Study design patterns in Java Platform Enterprise Edition version 6 for web application development

Acosta, Maricruz

North Technical University -UTN, Ibarra, Imbabura

jmary_86@hotmail.com

Abstract. *In the ongoing quest to improve the quality and design of software, techniques and technologies have evolved. Various strategies have been developed, several improvements are the result of previous strategies. Design patterns are probably the most versatile strategies consist reuse solutions, not only of source code, but provide solutions to recurring problems in software development.*

Importantly JEE design patterns, collected a set of best practices that have been developed in recent years for the development of web systems. The presence of design patterns leads to standard solutions, easily understandable and maintainable by developers.

The main focus of this work is the knowledge of design patterns JEE 6, and that despite the benefits it generates implementation, are often treated as J2EE patterns, blocking the functionality of the design patterns and platform.

Keywords

Patterns, JEE6, Design, Applications, Web.

Resumen. *En la búsqueda constante por mejorar la calidad y el diseño del software, técnicas y tecnologías han evolucionado. Se han desarrollado diversas estrategias y habilidades, varias son producto de mejoras realizadas sobre estrategias anteriores. Los patrones de diseño probablemente son las estrategias más versátiles, consisten en la reutilización de soluciones, no solamente de código fuente, sino que constituyen soluciones integrales a problemas repetitivos en el desarrollo del software.*

Es importante señalar que los patrones de diseño JEE, recopilan un conjunto de buenas prácticas que se han venido desarrollando en los últimos años para el desarrollo de sistemas web. La presencia de patrones de diseño conduce a soluciones estándares, fácilmente comprensibles y mantenibles por parte de los desarrolladores.

El enfoque principal de este trabajo es el conocimiento de los patrones de diseño JEE 6, ya que a pesar de los beneficios que su implementación genera, son tratados a menudo como patrones J2EE, obstruyendo la funcionalidad de los patrones de diseño y de la plataforma en sí.

Palabras Claves

Patrones, JEE6, Diseño, Aplicaciones, Web.

1. Introduction

Since the 90s, when the Internet began to spread worldwide, begins the search tactics and skills so as to take advantage of new technologies and provide solutions that suit the environment. This propagation, made the engineers and software developers will gradually leaving behind some architectures that, though they remain in force, tend to disappear over the years. The problem is not that the technologies change, but the ability that the developer has to adapt to them. Precisely this has been one of the principal obstacles in the development of software for the Web.

Design patterns originated as civil architecture concepts by architect Christopher Alexander [1]. Then the book is published Core J2EE Patterns [2], focused on best practices and design strategies for enterprise applications. Ultimately, the designs use of patterns or not, should be translated into code, so that the book is published Real World Java EE Patterns - Rethinking Best Practices [3], which is a rethinking of design patterns are now used in applications JEE. Due to the great reception that took its release, the book Real World Java EE Night Hacks [4], is published as a complement to the previous book, it reflects the use of patterns in web applications.

Although the existence and extensive use of design patterns, there is still a degree of ignorance about the existence and benefits of using design patterns for Java

Enterprise Edition 6 Web application development. A considerable number of developers using J2EE patterns JEE applications, which makes it impossible the appropriate use of the platform and the improvements it offers to the design and production of systems, increasing in a way, the development and delivery time thereof.

This work is intended as a guide that allows choose design patterns Java EE 6 platform best suited for web application deployment, taking into account the main features each has and the benefits they offer.

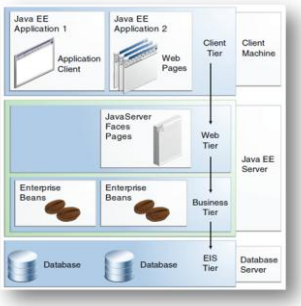

Figure. 1. Multilayer Applications[1]

2. Theorical Frame

Several questions may arise when starting the development of a Web project. These projects often delimited critical delivery times and must be executed by decreasing expenses. They should be light systems in consumption of resources scalable and able to exchange information. Maintainability is also critical in this type of projects.

This paper focuses on the best practices of Web application development, given a certain level of ignorance on the subject, we will analyze each design pattern Java EE platform version 6, therefore, the realization of this study seeks to the reader acquires a certain level of knowledge of design patterns and possible implementation seeks to highlight, the benefits of using design patterns JEE 6, and intended as a guide for choosing appropriate patterns.

Java Enterprise Edition there since 1998 as an initiative of a group of engineers from Sun Microsystems, and in more than 10 years of development has become one of the most robust architectures for building information systems. JEE was a pioneer in the development of design patterns, publishing the March 8th 2001, its own catalog of patterns that was very popular, and subsequently diffuses BluePrints patterns and incorporates vital concepts in the systems as scalability and performance.

Opted Into the analysis of JEE design patterns, because the primary sources of Web patterns used this implementation platform, although in practice, the patterns are sufficiently independent of the platform.

2.1 Layer model

The Java EE platform adopts an approach distributed multi-tier applications for enterprise applications, where application logic is divided into components according to function. Using a programming logic developed in levels or layers, which allows separate or encapsulate application elements in defined parts, to simplify the development and to clarify the responsibility of each of the components of an application.

2.2 Features and services of JEE

The fundamental objective of the Java EE 6 platform is to simplify the development by providing a common basis for different types of components of the Java EE platform. Developers benefit from the productivity improvements with more annotations and less XML configuration. It has new features like: profiles, contexts and dependency injection, new features for EJBs, Servlets and JSF components.

Among the services that JEE provided and that can be used in any application server that follow this standard include: Java Transaction API (JTA), Java Persistence API (JPA), Java Message Service (JMS), JavaMail, Java EE Connector Architecture (JCA), Java EE Connector Architecture (JCA), among others.

2.3 What is a Desing Pattern

The design patterns can be defined as predefined schemas or set of strategies applicable in various situations, so that the analyst makes sure that the design you are using has certain qualities that give quality.

The Design Patterns book proposes the following definition for pattern design: "The design patterns are descriptions of communicating objects and classes that are adapted to solve a general design problem in a particular context" (Erich Gamma, 1994). Therefore, the patterns are centered on the micro-structure of the applications, that is, classes and objects.

2.4 Why are they useful Design Patterns

The main objective of design patterns is to capture best practices to improve the quality of system design, identifying objects that support useful roles in a specific context, encapsulating the complexity and making it more

[1] Figure obtained from the JEE 6 Tutorial (OracleCorporation, pág. 41).

flexible. The benefits resulting from the use of a pattern can be measured in many ways, for example:

Contribute to reuse design, identifying key aspects of the structure of a design that can be applied in different situations. The design reuse is important because it reduces development efforts and maintenance, improve safety, efficiency and consistency of designs.

Improve flexibility, modularity and extensibility, internal factors and closely related to the quality perceived by the user, in addition, increase the design vocabulary of developers, helping to design from higher levels of abstraction.

2.5 What are JEE design patterns

With the appearance of JEE version 6, a new design pattern catalog was released, providing the benefits of this platform and solutions for problems typically encountered by architects and designers in the process of application development.

"JEE design patterns are the best solutions to help developers design and build applications on the JEE platform" (Bien, Real World Java EE Patterns – Rethinking Best Practices, 2009).

2.6 Design of web applications with design patterns

The design of web applications based on JEE patterns is intrinsically organized around the use of several elements: a front controller, dispatcher, composite views, static views (JSPs) and helpers of dynamic views formed with JavaBeans. This may be reflected in figure 2, which illustrates a design based on these mechanisms.

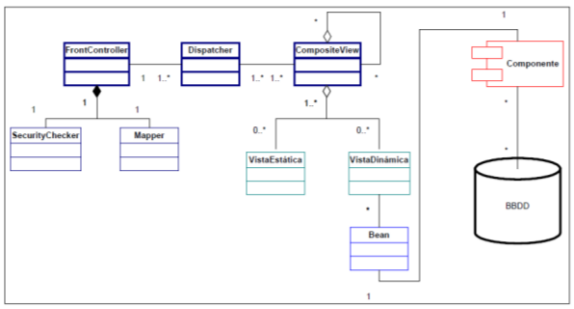

Figure. 2. Design based on JEE patterns[2]

[2] Figure taken the manual Web Application Design (Díaz, 2003).

2.7 Design patterns JEE 6 catalog

Most of the Java EE 6 projects continue using J2EE patterns but this platform was intrusive, the separation of technical infrastructure and business logic needed the mandatory introduction of patterns, which were mostly solutions for deficiencies of J2EE.

The functionality of this new version of the Java Enterprise platform, demand a rethinking of the design patterns used in J2EE and the elimination of superfluous code required to implement applications on that platform. Taking into account the previous point and following the logic of the GoF design patterns, a series of specific patterns of Web world is divided into three layers: Presentation, Business and Integration.

a) Patterns of presentation layer. This layer includes Servlets, JSPs and other interface elements that generate server-side user.

The design patterns of this layer are part of the J2EE catalog but can be used in JEE, however we must not forget the benefits this platform version 6 offers, so that the use of patterns in this layer should be adequately justified. In this study, the following patterns are mentioned:

Intercepting Filter. This pattern handles various types of requests that require specific processing, is applied especially in session validation processes. It relates to the Front Controller and Decorator patterns.

Front Controller. The Front Controller pattern accepts all requests from a client and routed to the appropriate handlers. It is related to View Helper, Intercepting Filter, Service to Worker and Dispatcher View patterns.

View Helper. Charge of to encapsulate the access logic databases. It relates to the Business Delegate, Front Controller, Service to Worker and Dispatcher View patterns.

Composite View. Used when required common subviews, such as headers and footers, multiple views, that can occur in different places within each page layout. It relates to the View Helper pattern.

Service to Worker. This pattern is similar to the MVC architectural pattern, uses the Front Controller patterns to the Controller and View Helper for the view with a dispatcher component.

Dispatcher View. This pattern is similar to Service to Worker, but the controller does not perform actions on the helper.

a) Patterns of the business layer. This layer exposes the logic required for the user through the interface, to interact with the functionalities of the application.

The Java EE platform defines the use of EJB business components to abstract the business logic of other general problems like concurrency applications, transactions, persistence and security. In this study, the following patterns are mentioned:

Service Facade. It is the improved version of the Application Service pattern. It is a stateless session bean with a local business interface that should be provided remotely only if you will use outside the Java Virtual Machine (JVM), and will not injected into a Servlet, backing bean, or another web component. It relates to the patterns: Application Service, Session Facade, Gateway, Service, Data Access Object, Data Transfer Object.

Service. Is the new name Session Facade pattern, is procedural and performs activities or threads. Its main objective is to make building business logic reusable and easier to maintain. A service is coordinated by Service Facades or Gateways, call DAOs and can produce DTOs.

Persistent Domain Object. Persistent Domain Object pattern (PDO) is a persistent entity. Relationships and status changes are automatically synchronized with the persistence at the end of the transaction. Interacts with the Composite Entity, Domain Store, Business Object, Data Transfer Object design patterns.

Gateway. A Gateway provides an entry point to the root of the PDOs. It relates to the patterns: Composite Entity, Domain Store, Business Object, Data Transfer Object, Service Facade, Service.

Fluid Logic. The changing algorithms often require re-compilation and even redeploy the entire application. Fluid Logic pattern is a specific Service.

Patterns removed. The patterns which are detailed below are removed from the principal applications of Java EE 6, but could be used for special purposes or for migrating J2EE to Java EE 6.

Service Locator. It was a pattern binding in J2EE applications. The main reasons for withdrawal are:

Dependency Injection is available in most Java EE components. The JNDI lookups are no longer necessary to access other session beans or resources.

The creation of an initial interface is optional, and consequently the creation of local and remote interfaces. Code complexity infrastructure was greatly reduced by the EJB specification 3.0. You should use the Dependency Injection whenever possible and only in exceptional cases, the implementation of a generic Service Locator.

Composite Entity. Represents a scheme of objects. The main reasons for withdrawal are:

The persistence CMP 2.1 does not support perfectly relationships. CMP entities had start interfaces which were used similarly to EntityManager.

With the advent of JPA, the implementation of the relationships became naturally.

JPA entities are domain objects that are persistent. You can apply any pattern desired without any overload. The Composite Entity pattern was degraded in JEE to a design pattern Composite GoF catalog.

Value Object Assembler. It was a pattern dedicated to merge, transform or extract data from different data sources. The main reasons for withdrawal are:

The EntityManager implements part of the original intention of the Value Object Assembler: it is capable of returning a submodel from a scheme of interconnected entities.

The creation of sub-models and the conversion of attached entities within Transfer Objects, is the responsibility of Service Facade or Service. There is no need for a specific component or pattern for implementing the conversion. In most cases, it could unravel the use of dedicated Transfer Objects and spend separate entities and bonded layers or levels.

Business Delegate. It was used to hide implementation details of the business and to reduce the coupling between it and the presentation layer. The main reasons for withdrawal are:

No longer necessary to use this pattern to separate business logic from the presentation because the business interfaces can be injected directly into the majority of presentation components.

With EJB 3.0 business interface is identical to the external interface of the Business Delegate. The Business Delegate pattern was using Service Locator to find the initial interface and internally created local or remote interface. In EJB 3.0, the source interfaces are optional, business interfaces can be retrieved directly from JNDI.

Domain Store. This pattern was used to transparently persist an object model and is independent of the model. His retirement is because the Entity Manager in JEE is considered a pattern Domain Store.

Value List Handler. This pattern was used to provide iteration between data and client. The reasons for withdrawal are:

Since the introduction of JPA, entities can be easily separated without additional effort. The implementation of the Value List Handler is not relevant, because it is the application of the Iterator pattern[3].

[3] Iterator, lets users make routes on composite objects regardless of the implementation of these.

The Value List Handler became strategy of the Paginator pattern.

a) Patterns integration layer. This layer gathers connection components with other systems, connectors with services, courier services and others. In this layer patterns are numbered as follows:

Data Access Object. It is used to separate the business logic from data access. In a Java EE 6, is reduced to an interface and a class with specific EJB annotations.

The DAO session bean is annotated whit @ TransactionAttribute, so that all DAO methods can be invoked in an existing transaction. It must be invoked from a component of the business logic and not directly from the display.

The use of a dedicated DAO is optional, can be replaced with EntityManager injected to a Service. A DAO is used for Service or Service Facade and may use Generic JCA owners to access the legacy resources.

Transfer Object y Data Transfer Object. The primary responsibility of these patterns is to optimize data transfer. The patterns Data Access Object, Service Facade and Service can consume and produce DTOs.

In J2EE, its use was recommended to hide the specifics of the container managed persistence, at present, JPA transferring data between layers, especially in a single JVM. The TOs are structurally similar to JPA entities, then, each change driven by a request arrives to the JPA entities.

Legacy Pojo Integration. There is a need to integrate POJOs and developed a Java EE application, and access them easily and without friction.

The POJO inherited is deployed as an EJB 3 to easily participate in transactions and be managed by the container. It could also be injected into a existing session bean.

Generic JCA. A JCA connector can access each resource incompatible EJB is portable, standardized, easily accessible and managed by the application server. It is a set of interfaces API y SPI connection oriented.

Is composed of two interfaces, four classes, and an XML file. It is completely transparent to the developer. The connectors can be accessed by the Service and Service Facade patterns.

Asynchronous Resource Integrator. The pattern Asynchronous Resource Integrator is a Service Activator with a scope and expanded responsibilities, although its main objective remains incompatible resource integration. The asynchronous invocation of synchronous services covered by the EJB 3.1 specification.

This pattern can work directly with DAOs and Services may be invoked by sending JMS messages.

b) Patterns of infrastructure and utilities. There are difficult to classify patterns that can be used in all layers, or not all are relevant in a general context and can be omitted from the description of the architecture. The patterns of infrastructure and utilities that can be used in the development of the applications are:

Service Starter. Su uso debe satisfacer las siguientes necesidades: Se necesita una forma portable para iniciar los servicios con impaciencia y la solución debe proporcionar enlaces extensibles mediante los cuales, beans de sesión sin estado existentes pueden ser inicializados.

The pattern Starter Service acts as a wrapper, to initialize a session bean should be injected reference and call the desired method, this will make injects container and initialize the bean.

Invokes Service, Service Facade or Gateway to produce its initialization, it is technically identical to the Singleton pattern.

Singleton. Their responsibility is to provide shared state and concurrency management. Can access to Service Facades existing, services and resources for serch information needed and caching afterwards.

Bean Locator. The Bean Locator Pattern is responsible for locating EJBs and building global JNDI names. Returns beans local or remote with stateful, stateless and Singleton, and should only be used in classes where Dependency Injection does not work or is not available.

Dependency Injection can be considered as a Bean Locator 2.0, because it uses a generic version of this pattern, configured with metadata derived from conventions, class files, annotations and XML. The J2EE Service Locator pattern is related to the Bean Locator, its implementation was limited due to the lack of global JNDI names and annotations.

Thread Tracker. The use of this utility, you must resolve the following requirements: The solution should be portable across servers. The expansion of control capabilities must be clearly separated from business code. Additional control functionality should be easy to remove before deployment in production.

It can connect to a given class using annotations or XML. Help to find potential bottlenecks and slow methods. For more flexible control and search of access points could be deployed Services with ThreadTracker.

Dependency Injection Extender. The Dependency Injection Extender (DIE) is practical for small fragments reuse of existing functionality.

The interceptor plays the most important, is the bridge between EJBs and legacy components. It is reusable across services and Service Facades.

Payload Estractor. Before the method is invoked, receives and detects the actual message type. Depending on the type, throw the message, extract its contents, and invokes the consumption method using reflection.

From the point of view of implementation, the Payload Extractor pattern is similar to Dependency Injection Extender. Forwards the message content to services, which are injected into the message-driven bean.

Resource Binder. The resources folder is responsible for: The initialization of resources needed before any invocation of business logic. Life cycle management of custom resource. Register the resource in the JNDI tree of the application server. Resource cleanup and cancellation of registration at the time of shutting down the server.

The resources exposed by this pattern are consumed in session beans and therefore the patterns Services, Service Facades and DAOs.

Context Holder. It is provided by the container or Java SE. In both cases, it is started before the first business method invocation of a Service Facade or Gateway. This can be done with an interceptor, dynamic proxy or servlet filter that is the injector.

Is used by the Service Façade pattern and their interceptors for data transport to services and DAO. Because the Context Holder uses interceptors, this pattern is similar to patterns Thread Tracker, Payload Extractor, and Dependency Injection Extender.

2.8 Applicative

With the aim of apply learning in the course of the study, it was decided the development of a small hotel system using as development tools NetBeans IDE version 7.2.1, Postgresql 9.0 and GlassFish Application Server 3.1, and for the documentation RUP methodology.

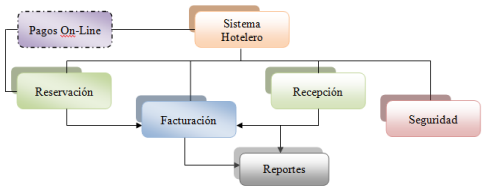

Figure. 3. Modules hotel system [4]

Reservation Module. This module allows: Check the availability of cabins. Entering customer data. Inquiries reservations. Register and cancel reservations. Providing payment services online.

Receiving Module. This module allows: Perform guest registration with or without reservation. Modification of reservation and receiving data. Check availability of cabins. Inquiries reservations.

Billing module. This module is responsible for issuing the corresponding receipt.

Security Module. Creating and deleting users. System access management.

Reports Module. Generate the listing of reservations and room occupancy.

For the service of on-line payments considering the use of PayPal to be the system most widely used online payment market.

Figure. 4. *User Interface: Project Login* [5]

3. Results

Web applications are often constructed in which each JSP page manages security services, content retrieval, and navigation. This creates a model with a high maintenance costs due to the existence of duplicate code, usually, by the use of the technique of copy and paste.

It can significantly improve the quality of these applications centralizing and encapsulating some mechanisms, making the application much more maintainable, simple, and clean. To achieve these objectives there is nothing better than the condensed experience of many years of development and design: JEE design patterns.

The JEE platform offers various design possibilities. However, the use of the MVC architectural

[4] Source: Author

[5] Source: Author

pattern and JEE Design Patterns 6 give scalability to applications, facilitating future expansions, migrations and changes in general. This could be verified in the course of developing the applicative where they were used principally Data Access Object, Services and paginator patterns.

4. Conclusions

The study of design patterns generates a range of implementation possibilities for all kinds of applications.

A design pattern is the collection of given solutions to a problem encountered during development.

The primary objective of the design patterns is to simplify the development, providing applications scalability, flexibility and reusability.

The use of design patterns JEE 6 in conjunction with the functionalities that this platform provides, generates robust applications with very little effort.

There is more benefit to the develop a distributed application layered using JEE design patterns, it facilitates the maintenance of the application.

A system based on design patterns and correctly implemented, it will require major changes when there are changes in the requirements as to the create a system using patterns, the basic structure is flexible and reusable software as they have been taken into account in the design stage, the possible changes that may arise in the future for the application.

There are large numbers of design patterns and strategies to implement them, the choice will depend on the complexity of the system and developer skills.

The design patterns JEE 6 platform evolved so that they are more powerful and provide greater benefits than J2EE patterns.

Although the functionality of J2EE patterns is minimal compared to the benefits of the design patterns JEE 6 and the platform itself may be used yet this will depend on the judgment of the developer.

5. Recommendations

Continue researching, studying and applying the different design patterns to obtain the benefits of these.

Is recommended the inclusion of design patterns in the development of Software by need and solve problems present at system being developed, but not add without much knowledge of them.

No design patterns applied indiscriminately, as this may complicate the design and implementation of a system. The patterns are intended to create reusable systems, portable, scalable, readable code, but achieving these objectives can lead to greater problems in design and implementation, if not applied in the right way.

Is recommended to read the documentation of the pattern, especially the section consequences, because it will give an idea of the result you get when you use the pattern in a given context.

The implementation strategies expressed in this document and forming a part of the original description of each pattern, are not a rule to follow. Its use is recommended if the application development warrants.

It is recommended, when you want to include a design pattern in a system, use a developer with experience in handling patterns, demand for its proper use have some practice in this regard.

Promote the development of web and enterprise applications using design patterns.

The use of Java Platform Enterprise Edition, for all the features and functionality offered.Se recomienda el uso de la plataforma Java Enterprise Edition, por todas las características y funcionalidades que ofrece.

Acknowledgements

I want to extend a sincere thank you to North Technical University staff especially Professor of Engineering Degree in Computer Systems Engineers represented by Jorge Caraguay, Mauricio Rea and José Luis Rodríguez, Thesis Director, who with his advice and expertise made be the culmination of my work.

Bibliographical References

[1] Alexander, C., Ishikawa, S., Silvertein, M., Jacobson, M., Fiksdahl-King, I., & Angel, S. (1977). *A Pattern Language: Towns/Builder/Construction.* New York: Oxford University Press.

[2] Deepak Alur, J. C. (2003). *Core J2EE Patterns: Best Practices and Design Strategies.* California: Sun Microsystems, Prentice Hall.

[3] Bien, A. (2009). *Real World Java EE Patterns – Rethinking Best Practices.* Kentucky.

[4] Bien, A. (2011). *Real World Java EE Night Hacks — Dissecting the Business Tier.*

[5] Óscar Belmonte, C. G. (2012). Desarrollo de Proyectos Informáticos con Tecnología Java. Universitat Jaume I.

[6] Miguel Abarca C., G. D. (2009). *Desarrollo Básico de Aplicaciones en la Plataforma J2EE*. Obtenido de http://xxito.files.wordpress.com/2008/05/manualj2ee.pdf

[7] OracleCorporation. (s.f.). *The Java EE 6 Tutorial.* Obtenido de http://docs.oracle.com/javaee/6/tutorial/doc/javaeetutorial6.pdf

[8] Gracia, L. M. (s.f.). *Un poco de Java.* Obtenido de http://unpocodejava.wordpress.com/2011/01/10/tecnicas-de-bloqueo-sobre-base-de-datos-bloqueo-pesimista-y-bloqueo-optimista/

[9] Torrijos, R. L. (s.f.). *Programación en Castellano*. Obtenido de http://www.programacion.net/articulo/catalogo_de_patrones_de_dise_j2ee_i_-_capa_de_presentacion_240

[10] Sun Microsystems, Inc. (2002). *ORACLE*. Obtenido de http://www.oracle.com/technetwork/java/catalog-137601.html

[11] Universidad de Alicante. (13 de 10 de 2011). *Servicio de Informática*. Recuperado el 21 de 09 de 2012, de http://si.ua.es/es/documentacion/asp-net-mvc-3/2-dia/razor/helpers.html

[12] William Crawford, J. K. (2003). *J2EE Design Patterns* (1st Edition ed.). California: O' Reilly.

[13] Beck, K., Cunningham, W., Inc, A. C., & Inc, T. (1987). *Using Pattern Languages for Object-Oriented Programs.* Orlando.

[14] Erich Gamma, R. H. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software*. Massachusetts: Addison Wesley.

[15] Martin Fowler, D. R. (2002). *Patterns of Enterprise Application Architecture.* Massachusetts: Addison Wesley.

[16] Riehle, D., & Züllighoven, H. (1996). *Understanding and using patterns in software development.* John Wiley & Sons, Inc.

[17] Gabriel, R. P. (1996). *Patterns of Software: Tales from the Software Community*. Oxford University Press, Inc.

[18] Coplien, J. (1996). *Software Patterns.* SIGS Books.

[19] Díaz, M. (2003). *MOISESDANIEL.COM*. http://www.moisesdaniel.com/es/wri/disaplicj2ee.html

[20] Mejía, D. P. (s.f.). Recuperado el 4 de Noviembre de 2012, de http://delta.cs.cinvestav.mx/~pmejia/softeng/Patrones.ppt

Reference source Wikipedia

[21] http://es.wikipedia.org/wiki/ASP.NET

[22] http://es.wikipedia.org/wiki/Java_EE

[23] http://es.wikipedia.org/wiki/PHP

[24] http://es.wikipedia.org/wiki/Sun_Microsystems

[25] http://es.wikipedia.org/wiki/Java_Community_Process

[26] http://es.wikipedia.org/wiki/SDK

[27] http://es.wikipedia.org/wiki/JavaBean

[28] http://es.wikipedia.org/wiki/Plain_Old_Java_Object

[29] http://es.wikipedia.org/wiki/Java_Servlet

[30] http://es.wikipedia.org/wiki/Java_Archive

[31] http://es.wikipedia.org/wiki/WAR_(archivo)

[32] http://es.wikipedia.org/wiki/Enterprise_JavaBeans

Other sources

[33] http://www.programacion.com/articulo/catalogo_de_patrones_de_diseno_j2ee__i__capa_de_presentacion_240.

[34] http://cursoj2ee.blogspot.com/2009/02/controlador-frontal.html

[35] http://www.corej2eepatterns.com/Patterns2ndEd

[36] http://migranitodejava.blogspot.com/2011/05/patrones-de-diseno-de-gof.html

About the Author

Author-Maricruz ACOSTA Graduated from the School of Computer Systems Engineering from the Technical University of the North.