralize the control of the workflow management; (iv) high level software reuse and (v) simplify many repetitive and tedious development tasks. Eventually, the resulting web applications are more flexible, reusable and easier to maintain.

To this end, in this paper, we first start in Section II with an extensive survey of existing Java EE presentation frameworks along with the core Java EE design patterns used in our framework. The analysis results of both frameworks and design patterns set the requirements for developing our framework, which is reported from all the phases of the life cycle in Section III. In Section IV, the framework is tested by developing a naïve application for evaluation purposes. Finally, Section V concludes the paper highlighting the main ideas and evaluation results, and outlining future work.

978-1-4799-4325-8/14 $31.00 © 2014 IEEE
DOI 10.1109/CISIS.2014.59

II. Background

In this section we review the main software frameworks and design patterns supporting Java EE that we will use in later sections to present our framework.

A. Struts 2

Apache Struts [2][5] is a Java EE framework in support for the development of web applications, making their implementation easier and simpler. To this end, by automating certain tasks this framework alleviates common and tedious development work in the application domain. Struts 2 also makes Java EE applications more robust and flexible by providing an architectural solution within the domain workflow.

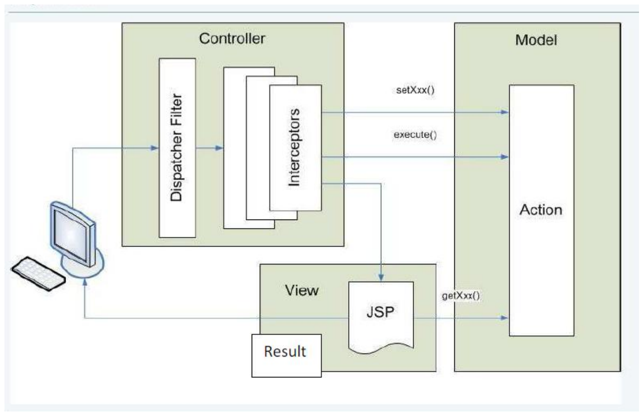


Figure 1. Struts 2 architecture [2]

Within the Java EE 3-tier architecture (i.e., presentation, logic and domain tiers, see Fig. 1), Struts 2 supports the presentation layer by implementing the Model-View-Controller (MVC-Type II) design pattern, which the framework can set up in different ways, provides specific components for the View, and allows for a smooth integration with other popular frameworks, such as Hibernate and Spring [4] supporting the Model component.

Actions are the core of the Struts 2 framework within MVC (see Fig. 1). Each URL is mapped to a specific action, which provides the processing logic necessary to service the request from the user. Actions also play an important role when transferring data from the user's request to the View (either JSP or any type of result) as well as in determining which result should render the view as a response to the request.

Finally, Struts 2 includes a library of web tags to make it easier rendering dynamic data. Tags interact with evaluation and international features in order to validate inputs and locale outputs. The library can also be used with Java Server Pages (JSP), FreeMarker, o Velocity [5]. Finally, external tag libraries can also be used in the framework, such Java Server Pages Standard Tag Library (JSTL) and support the use of Java Server Faces (JSF) components.

To sum up, Struts 2 makes the web application development easier for developers mainly by reducing the tedious configuration tasks. To this end, the framework provides an intelligent set of default values and makes use of annotations to automate configuration by following certain conventions.

The advantages of Struts 2 in comparison to other presentation frameworks (see next subsections) are [2][6]:

- Simple architecture
- Easier to learn with a short learning curve
- Complete data conversion and validation
- Most concise and mature and with the most functionalities [5]
- The most used framework [1]

Struts 2 disadvantages are:

- Poor event management
- Lower reuse level of UI components

B. Spring MVC 3

Spring [4] is a framework purposed to simplify the development of enterprise applications in Java. Spring uses simple JavaBeans with similar results to complex Enterprise JavaBeans (EJB) of previous versions to EJB3, which required to use complex methods such as ejbActivate(), ejbpassivate(), etc. In Spring, typically any component (see Fig. 2) is a simple Java class (Plain Old Java Object or POJO).

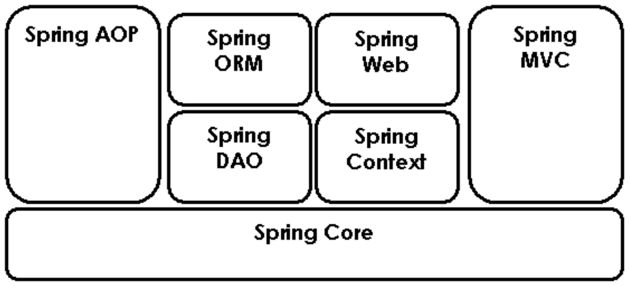


Figure 2. Major Spring components [3]

In order to reduce complexity, Spring proposes four strategies:

- Light development with simple Java classes or POJOs.
- Weak coupling by injecting dependencies (further information on the Spring concept of Dependency Injection see [3]) and interface orientation.
- Declarative programming by aspects and common conventions.
- Decrease of repeated code by the use of templates and aspects (e.g. database connection management).

The advantages of Spring in comparison to other presentation frameworks (see rest of subsections) are:

- Reduction of significant configuration work by automating connection and detection.
- Use of simple POJOs by annotations
- Use of Aspect Oriented Programming (AOP) to manage transactions, security, etc.

- MVC Spring 3 includes compatibility with Web services REST.
- Further an MVC framework, Spring includes messaging, testing utilities, etc.
- Straightforward integration with other frameworks

The main disadvantage of Spring is the long learning curve due mainly to the many capabilities offered.

C. *Java Server Faces 2*

Java Server Faces (JSF) [6][7] is a specification published in 2001 by the Java Community Process (JCP) as Java Specification Request (JSR-127). In 2006 JSF 1.2 was incorporated in Java EE 5 as JSR 252. In this version JSP was used but with integration problems with JSF due to different life cycles and as a result Facelets were introduced as an alternative to JSP with more capabilities, such as support to XHTML pages. Finally, JSF 2 was developed as JSR 314 becoming the preferred option in Java EE 6 for web development. JSF 2 was inspired in many open source web frameworks, thus having many of their features.

The purpose of JSF is to develop web applications similarly to desktop applications with Java Swing, AWT, SWT and related APIs. This way, JSF simplify the construction of web applications by providing a web framework that manages the web actions performed by the user and translate them into events sent to the server to update the web page. This way, JSF develops web applications where the client window is a HTML page instead of JFrame or similar.

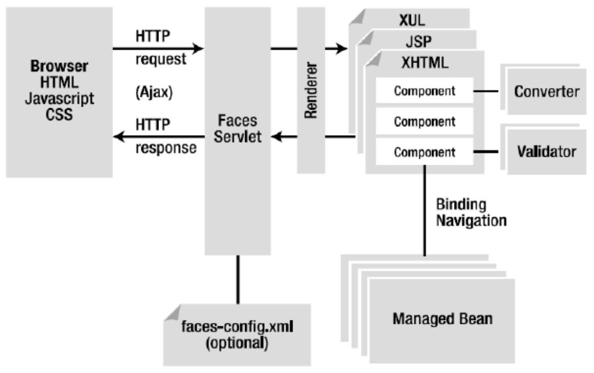


Figure 3. Java Server Faces 2 architecture [7]

As shown in Fig. 3, JSF applications are standard web applications that manage HTTP requests via Faces servlet and produce HTML code.JSF architecture allows for different web languages and pages can be viewed in different devices as well as create pages by using components, events and listeners similarly to as Swing works. For instance, JSF has a set of UI tools, such as buttons, checkboxes, text fields, etc., and allow for an easy integration of third-party components.

The advantages of JSF 2 in comparison to other presentation frameworks (see rest of subsections) are:

- As an official specification published by the Java Community and incorporated in Java EE (currently JSF 2 is part of Java EE 6), all Java EE servers include JSF.
- Incorporation of many features from open source frameworks.
- Use of simple POJOs, such as "managed beans" [6] by just meeting the bean requirements, and also use of annotations.
- Extensive set of APIs and tags for associated clients to create HTML forms in a similar way to developing desktop applications by java Swing.
- Generation of graphics in different HTML formats by using communication protocols different from HTTP.
- Manage complex GUIs in the same component and allow for using more component libraries from third parties than other frameworks.
- Use of jQuery and AJAX through simple tags. AJAX support is less complex.
- Better support and integration into development tools, such as NetBeans and Eclipse.

JSF 2 main disadvantages are:

- Learning curve is slower than the other frameworks, such as Struts).
- Lack of validation of the client tier (i.e Struts has support by using Javascripts.

D. *Java EE Design Patterns*

Design patterns purpose to produce a solution or best practice for a common problem with a given set of functionality that is provided by the environment [8]. Core Java EE pattern [9] catalog includes design patterns for presentation, business and integration/domain tiers. In this subsection we report on and describe a selection of core Java EE patterns for the presentation tier that will be used in the construction of our framework.

1) *MVC 2 (Type II)*

Well-known Model-View-Controller (MVC) is a general purpose pattern for the presentation ties. It separates the modeling of the domain, the presentation, and the actions based on the user input, into three separate classes: The Model class manages the behavior and data of the application domain, responds to requests for information about its state, and responds to instructions to change state; the View class manages the display of information; and the Controller class interprets the user inputs, informing the Model and/or the View to change as appropriate.

Implementation of Type I of MVC showed problems of maintainability due to the Controller logic was embedded into hard-coded JSPs. Type II of MVC overcome this issue by moving the Controller code into a servlet leaving JSP for its natural target to manage the view components.

2) *Intercepting Filter*

The Intercepting Filter [9] is a core Java EE pattern used by Struts 2 [2] to manage the presentation tier's many types of different requests and responses from Web clients, such as test client's authentication and valid session, each test requiring different type of pre- and post-processing.

Usually, this situation is programmatically managed by long series of if-else conditional checks, with any failed check aborting the request, thus making the programming obscure and the code hard to maintain. The Intercepting Filter pattern creates, adds and removes processing components (ie. connected filters), each completing a filtering action. Filters intercept the input requests and the output responses allowing for pre and post-processing in a flexible way without requiring changes in the main code of the request processing.

Figure 4. Intercepting Filter pattern diagram class

3) Service to Worker

Based on the MVC pattern, the Service to Worker pattern overcomes the problem of no centralized control and handling of client's requests coming from different views, and as a result obtains the corresponding presentation model prior to give back control to the view. The view then triggers a dynamic response based on the obtained model. This solution also alleviates the supplicate control code in various views making the code more reusable and easier to maintain, and eventually the system becomes more flexible.

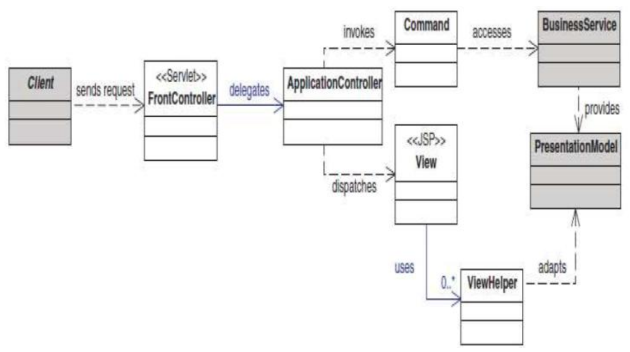


Figure 5. Service to worker pattern class diagram[9]

Service to Worker [9] can be described as the combination of several design patterns [8]: Front Controller, Context Object, View Helper, Application Controller and Command patterns. The Front Controller receives the input request and uses the Context Object pattern to encapsulate the request's state before delegating the request to Application Controller, which uses Command to map logic names to the appropriate command or actions. The command defines the rendering model and then invokes a business service, which returns the requested data with the appropriate View to which passes the workflow control. Finally, the View Helper pattern adapts and transforms the rendenring model to the View.

III. A JAVA EE PRESENTATION FRAMEWORK

In this section we describe our Java EE presentation framework. First, the requirements of the framework are obtained from the related work presented in the previous section. Then the design of our framework is presented and finally the implementation. The framework evaluation is provided in next section.

A. Requirements

In the previous section we have presented three of the most used Java EE frameworks for the presentation tier. From the advantages and disadvantages analyzed we can determine the following list of desirable features that any presentation framework should have making up the requirements of our framework (see Section II for a review):

- In order to separate the view and model, MVC 2 pattern should be considered, where a servlet acts as a controller of the client's requests. This controller will centralize the logics of the next view selected according to the input and output parameters and the results of handling the user's request. To this end, the use of the Application Controller pattern is also needed.
- Automatic data transmission and conversion of the client's request into Actions
- Automatic data validation
- Declarative workflow so that given a user's request the sequence of commands and views is declared independently from the application code.
- Tag libraries are needed to avoid hardcoding JSPs
- Support for internationalization so that the application is shown in the user's preferred language without recode the application
- Easy extensibility in order to add new functionalities/commands that handle the user's request and render the appropriate results as a response to the request.

To sum up, the above requirements determine that our framework will follow most of features of Struts 2 framework oriented to Actions. Next, the framework design will make certain decisions based on design patterns to meet these requirements and overcome some of the mentioned disadvantages of Struts.

B. Framework Design

The design of our framework will extensively use Java EE presentation design patterns. MVC 2 will be used with a front controller whose class is named FrontController (inherited from HttpServlet). In this context, the Application Controller pattern will be used to extract the application logics out of the controller. In addition, the Context Object pattern is used to encapsulate the data from the request with the class named RequestContext, so that FrontController can create a requestContext object from the request's data it already knows and delegate it to the ApplicationControler. Finally, the Service to Worker pattern will be used, for better design of the actions and for improving control centralization and user's request handling (see Section II for a review of design patterns)

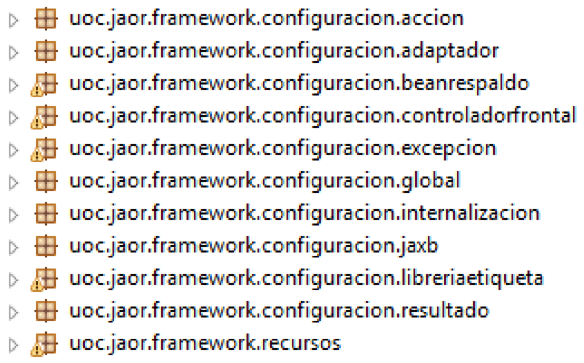
The application's presentation tiers will be made up of actions and forms. Automatic invocation of actions and forms will be realized by API Reflection, thus meeting this requirement of our framework. The invocation and validation of the form attributes will be realized by the filter-based classes that collaborate with the forms but has no pre- and post-execution behavior. Each action will implement the Command pattern, which is used exhaustively in our framework design (e.g. it is applied in the filters and the results). By Command we can requests operations to objects whose content is a priori unknown.

Finally, our framework structure will take full advantage of the Java Architecture for XML Binding [10]. This way, the client will define its presentation tier through XML file, where forms, actions, results and filters will be defined. This file will be validated against the XML scheme. Moreover, a result is an element with attributes that allows for defining the navigation paths (ie., normal, error or ad hoc paths) through XML files. This way we can parameterize the navigation as one of the framework's goals.

To sum up, the presentation tier will be defined in terms of forms and actions, thus in the configuration there will be forms and actions, each realized by a class. In addition, each action will define the results.

C. Framework Implementation

The package structure of the framework is depicted in Fig 6.



```
▷ uoc.jaor.framework.configuracion.accion
▷ uoc.jaor.framework.configuracion.adaptador
▷ uoc.jaor.framework.configuracion.beanrespaldo
▷ uoc.jaor.framework.configuracion.controladorfrontal
▷ uoc.jaor.framework.configuracion.excepcion
▷ uoc.jaor.framework.configuracion.global
▷ uoc.jaor.framework.configuracion.internalizacion
▷ uoc.jaor.framework.configuracion.jaxb
▷ uoc.jaor.framework.configuracion.libreriaetiqueta
▷ uoc.jaor.framework.configuracion.resultado
▷ uoc.jaor.framework.recursos
```

Figure 6. Package structure (names are in Spanish)

The most representative packages in the framework are described next (refer to Fig. 6):

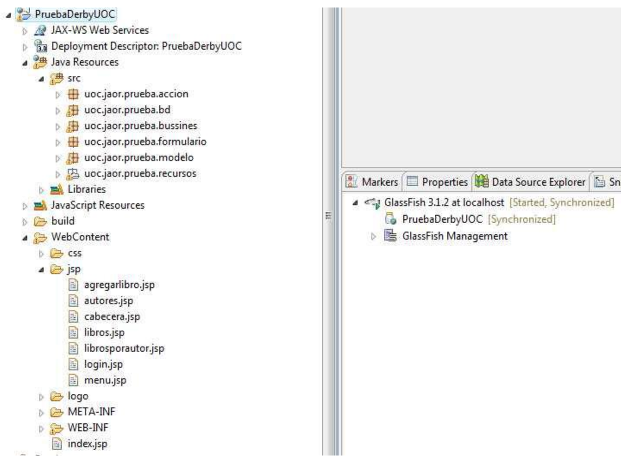
- *accion* includes the interface Action, the RequestContext class that implements Context Object pattern, and the ApplicactionController class that implements the corresponding design pattern. This implementation sets the requirements for the actions that use the framework and how the actions will be handled when the front controller of the framework delegates the execution.
- *beanrespaldo* includes, among other elements, the classes to handle the forms along with the interfaces that all forms and filters on the forms must comply.
- *controladorfrontal* includes the servlet that implements the FrontController pattern.
- *global* includes the Factory class in charge of defining the client's presentation tier from generating through JAXB [9] the objects defined in the client XML.
- *resultado* includes those classes that implement the results by using servlets. A specific interface is defined that all the resultas using this framework must comply.

Finally, the libraries used for the framework development are GlassFish 3.1.2 [3] and JAXB [10]. Eclipse Java EE IDE for Web Developers (Indigo version), with a plugin for GlassFish.

IV. Evaluation

A naive testing application called PruebaDerbyUOC (see Fig. 7) is presented in this section to evaluate our presentation framework whose development has been reported in the previous section. The testing was executed in the server GlassFish [3].

The purpose of the testing was to develop a web application for a book store by using our framework. The application's requirements include authentication (see Fig. 8 and 9), list of authors (see Fig. 10), search books by a specific author (se Fig. 11), search author by name and add new books.



```
PruebaDerbyUOC
  JAX-WS Web Services
  Deployment Descriptor: PruebaDerbyUOC
  Java Resources
    src
      uoc.jaor.prueba.accion
      uoc.jaor.prueba.bd
      uoc.jaor.prueba.bussines
      uoc.jaor.prueba.formulario
      uoc.jaor.prueba.modelo
      uoc.jaor.prueba.recursos
    Libraries
  JavaScript Resources
  build
  WebContent
    css
    jsp
      agregarlibro.jsp
      autores.jsp
      cabecera.jsp
      libros.jsp
      librosporautor.jsp
      login.jsp
      menu.jsp
    logo
    META-INF
    WEB-INF
    index.jsp
```

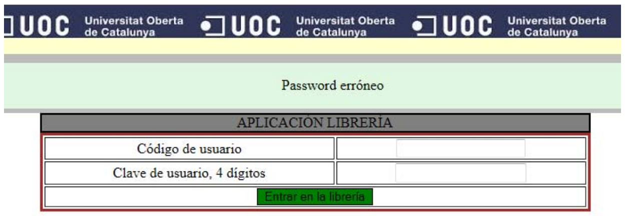
```
GlassFish 3.1.2 at localhost [Started, Synchronized]
  PruebaDerbyUOC [Synchronized]
  GlassFish Management
```

Figure 7. Package structure of the testing (names are in Spanish)

The main packages of the testing application are depicted in Fig. 7:

- *accion* defines the application's actions with the base AccionBase class that implements the Accion interface and the rest of classes the inherits from the base class.
- *business* mainly implements the façade pattern to have a unique point of access to the business tier represented by this package.

- *formulario* defines all the application's forms. The base class for forms is FormularioBase which implements Formulario interface. All forms inherits from the base class, such as FormularioEntrada for login (see Fig. 8), FormularioLibrosDeAutor for searching the books by an author (see Fig. 11).
- *modelo* represents the integration tier where the main entities for users, authors and books are defined.

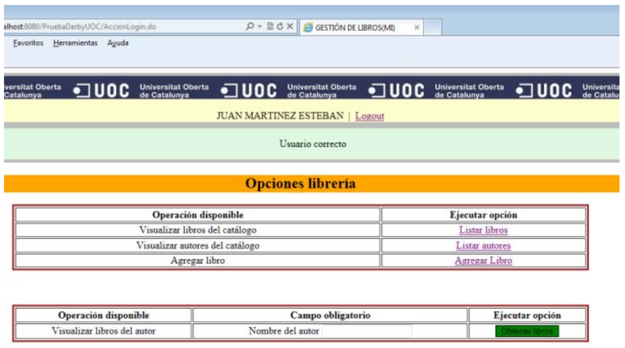


Password erróneo

APLICACIÓN LIBRERÍA

Código de usuario	
Clave de usuario, 4 dígitos	

Figure 8. Authentication screen (with wrong login)



JUAN MARTINEZ ESTEBAN | Logout

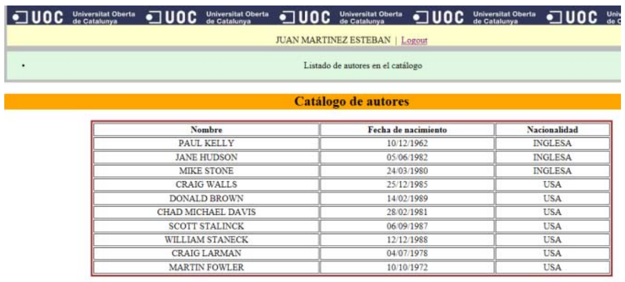
Usuario correcto

Opciones librería

Operación disponible	Ejecutar opción
Visualizar libros del catálogo	Listar libros
Visualizar autores del catálogo	Listar autores
Agregar libro	Agregar Libro

Operación disponible	Campo obligatorio	Ejecutar opción
Visualizar libros del autor	Nombre del autor	Obtener libros

Figure 9. Aplication menu after login



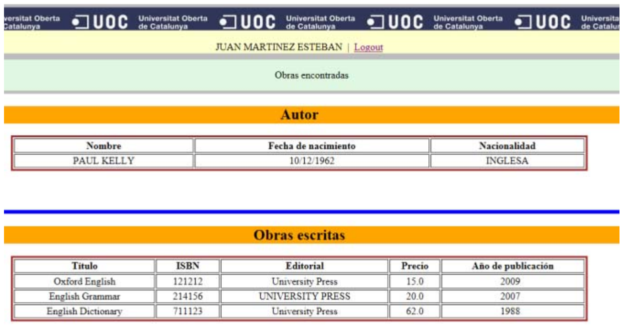
JUAN MARTINEZ ESTEBAN | Logout

Listado de autores en el catálogo

Catálogo de autores

Nombre	Fecha de nacimiento	Nacionalidad
PAUL KELLY	10/12/1962	INGLESA
JANE HUDSON	05/06/1982	INGLESA
MIKE STONE	24/03/1980	INGLESA
CRAIG WALLS	25/12/1985	USA
DONALD BROWN	14/02/1989	USA
CHAD MICHAEL DAVIS	28/02/1981	USA
SCOTT STALINCK	06/09/1987	USA
WILLIAM STANECK	12/12/1988	USA
CRAIG LARMAN	04/07/1978	USA
MARTIN FOWLER	10/10/1972	USA

Figure 10. List of authors



JUAN MARTINEZ ESTEBAN | Logout

Obras encontradas

Autor

Nombre	Fecha de nacimiento	Nacionalidad
PAUL KELLY	10/12/1962	INGLESA

Obras escritas

Titulo	ISBN	Editorial	Precio	Año de publicación
Oxford English	121212	University Press	15.0	2009
English Grammar	214156	UNIVERSITY PRESS	20.0	2007
English Dictionary	711123	University Press	62.0	1988

Figure 11. Books by author

V. CONCLUSIONS AND FURTHER WORK

In this paper we have presented a software framework to develop Java EE application that supports the presentation tier. First, an extensive survey of existing Java EE presentation frameworks has been conducted along with the core Java EE design patterns used in our framework. The analysis results of both frameworks and design patterns set the requirements for developing our framework, which is reported from all the phases of the life cycle. Finally the framework is tested by developing a naïve application for evaluation purposes.

From the evaluation we conclude that our framework simplified and made far easier the development of the testing application by following the Struts 2 Actions, which provides a clear model and implementation of many Java EE design patterns, for instance MVC-2 and ApplicationController. Our framework, however, achieves better performance by implementing certain design patterns that work in conjunction with the Struts patterns. For instance the Service to Worker and ContextObject patterns help provide a centralized control point that allows for parameterized navigation through the result, attributes' invocation and validation become easier. Also our framework support filters to intercept the input requests and the output responses allowing for pre- and post-processing in a flexible way. Finally, the framework provides internationalization and a tag library to support annotation.

We plan to evaluate further our framework by using it for developing full web applications in various domains in order to receive exhaustive feedback prior to develop a new iteration of the framework. We also study to implement more Java EE patterns into the framework to make the development work easier and with more reusable and maintainable applications.

REFERENCES

[1] A. Gupta, « Java EE 7 Essentials», O'Reilly, 2013.

[2] D. Brown, M. D. Chad, and S. Scott. «Struts 2 in action». Dreamtech Press, 2008.

[3] A. Goncalves, «Beginning Java™ EE 6 Platform with GlassFish™ 3», Apress, 2010.

[4] C, Walls, «Spring in Action» Manning, 2008

[5] B. Kurniawan, «Struts 2 Design and Programming: A Tutorial», Brainy Software, 2008.

[6] M. Hall. «JSF 2.0: Introduction and Overview», Retrieved from the Web as of March 2014 at: http://www.coreservlets.com/JSF-Tutorial/jsf2/

[7] D. Geary, C. Horstmann, «Core Java Server Faces», Third Edition, , Prentice Hall, 2010.

[8] C. Larman, «UML y Patrones», Prentice Hall, 2004.

[9] D. Alur, J. Crupi, D. Malks, «Core J2EE Patterns: Best Practices and Design Strategies», Prentice Hall, 2003.

[10] Dennis Sosnoski. «XML and Java technologies: Data binding, Part 2: Performance». Oracle, 2007