Análisis de la adecuación de lenguajes de programación Web a un desarrollo basado en patrones de diseño J2EE de alto nivel

UNIVERSIDAD COMPLUTENSE
MADRID

PROYECTO FIN DE MÁSTER

Autor: **Óscar Mauricio Morales Franco**

Director: **Antonio Navarro Martín**

Máster en Investigación en Informática

Facultad de Informática

Universidad Complutense de Madrid

Curso académico: **2008/2009**

Análisis de la adecuación de lenguajes de programación Web a un desarrollo basado en patrones de diseño J2EE de alto nivel

PROYECTO FIN DE MÁSTER

Autor: **Óscar Mauricio Morales Franco**

Director: **Antonio Navarro Martín**

Máster en Investigación en Informática

Facultad de Informática

Universidad Complutense de Madrid

Curso académico: **2008/2009**

El abajo firmante, matriculado en el Máster en Investigación en Informática de la Facultad de Informática, autoriza a la Universidad Complutense de Madrid (UCM) a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a su autor el presente Trabajo Fin de Máster: "Análisis de la adecuación de lenguajes de programación Web a un desarrollo basado en patrones de diseño J2EE de alto nivel", realizado durante el curso académico 2008-2009 bajo la dirección de Antonio Navarro Martín en el Departamento de Ingeniería del Software e Inteligencia Artificial, y a la Biblioteca de la UCM a depositarlo en el Archivo Institucional E-Prints Complutense con el objeto de incrementar la difusión, uso e impacto del trabajo en Internet y garantizar su preservación y acceso a largo plazo.

Madrid, a 14 de septiembre de 2009

Óscar Mauricio Morales Franco

Agradecimientos

En la vida, sería muy difícil lograr metas sin estar rodeados de personas tan valiosas como aquellas que brindan siempre confianza, ideas positivas y apoyo incondicional.

La realización de este trabajo es posible gracias al constante apoyo de un gran equipo de trabajo llamado *familia.*

También, un agradecimiento especial a Antonio, mi director, por sus sabias recomendaciones y excelente gestión.

Abstract

Technologies and techniques for software construction have evolved quickly in the last ten years. New technologies in the Web Applications field have surged and now the all compete together. Software architects and developers keep building applications using old practices, others, do not know the alternatives and how interesting can they be.

This work is focused in the comparison of J2EE alternatives for building enterprise systems. ASP.NET/C#, PHP and PHP/Zend Framework will be the main characters in this work's discussions. The analysis made in this job, is based in a common and universal vocabulary such as J2EE design patterns. Each technology's capacity will be determined by the level of support it brings into the each high-level pattern implementation.

It is important to notice that J2EE design patterns, group together a set of good practices that have been under development during recent years for Web Systems' construction. Even in practice, many Web Applications are built using an ad-hoc architecture, design patterns lead to standard solutions, easy to comprehend and maintainable for developers.

The chose of certain Web Development Programming Language depends of certain aspects such as the learning curve of a specific one, or the hardware capabilities of the different servers that made the whole system. Nonetheless, on advanced development entities, these characteristics can be non-determinant. The question is raised, whether or not all available programming languages are capable of implementing solutions in accordance with high-level Web design patterns. This works tries to answer this question.

Also, in this work, J2EE design patterns main sources are analyzed (known and used as a guide for software developers and architects around the world). During this analysis, a mapping between different patterns ontologies defined by each source will be created. This way, a mechanism that eases the understanding of these patterns in general is given, allowing an open discussion throughout the document.

Keywords

Design patterns, software engineering, web applications, best practices, enterprise applications, PHP, ASP.NET, J2EE

Resumen

Las tecnologías y las técnicas para construcción de software han evolucionado rápidamente en los últimos diez años. En el ámbito de aplicaciones web han surgido nuevas tecnologías que ahora compiten entre sí. Los arquitectos y desarrolladores de software continúan desarrollando aplicaciones con prácticas antiguas, y otros, no conocen las alternativas y lo interesantes que pueden ser.

Este trabajo está enfocado en comparar alternativas a J2EE para la construcción de sistemas empresariales. ASP.NET/C#, PHP y PHP/Zend Framework serán los protagonistas de las discusiones en este trabajo. El análisis que aquí se desarrolla está basado en un vocabulario común y universal como es el de los patrones de diseño J2EE. La capacidad de cada tecnología será determinada por el nivel de soporte que brinde para la implementación de cada patrón de alto nivel.

Es importante darse cuenta que los patrones de diseño J2EE recopilan un conjunto de buenas prácticas que se han venido desarrollando en los últimos años para el desarrollo de sistemas web. Aunque en la práctica muchas aplicaciones web se construyen con una arquitectura ad-hoc, la presencia de patrones de diseño conduce a soluciones estándares, fácilmente comprensibles y mantenibles por parte de los desarrolladores.

En la práctica, la elección de un determinado lenguaje de programación web viene dada por cuestiones tan importantes como puede ser la curva de aprendizaje de un determinado lenguaje, o las posibilidades hardware de los distintos servidores del sistema a construir. Sin embargo, en entidades de desarrollo lo suficientemente avanzadas, estas características pueden no ser determinantes. Surge por tanto la pregunta de si todos los lenguajes disponibles son capaces de implementar soluciones diseñadas acorde a patrones de diseño web de alto nivel. Precisamente este trabajo pretende responder esta pregunta.

Además, en este trabajo, se analizan las principales fuentes de patrones de diseño J2EE (conocidos y usados como guía por desarrolladores y arquitectos alrededor del mundo). Durante este análisis se creará un mapping entre las distintas ontologías de patrones definidas por cada fuente. De esta forma, se proporciona un mecanismo que facilita la comprensión de estos patrones en general, y que permite una libre discusión a lo largo del documento.

Keywords

Patrones de diseño, ingeniería de software, aplicaciones web, buenas prácticas, aplicaciones empresariales, PHP, ASP.NET, J2EE

Tabla de contenido

1. Introducción

1.1 Motivación

Desde los años 90s, cuando Internet empieza a propagarse mundialmente, se inicia la búsqueda de estrategias para el aprovechamiento de las nuevas tecnologías que de allí se desprenden. A raíz de esta propagación, ingenieros de software y desarrolladores poco a poco han ido dejando de lado algunas arquitecturas que, aunque siguen siendo vigentes, tienden a desaparecer con el paso de los años. El problema no está en que las tecnologías cambien, sino en la capacidad que tengamos para adaptarnos a ellas. Precisamente, éste ha sido uno de los más grandes obstáculos en el desarrollo de software para la Web.

Hace algunos años se empezó a introducir el concepto de Ingeniería Web [Yogesh 99], disciplina que nace como la combinación de varios campos de conocimiento, tales como ciencias de computación, ingeniería de software, seguridad y diseño gráfico. Ya en el principio, se detectó que el desarrollo de aplicaciones Web era más complejo que el de software convencional, y requería de más profesionales de diferentes áreas.

A diferencia de hace diez años, ahora contamos con diferentes alternativas para la implementación de un proyecto de software Web, como diferentes lenguajes de programación, arquitecturas y técnicas. Además, disponemos de diversos patrones de diseño web de alto nivel. Sin embargo, a la hora de llevar a cabo un proyecto Web, surgen dudas acerca del tipo de tecnología a usar, ya que cada tecnología tiene su alcance, sus ventajas, sus desventajas y su elección puede diferenciar el éxito o fracaso de un proyecto. Precisamente este trabajo intenta ser una guía que permita elegir la tecnología de implementación más adecuada en función de la naturaleza del proyecto, y de los patrones incluidos en su arquitectura.

Además, el trabajo proporciona un mecanismo de relación entre los patrones de diseño web provenientes de distintas fuentes, de esta forma se facilita la comprensión y utilización de los mismos.

1.2 Antecedentes

Los patrones fueron originados como conceptos de arquitectura civil alrededor del año 1977 por el arquitecto Christopher Alexander [Alexander 77]. Luego en el contexto de OOPSLA 87, Kent Beck y Ward Cunningham [Beck 87] empezaron a aplicar el concepto de patrones en la informática.

En 1991 Erich Gamma y Richard Helm trabajaron en su primer catálogo, pero no se populariza hasta el año 1994, tras la publicación del libro maestro de patrones de diseño *"Design Patterns"* [Gamma 94]. Hoy en día existen diferentes tipos de patrones que buscan solucionar la mayoría de problemas que se presentan en el desarrollo de software. Algunos han sido diseñados específicamente para algunas arquitecturas, pero muchos de ellos se pueden implementar en diferentes lenguajes.

En 2002 Martin Fowler et al. publicaron el libro *"Patterns of Enterprise Application Architecture"* [Fowler 02]. En este libro se divide la aplicación empresarial en diferentes capas y se enfatiza en los patrones relacionados con mappings entre objetos y bases de datos relacionales. El libro recogía una serie de buenas prácticas utilizadas por Martin Fowler y otros arquitectos software en el desarrollo de diversos sistemas software.

En 2003 Deepak Alur, John Crupi y Dan Malks publicaron la segunda edición de su libro, publicado en 2001, enfocado a las buenas prácticas y estrategias de diseño para aplicaciones empresariales, *"Core J2EE Patterns"* [Alur 03]. En este trabajo se estudian 21 patrones de diseño con ejemplos de código y se ilustran los puntos clave para la refactorización de este tipo de aplicaciones utilizando patrones de diseño. El libro recogía una serie de buenas prácticas utilizadas en los desarrollos llevados a cabo por Sun Microsystems.

También en 2003, William Crawford y Jonathan Kaplan hicieron público su trabajo *"J2EE Design Patterns"* [Crawford 03]. Este libro está enfocado en el estudio de patrones de diseño para aplicaciones empresariales y abarca temas críticos como extensibilidad/mantenimiento, escalabilidad, modelado de datos, transacciones e interoperabilidad.

En última instancia, los diseños, utilicen patrones o no, tienen que ser traducidos a código. En la actualidad, existen al menos tres grandes tecnologías asociadas a la programación Web: ASP.NET [ASPNET 09], J2EE [Java-J2EE 09] y PHP [PHP 09].

ASP.NET es el *framework* de desarrollo que Microsoft inauguró en el año 2002. Es la evolución de ASP (*Active Server Pages*) y permite construir dinámicamente aplicaciones Web y *Web Services* [Papazoglou 07], con soporte además para el marco .NET [Net 09].

ASP.NET ayuda al rendimiento de las aplicaciones – a diferencia de otros lenguajes de *scripting* – compilando partes del código del lado del servidor en uno o varios archivos DLL (*Dynamic-Link Library*) en el servidor Web.

Los servidores para alojar este tipo de aplicaciones deben estar basados en sistemas operativos Microsoft, y esto se puede convertir en una gran desventaja de esta tecnología. También existen iniciativas *Open Source* como *Mono* [Mono-Project 09] que permiten portar las aplicaciones a otras plataformas, aunque éstas no cuentan con el soporte Microsoft.

Java Enterprise Edition (J2EE) existe desde el año 1998 como iniciativa de un grupo de ingenieros de *Sun Microsystems*, y en más de 10 años de desarrollo se ha convertido en una de las arquitecturas más robustas para la construcción de sistemas de información. J2EE ha sido pionero en la elaboración de patrones de diseño de todo tipo [Alur 03, Crawford 03] y en incorporar conceptos vitales como escalabilidad y rendimiento de grandes sistemas.

Sin embargo, los entornos de ejecución de J2EE son mucho más exigentes que los requeridos por otras arquitecturas y la curva de aprendizaje se conoce como más costosa.

PHP es un lenguaje orientado a scripts que fue expuesto al público en 1994. Aunque inició como un simple *parser* para reconocer macros y cambiar dinámicamente el contenido de páginas Web, a lo largo de estos años se ha convertido en uno de los lenguajes más utilizados a nivel mundial para la construcción de sitios dinámicos y aplicaciones Web.

PHP ha sido un lenguaje polémico, tiene ventajas como ser ampliamente soportado en empresas de *web hosting*, tener una curva de aprendizaje suave y ser potente y flexible para el desarrollo. Esta flexibilidad ha ocasionado problemas, porque a diferencia de otros lenguajes, PHP se presta para convertirse en un caos si no se acude a buenas prácticas en el desarrollo de un proyecto.

Como se puede apreciar, la diversidad de patrones de diseño de alto nivel para aplicaciones web, y la variedad de tecnologías disponibles hacen bastante difícil disponer de mecanismos objetivos para elegir un determinado lenguaje u otro. Este trabajo analiza los lenguajes/tecnologías mencionados anteriormente. En particular se estudiarán las capacidades de estos lenguajes para afrontar un desarrollo basado en patrones para distintos tipos de aplicaciones Web. Se ha optado por analizar patrones J2EE, porque las principales fuentes de patrones Web utilizan esta plataforma de implementación, aunque en la práctica, los patrones son bastante independientes de la plataforma.

1.3 Ámbito de la contribución

Pueden surgir muchas dudas en el momento en que se va a iniciar un proyecto de software Web. Estos proyectos suelen delimitarse con tiempos críticos de entrega y deben ejecutarse disminuyendo gastos. Además, deben ser sistemas ligeros en consumo de recursos, escalables y capaces de intercambiar

información. Por último, la mantenibilidad también es fundamental en este tipo de proyectos. Es por esto que una de las preguntas que más me ha motivado a realizar esta investigación ha sido:

> *Supuesto que se va a diseñar una aplicación Web utilizando patrones arquitectónicos,*
>
> *¿cuál es el marco de programación más adecuado teniendo en cuenta la complejidad de la aplicación a desarrollar?*

Hace muchos años, y quizás cuando no habían tantas alternativas, esta respuesta era más sencilla. Se sabía que simplemente Perl [Perl 09] o ASP podrían servir para crear una página de contenido dinámico. Ahora la respuesta es más complicada, se conocen tres tecnologías como ASP.NET, J2EE y PHP como adecuadas para el desarrollo de aplicaciones Web. Por tanto, cabe preguntarse,

> *¿Cuál es la diferencia y cuál es más conveniente elegir?*

Este trabajo está enfocado en las buenas prácticas de desarrollo de software Web y para esto se analizará el alcance de cada tecnología para el desarrollo de sistemas diseñados acorde a patrones arquitectónicos de diseño Web. De esta forma, se busca un nivel de profundidad especial en los patrones de diseño necesarios y su posible implementación en cada tecnología.

Para esto, se realiza y aporta una clasificación de las aplicaciones en diferentes tipos. Como base se tomarán las aplicaciones más simples que no requieren persistencia y que de hecho pueden implementarse en los tres lenguajes. Cada vez se le agregará un grado de complejidad a la aplicación y así se llegará a tres tipos para los cuales se definirán las necesidades en cuanto a patrones de alto nivel para su implementación. Finalmente, se analizará la adecuación de cada lenguaje para una implementación natural de dichos patrones de diseño.

El análisis será realizado tomando como base J2EE que es la arquitectura más robusta y punto de referencia en cuanto a patrones de diseño se trata. Los textos base principales para la elaboración del trabajo son *Core J2EE Patterns* [Alur 03] y *J2EE Design Patterns* [Crawford 03]. Al disponer de dos fuentes de patrones de diseño, este trabajo realizará una equivalencia entre los patrones expuestos por los autores de ambos libros.

Por tanto, las aportaciones del trabajo son varias. En primer lugar, al realizar una equivalencia entre dos fuentes de patrones heterogéneas, se proporciona un vocabulario común a distintas comunidades de ingenieros software. Disponer de un vocabulario común en diseño es fundamental en ingeniería del software [Booch 05, Booch 07].

Esta equivalencia define en última instancia un mapping entre ontologías. Estos mappings son muy útiles a la hora de generar relaciones semánticas entre conceptos heterogéneos. Por ejemplo, el trabajo de [Santacruz-Valencia 08] los utiliza para hacer una comparación entre requisitos y competencias de objetos de aprendizaje, permitiendo de esta forma un ensamblaje semántico, en vez de meramente sintáctico.

Por último, este trabajo es una guía para la elección de lenguajes de programación Web. Así, en base a la complejidad de la aplicación que se desee construir, y por tanto, en base a los patrones de diseño Web requeridos, este trabajo ilustra qué lenguaje de programación Web es el más adecuado. Disponer de este tipo de guía es fundamental en proyectos Web actuales. Así, por ejemplo, algunos proyectos de investigación como el proyecto *Integración de Plataformas y Servicios en el Campus Virtual* (IPS-CV) (TIN2008-06708-C03-01/TSI) o el proyecto *Arquitecturas Avanzadas en Campus Virtuales* (AACV) (TIN2009-14317-C03-01) pueden encontrar en este trabajo un aporte fundamental. Los campus virtuales de grandes universidades evolucionan hacia complejas aplicaciones web que necesitan resolver problemas de integración de plataformas y servicios, mantenibilidad, interoperabilidad y autoría de contenidos y calidad. Los proyectos de esta envergadura requieren una investigación centrada en la integración de aplicaciones web de dimensión industrial según diversas arquitecturas (multicapa, de integración y orientadas a servicios).

Es importante darse cuenta que las distintas plataformas de e-learning (p.e. *Blackboard* [Blackboard 09], *Moodle* [Moodle 09], *Sakai* [Sakai 09]) están escritas en distintos lenguajes (p.e. *Java* y *PHP*). El disponer

de una guía que en cada caso facilite la elección de un lenguaje es fundamental en este tipo de proyectos de integración.

1.4 Esquema del trabajo

En este *primer capítulo* se presenta una motivación para el lector, un resumen de antecedentes relacionados con el estudio de patrones de diseño y el ámbito de la contribución.

En el *segundo capítulo* se encuentra información relacionada con patrones de diseño actuales y los que serán analizados en este trabajo. También, información acerca de las tecnologías analizadas. Todo esto enmarcado por el *estado del arte*.

En el *tercer capítulo* se introduce el concepto de *mapping entre ontologías* y se define un mapping semántico entre las ontologías inducidas. Este mapping es útil para desarrollar el trabajo dado que establece un vocabulario común y una relación entre patrones de las diferentes fuentes.

En el *cuarto capítulo* se define una *jerarquía de aplicaciones*. Luego se establece una relación entre cada tipo de aplicación con los patrones de diseño que corresponden, con respecto a la complejidad.

En el *quinto capítulo* se analizan los patrones de diseño relacionados con las *aplicaciones con persistencia*. En el *sexto capítulo*, los relacionados con las *aplicaciones con persistencia y concurrencia*, y en el *séptimo capítulo*, los relacionados con las *aplicaciones con persistencia, concurrencia y mensajes*.

Finalmente, en el *octavo capítulo* se describen las *conclusiones y trabajo futuro* que resaltan las aportaciones de este trabajo.

2. Estado del arte

2.1 Patrones de diseño J2EE

La construcción de aplicaciones empresariales aumenta en complejidad en la misma medida que los sistemas se hacen más grandes. Tras la llegada de nuevas tecnologías como *Java 2 Enterprise Edition*, diferentes autores han investigado sobre las arquitecturas más ventajosas, y han expuesto sus conclusiones en diferentes textos. Estos libros de alguna forma se han convertido en la base para futuras investigaciones y para implementaciones reales a nivel mundial.

Los autores de *Core J2EE Patterns* [Alur 03] y de *J2EE Design Patterns* [Crawford 03] han elaborado un par de libros útiles para arquitectos de software y desarrolladores. Todos los patrones y estrategias de diseño recomendadas se encuentran acompañadas por código fuente en Java. Esto facilita su comprensión.

Core J2EE Patterns y *J2EE Design Patterns* están basados en patrones de diseño para *Java 2 Enterprise Edition.* Sin embargo ambos textos son diferentes en cuanto a la cantidad de patrones que presentan, el nivel de implementación, e incluso los nombres que se usan. Este trabajo analiza ambas fuentes de patrones, y aporta además un mapping entre las ontologías inducidas por ambas fuentes de patrones. Dicha ontología se presenta en este trabajo como una tabla de relación entre los distintos patrones.

Core J2EE Patterns es elegido como base para el análisis de la mayoría de patrones porque tiene más profundidad en detalles de implementación y porque los patrones definidos provienen de la experiencia práctica de los ingenieros de *Sun Microsystems*. Los patrones que no se encuentran definidos en este libro se extraen entonces de *J2EE Design Patterns* [Crawford 03].

En el año 2002, Martin Fowler et al. publican el libro *Patterns of Enterprise Application Architecture* [Fowler 02]. Este es otro excelente texto de patrones de diseño de alto nivel. En el desarrollo del trabajo se realizarán referencias a este libro. No se incluye este libro en el mapping entre ontologías definido en este documento, porque el propio libro *Core J2EE Patterns* explícitamente relaciona los patrones allí expuestos con los patrones de Martin Fowler et al.

Además, es importante darse cuenta como los diferentes autores introducen conceptos que pueden variar sobre los mismos patrones de diseño. En *Core J2EE Patterns* se encuentra una distribución de cinco capas de aplicación. Esta distribución no se mantiene homogénea entre las distintas fuentes de patrones. Precisamente, como resultado del mapping propuesto en este trabajo, estas capas se relacionan con las expuestas en el libro *J2EE Design Patterns* [Crawford 03].

Entre los diferentes textos disponibles hoy en día, se encuentran guías para diseño con patrones (universales, independientes del lenguaje) y textos especializados en implementar patrones de diseño en tecnologías específicas (J2EE, PHP, ASP.NET y otros).

A la fecha no hay investigaciones disponibles con las características de este trabajo. Por tanto, establecer una diferencia entre lenguajes basada en la capacidad de implementación de patrones será una herramienta útil para arquitectos y desarrolladores.

2.2 Patrones

2.2.1 Introducción

Antes de especificar los patrones que serán analizados en este trabajo es importante introducir un poco el concepto de éstos y resaltar su importancia.

Para definir un patrón se puede partir de la definición que Christopher Alexander [Alexander 77] hizo en su libro:

> *"Un patrón describe un problema que ocurre una y otra vez en nuestro entorno, así como la solución a ese problema de tal modo que se puede aplicar esta solución un millón de veces, sin hacer lo mismo dos veces".*

Un patrón de diseño se puede ver como una solución repetitiva y experta a problemas recurrentes, es decir, estrategias que se conocen desde un principio y se sabe qué tipo de problemas van a solucionar.

Una característica fundamental de los patrones de diseño es que surgen de la experiencia de la comunidad de programadores. De esta forma, los patrones son soluciones que se han probado en proyectos reales, y hay consenso sobre su utilidad.

Existen diferentes definiciones por cada autor, pero los patrones tienen algunas características identificadas [Alur 03] como:

- Los patrones son identificados a partir de la experiencia humana.
- Los patrones previenen reinventar la rueda.
- Los patrones existen en diferentes niveles de abstracción.
- Los patrones experimentan mejoras continuas y son elementos reutilizables.
- Los patrones hablan de diseño y buenas prácticas y se pueden usar varios a la vez para resolver grandes problemas.

Al igual que otras disciplinas y ciencias, la ingeniería de software depende en gran parte del uso de patrones de diseño para garantizar rendimiento de los sistemas y eficiencia en el ciclo de desarrollo. Una de las metas de un arquitecto de software o desarrollador es saber abordar los problemas con las mejores herramientas de diseño y para esto es importante conocer los patrones y las técnicas.

2.2.2 ¿Porqué es Importante Usar los Patrones de Diseño en aplicaciones Web?

Los patrones de diseño Web son importantes para garantizar de la mejor forma que una aplicación empresarial sea [Crawford 03]:

- **Extensible**. Que se determina en la forma en que la aplicación puede crecer para incorporar nuevos requisitos de los usuarios, no sólo desde el punto de vista operativo sino también técnico, que no sea difícil para los programadores extender funcionalidades ni deban repetir código fuente. Para esto se utilizan algunas técnicas conocidas como: desacoplamiento, centralización y reusabilidad.

- **Escalable**. Que se determina en la forma en que la aplicación podrá soportar mayor cantidad de usuarios y peticiones en un ambiente exigente. La escalabilidad está relacionada directamente con el rendimiento.

- **Fiable.** Que se determina no sólo por la calidad del software, sino porque el hardware y la red se comporten de la forma esperada. Cada entrada al software debe siempre retornar la misma salida.

- **Puntual en tiempos del proyecto.** La meta en cualquier proyecto de software es realizar una entrega en el tiempo proyectado, finalmente es lo que más le interesa al usuario final sin importarle a él qué hay detrás de todo.

2.3 Clasificación de Patrones

Diferentes autores han sugerido numerosas categorías para clasificar los patrones de software, así como muchos otros han introducido nuevos patrones.

Cuando se publica el libro "Design Patterns" [Gamma 94] se introducen 23 patrones de diseño y hoy en día se conocen cerca de 100 [Metsker 06]. Toda la documentación que se encuentra sobre patrones está desarrollada sobre los diferentes niveles de abstracción, hay patrones para arquitectura, diseño, análisis y programación.

Para la elaboración de este trabajo se tomarán como referencia los patrones definidos en "Core J2EE Patterns" [Alur 03] y "J2EE Design Patterns" [Crawford 03] para mensajería y algunos de concurrencia. En algunas discusiones se mencionarán también patrones del libro "Patterns of Enterprise Application Architecture" [Fowler 02].

A continuación se puede ver el esquema con el que los autores clasifican y discuten los patrones en sus trabajos. En el siguiente capítulo se realiza un análisis y se crea un mapping entre las ontologías.

2.3.1 Clasificación según Core J2EE Patterns

Alur et al. [Alur 03] clasifican los patrones en este libro utilizando tres capas principales, pero en total se definen cinco capas de aplicación.

La *capa de cliente* agrupa dispositivos y clientes que acceden a la aplicación, como un navegador web, un móvil, un applet, etc. La *capa de presentación* incluye *Servlets*, *JSP* y otros elementos que generan interfaz de usuario del lado del servidor. La *capa de negocio* contiene la lógica más importante de la aplicación, gestión de transacciones, servicios y otros (*EJBs, Business Objects)*. La *capa de integración* agrupa componentes de conexión con otros sistemas, conectores con servicios, servicios de mensajería y otros. La *capa de recursos* contiene bases de datos y sistemas externos.

En este libro los patrones de diseño se clasifican en las capas: presentación, negocio e integración. Además de la discusión de cada patrón de diseño, los autores agregan estrategias de implementación que incluyen detalles de bajo nivel. Estas estrategias son muy útiles para los desarrolladores por el énfasis técnico que tienen [Alur 03].

2.3.2 Clasificación según J2EE Design Patterns

En este libro los patrones de diseño se introducen de acuerdo al contexto. El objetivo de los autores es explicar un conjunto de patrones que permiten construir otros [Crawford 03]. Todos los patrones son presentados en un gran contexto de aplicaciones J2EE. En un anexo se ofrece un catalogo de patrones convencional.

Se usan cinco capas para distribuir la aplicación empresarial. La *capa de presentación del lado del cliente* contiene navegadores web, dispositivos móviles e incluso otras aplicaciones. La *capa de presentación del lado del servidor* integra los elementos necesarios para construir el interfaz gráfico que requiere la capa de presentación del lado del cliente. La *capa de lógica del negocio* incluye todos esos métodos que contienen el procesamiento real de la aplicación. El *modelo de dominio* está compuesto por el modelado de datos de la aplicación, con el que la lógica del negocio se comunica constantemente. La *capa de integración empresarial* permite conectar la aplicación con otros sistemas.

La discusión de patrones inicia en la capa de presentación del servidor y se discuten estrategias de rendimiento y gestión de recursos para sistemas escalables. Luego se introducen patrones relacionados con la capa de negocio. Para comunicación eficiente y escalable entre capas se definen otros patrones de integración. Después se discuten patrones de persistencia y concurrencia y finalmente se introducen todos los patrones de mensajería que están relacionados con la capa de integración empresarial. Crawford y Kaplan también introducen el concepto de *antipatrones* al final del trabajo, al igual que Alur et al.

2.3.3 Clasificación según Patterns of Enterprise Application Architecture (PoEAA)

En PoEAA [Fowler 02] se centra la discusión de patrones en tres capas principales. La *capa de presentación* es la lógica que define cómo se debe gestionar la interacción entre el usuario y la aplicación. Fowler et al. se refiere en este libro a un navegador web basado en HTML y un *cliente enriquecido* (Windows/Swing/UI pesada).

La *capa de datos* (o fuente de datos) contiene lógica encargada de la comunicación con otros sistemas de importancia para la aplicación. Estos sistemas pueden ser otras aplicaciones, bases de datos, gestores de transacciones, sistemas de mensajería u otros paquetes.

La *capa de dominio*, que también es llamada lógica del negocio es una de las piezas más críticas del sistema. En esta capa se encuentran los cálculos y validaciones sobre datos de entrada que llegan desde la capa de presentación y el procesamiento de información almacenada en fuentes de datos.

La discusión de cada patrón se basa en un esquema muy útil que permite identificar con facilidad cada patrón. Una introducción de máximo tres líneas con un dibujo, y una sección que discute la solución. Luego se presenta una sección que especifica en qué casos se puede usar el patrón y al final una sección que contiene enlaces a otras fuentes para profundizar más, y un ejemplo en código (Java o C#).

2.4 Patrones que se analizan

Como ya se ha comentado anteriormente, este trabajo utiliza las capas propuestas por Alur et al. para organizar los patrones. A continuación se describen los patrones analizados en este trabajo relacionados con las capas de presentación, negocio e integración. Nótese que al asignar patrones con origen en el trabajo de Crawford y Kaplan, en capas propuestas por Alur et al., esta sección avanza los resultados del mapping entre ontologías de la siguiente sección. Más adelante serán relacionados con cada una de las aplicaciones de la jerarquía.

2.4.1 Capa de Presentación

- *Intercepting Filter*: facilita el procesamiento que se hace antes y después de una solicitud [Alur 03].
- *Front Controller*: provee un controlador central, dependiente del protocolo de comunicación, para gestionar las solicitudes [Alur 03].
- *Context Object*: provee un mecanismo independiente de protocolo para compartir el estado de la aplicación [Alur 03].
- *Application Controller*: centraliza y modulariza la administración de acciones y vistas, de forma independiente al protocolo de comunicación [Alur 03].
- *View Helper*: previene la sobre-especialización de las vistas separando ayudantes para que sean reutilizados [Alur 03].
- *Composite View*: crea una vista agregada de subcomponentes atómicos [Alur 03].
- *Service to Worker*: combina un componente despachador con patrones *Front Controller* y *View Helper* [Alur 03].
- *Dispatcher View*: combina un componente despachador con patrones *Front Controller* y *View Helper* posponiendo varias actividades al procesamiento de vistas [Alur 03].

2.4.2 Capa de Negocio

- *Transfer Object*: lleva datos de una capa a otra, ocultando la representación interna de los datos en cada capa [Alur 03].
- *Transfer Object Assembler*: reúne datos de diferentes fuentes en un objeto [Alur 03].
- *Business Delegate*: encapsula el acceso y uso de un servicio del negocio remoto [Alur 03].
- *Service Locator*: encapsula el acceso de servicios y componentes remotos [Alur 03].
- *Session Facade*: encapsula componentes de la capa de negocio y los expone en servicios únicos a los clientes [Alur 03].
- *Application Service*: centraliza y agrega comportamientos para proveer una capa de servicio uniforme [Alur 03].
- *Business Object*: separa datos del negocio y lógica usando un modelo de objetos [Alur 03].
- *Composite Entity*: representa agregaciones de *Business Objects* [Alur 03].
- *Value List Handler*: gestiona las búsquedas, guarda en caché los resultados y provee capacidades de examinar y seleccionar registros de los resultados [Alur 03].

2.4.3 Capa de Integración

- *DAO*: separa el código para acceder a los datos (al mecanismo de persistencia) del código que procesa los datos [Alur 03].
- *Service Activator*: recibe mensajes e invoca el procesamiento asincrónicamente [Alur 03].
- *Domain Store*: provee un mecanismo de persistencia transparente para objetos del negocio. Incluye gestión de concurrencia, carga perezosa de objetos del negocio y gestión de transacciones [Alur 03].
- *Web Service Broker*: expone uno o más servicios usando XML y protocolos Web [Alur 03].

Concurrencia

- *Optimistic Concurrency*: para probabilidades bajas de colisión, se utiliza un sistema de versión para detectar alteraciones y advertir sobre problemas para que el mismo usuario corrija los datos o reintente la petición [Crawford 03].
- *Pessimistic Concurrency*: para probabilidades altas de colisión, se bloquean recursos completos hasta que el usuario termine la tarea [Crawford 03].
- *Version Number*: similar a la concurrencia optimista pero asocia cambios de estado a algunas partes de la aplicación, en especial a los *Data Access Objects* (DAOs) [Crawford 03].

Mensajes

- *Point-to-Point Distribution:* comunicación entre dos diferentes actores [Crawford 03].
- *Publish-Subscribe:* facilita la comunicación con múltiples destinatarios y facilita mantenibilidad y escalabilidad. [Crawford 03]

- *Malformed Message Channel*: enrutamiento de mensajes incorrectos para ser analizados posteriormente por administradores [Crawford 03].

- *Message Selector:* permite gestionar múltiples tipos de mensajes en una misma cola [Crawford 03].

- *Competing Consumers*: permite procesar diferentes mensajes en paralelo [Crawford 03].

- *Event-Driven Consumer*: despacha mensajes a los clientes tan pronto como sea posible [Crawford 03].

- *Message Façade*: oculta lógica del negocio detrás de una interfaz estándar que pueda ser accedida asincrónicamente.

- *Message Handler:* permite cambiar la forma en que se reciben los mensajes sin cambiar código relacionado con procesamiento [Crawford 03].

- *Polling Consumer*: permite a una aplicación participar en intercambio de mensajes sin necesidad de una conexión continua con el servidor de mensajes [Crawford 03].

- *Content Aggregator*: permite a un mismo manejador procesar diferentes mensajes que de múltiples fuentes que tienen diferentes formatos pero el mismo contenido [Crawford 03].

- *Pipes and Filters*: permite definir procesamiento de mensajes a través de múltiples consumidores creando un flujo de datos flexible y con posibilidad de reutilización [Crawford 03].

- *Content – Based Routing*: provee un punto único de recepción de mensajes asociados con un caso particular. Así, los mensajes podrán ser redirigidos a los diferentes gestores [Crawford 03].

- *Control Bus*: provee una interfaz centralizada para administración asíncrona de múltiples sistemas [Crawford 03]

2.4.4 Tabla de Análisis y Calificación

Las tecnologías se analizarán de acuerdo a su capacidad de implementación de cada uno de los patrones de diseño que hacen parte de cada tipo de aplicación. Cada tecnología obtendrá una calificación distribuida en *"solución"* y *"complejidad"*.

La *solución* indica el grado en que la tecnología de acuerdo a sus herramientas integradas o librerías puede resolver el problema del patrón de diseño. Se plantean entonces tres niveles de solución, como se indica en la Tabla 2.1.

Solución	Descripción
Buena	La tecnología incluye una implementación del patrón de diseño u ofrece mecanismos para una implementación rápida.
Media	La tecnología soporta la implementación del patrón de diseño.
Mala	La tecnología no soporta la implementación del patrón de diseño.

Tabla 2.1. **Niveles de solución de cada tecnología.**

La *complejidad* indica el esfuerzo requerido para implementar el patrón de diseño. Esta calificación se realizará de acuerdo a la capacidad de *solución* de la tecnología, y la cantidad de código que se requiera escribir. En la Tabla 2.2 se describen.

Complejidad	Descripción
Mayor	Se requiere escribir mucho código para solucionar el problema.
Media	Se requiere escribir código pero el lenguaje o el marco poseen herramientas que facilitan el trabajo.
Menor	El marco incluye una implementación integrada o hay que escribir muy poco código.

Tabla 2.2. Niveles de complejidad de cada tecnología.

2.5 Tecnologías y lenguajes

Es importante también mencionar las características principales de las tecnologías que serán analizadas en este trabajo.

2.5.1 ASP.NET/C#

ASP.NET es un Framework para desarrollo Web que se basa en el Framework .NET de Microsoft. Este Framework reúne las tecnologías necesarias para construcción de aplicaciones de escritorio para Windows, aplicaciones Web, servicios Web y más en un solo paquete [Darie 08].

C# es uno de los lenguajes de programación soportados por el Framework, está basado en programación orientada a objetos y en su construcción ha tenido influencia de C++ y Java. Este lenguaje ha sido creado con el objetivo de ser simple, moderno y de propósito general. Desde Julio de 2000 fue incorporado en .NET y la versión más reciente del lenguaje es 3.0, se integra con .NET Framework 3.5. Actualmente se encuentra en desarrollo la versión 4.0 de C#.

Para estudiar las posibles implementaciones de patrones de diseño con C# y ASP.NET es necesario primero conocer un poco sobre la tecnología.

Características del marco ASP.NET

- ASP.NET permite escribir código del lado del servidor usando diferentes tipos de lenguajes. El Framework .NET actualmente soporta alrededor de 40 lenguajes y muchos de éstos pueden ser usados para construir aplicaciones ASP.NET. C# es una de las elecciones más populares.
- Los scripts de ASP.NET son compilados en la primera ejecución, *Just In Time* (JIT).
- ASP.NET tiene acceso completo a las funcionalidades del Framework .NET incluyendo soporte para XML, servicios Web, interacción con bases de datos, correo, etc.
- ASP.NET trae consigo una estructura que separa el código que contiene la lógica de las páginas HTML.
- ASP.NET es un Framework que ya incluye diferentes controles y librerías disponibles para el desarrollador.
- Para construir ágilmente aplicaciones empresariales se requiere usar Visual Studio, ya sea en sus versiones libres (*Express*) o de pago.
- Se ejecuta sobre el servidor de Internet de Microsoft IIS.
- Compilación en tiempo de ejecución (primer hit) JIT.
- Conectividad con bases de datos (ODBC).
- Gestión de transacciones para bases de datos y servicios Web con MTS.
- Mensajes con MSMQ.
- Gestión de concurrencia a nivel de base de datos.
- Soporte para gestión de ficheros vía FTP, SFTP.
- Componentes SMTP y POP3 para envío y recepciones de mensajes.
- Acceso a servicios de directorio a través de ADAM.
- Sistema propio de caché.

Características del lenguaje C#

- Se usa sintaxis de C++
- C# permite sobrecarga de operadores.
- En la declaración no se puede distinguir entre herencia y una implementación de una interfaz, se declara de la misma forma.
- Se debe declarar específicamente que un método puede ser sobre-escrito y que un método sobre-escribe algún otro.
- C# usualmente no hace uso de punteros, se puede únicamente aumentar o disminuir una variable como si fuese un puntero a memoria dentro de un ámbito inseguro.
- C# tiene tipos de datos enumerados.
- C# no permite listas variables de argumentos, se debe definir un método para cada cantidad y tipo de argumentos.
- Se permite el uso de métodos *Indexer* que permiten diseñar clases y estructuras que pueden ser indexadas como arreglos.
- Se pueden declarar funciones anónimas en el ámbito de un método.
- C# permite usar *Reflection* para llamar métodos y crear clases dinámicamente en tiempo de ejecución.
- Soporte para manejo de excepciones.
- Soporte para acceso a objetos remotos.

Requisitos para el uso

Los requisitos para usar esta tecnología se pueden dividir en dos, el entorno para el desarrollador y el entorno de ejecución. A continuación se describen de acuerdo a las sugerencias de Microsoft.

Entorno para el desarrollador

El desarrollador de ASP.NET/C# usa *Visual Studio* o *Visual Web Developer Express* de Microsoft. Este entorno solo funciona bajo algunas versiones de *Microsoft Windows*.

El entorno de desarrollo de *Microsoft* es una potente herramienta que facilita las tareas de programación. Esta es una ventaja cuando se compara con otras tecnologías. El software se puede construir más rápido.

Entorno de ejecución

Para la ejecución de una aplicación desarrollada en ASP.NET se requiere de un servidor con alguno de los siguientes sistemas operativos:

- Microsoft Windows 2000 Professional and Server (SP2 recomendado)
- Microsoft Windows XP Professional
- Microsoft Windows Server 2003

Además para la ejecución se requiere el servidor Web *Internet Information Server* de Microsoft (IIS). Este servidor no es instalado por defecto en el sistema operativo. Se requiere también el Framework .NET.

2.5.2 PHP

PHP es un lenguaje orientado a *scripts* que nace en el año 1994. En sus inicios fue creado para agregar dinamismo de contenido a las simples páginas web de la época. Poco a poco ha empezado a ser usado por una gran comunidad de programadores alrededor del mundo. PHP tiene influencias de *C* y *Perl* en su sintaxis.

Como resultado de su rápida evolución, la comunidad de desarrolladores y empresas como *Zend Technologies Ltd.* [Zend-Web 09] aportaron en la transformación de éste lenguaje en uno más amplio y con soporte para hacer aplicaciones web más exigentes.

A diferencia del marco ASP.NET de *Microsoft*, los *scripts* siempre son interpretados (por el intérprete PHP) y no se generan ficheros compilados en código intermedio para futuras lecturas. Las aplicaciones construidas con este lenguaje pueden ejecutarse en cualquier plataforma. Hay soporte nativo para los diferentes sistemas operativos.

A fecha de hoy, la versión estable de PHP es 5.3.0. En este momento está en construcción la versión 6 que promete ser mucho más empresarial.

Sobre el lenguaje PHP

- Se ejecuta en cualquier plataforma sobre diferentes servidores web (Apache, IIS, otros).
- Orientado a Scripts y no hay ficheros compilados.
- Programación orientada a objetos.
- Soporte al manejo de excepciones.
- Permite listas variables de argumentos.
- Los objetos poseen *métodos mágicos* que facilitan la creación de comportamientos dinámicos en tiempo de ejecución.
- Conectividad con bases de datos.
- Gestión de transacciones para bases de datos.
- Soporte para gestión de ficheros vía FTP, SFTP.
- Servicio de mensajes (SAM).
- Librerías para interacciones con servidores de correo para envío y recepciones de mensajes.
- Acceso a servicios de directorio LDAP.
- Soporte nativo para Memcached como sistema de caché.
- No tiene soporte nativo para acceso a objetos remotos.

Requisitos para el uso

Este lenguaje no obliga a poseer un entorno de desarrollador. Sin embargo, es recomendable el uso de éstos en términos de productividad. También se requiere un entorno de ejecución que es multiplataforma.

Entorno para el desarrollador

Los desarrolladores podrían programar en PHP utilizando editores de texto. Sin embargo es recomendable utilizar editores avanzados *Integrated Development Environment (IDE)* como *Netbeans PHP*, *Eclipse PDT* y *Zend Studio*.

Entorno de ejecución

PHP se puede ejecutar sobre cualquier plataforma. Esta es una de las ventajas de PHP con respecto a ASP.NET.

Para su ejecución es recomendable utilizar la tecnología combinada de *GNU*/*Linux* como sistema operativo y servidor web *Apache*.

En este aspecto, ASP.NET necesita un entorno de ejecución con sistemas operativos que suponen la compra de licencias. PHP por otro lado, permite hacer una configuración sin invertir dinero en licencias.

2.5.3 Zend Framework

La comparación directa entre ASP.NET y PHP no es completamente lógica. ASP.NET es un marco de trabajo, PHP es un lenguaje de programación que no ofrece todas las bondades que tiene un marco de trabajo.

En términos de marcos de trabajo, existen diferentes alternativas para PHP. Zend Framework se ha elegido porque es desarrollado por la empresa pionera de PHP en el mundo y posee un diseño moderno y con buena documentación. Este marco ha sido construido haciendo uso de las últimas versiones del lenguaje PHP e incluye varios patrones de diseño desde sus raíces. El aprendizaje es fácil y permite desarrollar aplicaciones web rápidamente [Allen 08]. Este marco es desarrollado por la empresa *Zend Technologies Ltd.*

Se extienden las mismas características mencionadas para PHP, incluyendo su entorno de ejecución y desarrollo. La versión a la fecha de este marco de trabajo es 1.9.

Características de Zend Framework

- Incluye algunos patrones de diseño implementados.
- Posee diferentes clases que facilitan la implementación de patrones.
- Posee implementaciones de algunos web services famosos (*Google Maps, Youtube*, etc.)
- Posee adaptadores de fácil uso para acceso a capa de datos (RDBMS, LDAP).

2.5.4 Java 2 Enterprise Edition (J2EE)

El lenguaje de programación usado en J2EE es Java. La plataforma J2EE está orientada a la construcción de aplicaciones distribuidas y multicapa. Para lograrlo, se ofrecen diferentes servicios basados en distribución de objetos, gestión de transacciones, seguridad, y gestión de recursos [Monnox 05].

Al proveer una infraestructura empresarial, varias características de J2EE disminuyen el esfuerzo de los desarrolladores en la codificación. Así, los desarrolladores sólo se enfocan en la implementación de los componentes distribuidos que forman parte de la lógica del negocio.

Sobre el lenguaje Java

- Orientado a objetos.
- Basado en componentes: Java tiene diferentes componentes que permiten construir aplicaciones empresariales acelerando el desarrollo.
- Arquitectura independiente de plataforma.
- Recolector de basura.
- Libre de punteros.
- Soporte para hebras.
- Serialización de objetos para integración con otras aplicaciones Java.
- Soportado en algunos dispositivos móviles.
- Varias APIs para servicios como mensajería, LDAP, conectividad con bases de datos, etc.

Características de la plataforma

- *Servlets* y *JavaServer Pages* para la construcción de aplicaciones web.
- *Enterprise JavaBeans* que permiten escribir código sin detalles de implementación de persistencia, transacciones, comunicación remota y otros servicios.
- *Java Transaction API* que permite tener transacciones por aplicación, incluso en aquellas que son distribuidas.

- *Java Message Service* que permite utilizar una API completa para compartir mensajes entre una misma o diferentes aplicaciones.
- *Java Naming and Directory Interface (JNDI)* que provee soporte para localización de servicios.
- JavaMail que provee soporte SMTP, IMAP y POP.

Requisitos para el uso

Aunque no es requerido un entorno para el desarrollador, existen entornos que facilitan la tarea de codificación. Por otro lado, el entorno de ejecución funciona en cualquier plataforma.

Entorno para el desarrollador

Caben destacar dos entornos de desarrollo recomendados para construir aplicaciones con J2EE. Uno de ellos es *NetBeans* [NetBeans 09], el oficial de *Sun Microsystems*. Otro es Eclipse [Eclipse 09], que es un entorno de desarrollo muy flexible y tiene un buen repositorio de plugins para extender sus funcionalidades.

Los dos entornos que se han mencionado son de distribución gratuita.

Entorno de ejecución

La plataforma J2EE puede ser instalada en cualquier sistema operativo y no se debe comprar licencias para su uso. Es una ventaja que tiene, al igual que PHP, frente a ASP.NET/C#.

Para su ejecución se requiere instalar un contenedor de aplicaciones, como *Apache Tomcat* [Tomcat 09] o *GlassFish* [GlassFish 09].

3. Mapping entre ontologías

Como ya se ha comentado en este trabajo, existen diversas fuentes bibliográficas referentes a patrones de diseño. En el momento de realizar este trabajo hay dos fuentes fundamentales entre las cuales hay relaciones semánticas entre conceptos de los diferentes autores. Sin embargo, no hay una relación explícita que permita identificar a simple vista estas relaciones entre capas y patrones de diseño.

Los patrones de diseño establecen un vocabulario común, este vocabulario tiene un alto grado de importancia porque permite tener consistencia en la discusión y elaboración de documentos relacionados con diseño de software [Gamma 94].

El problema de las múltiples fuentes de conceptos semánticamente relacionados, aunque sintácticamente distintos no es nuevo. De hecho un mecanismo de solución a este problema es el mapping entre ontologías. Lo que se busca en este capítulo es introducir el concepto de este tipo de mapping y utilizarlo para unificar las fuentes de información de patrones. De esta forma se propone una solución y un vocabulario común, útil para la comunidad de desarrolladores, y para las discusiones que se realizan a lo largo de este trabajo.

3.1 Ontologías y mappings semánticos

Una ontología es una especificación explícita de un esquema conceptual dentro de uno o varios dominios [Gruber 93]. En este trabajo, se consideran las dos ontologías inducidas por los libros *Core J2EE Patterns* [Alur 03] y *J2EE Design Patterns* [Crawford 03]. Hay una relación implícita entre los conceptos de Alur et. al. y Fowler et. al. que se mencionará más adelante.

Para el propósito de este trabajo, las ontologías especificarán una caracterización jerárquica de conocimiento. Este conocimiento es representado por los patrones y clasificaciones propuestas en cada dominio.

La presencia de ontologías no garantiza la capacidad de hacer comparaciones. Esta capacidad se puede lograr sólo después de realizar mappings entre ontologías o mappings semánticos [Santacruz-Valencia 09]. Un mapping semántico es el proceso donde dos ontologías o modelos conceptuales son relacionados semánticamente en un nivel conceptual [Silva-Rocha 03].

Proyectos como *OntoGlue* utilizan las ontologías y los mappings para ensamblar objetos de aprendizaje con base en el conocimiento asociado [Santacruz-Valencia 09]. Este conocimiento puede ser por ejemplo requisitos y competencias. Este proyecto plantea además la forma en que se pueden hacer mejores comparaciones y búsquedas al tener el conocimiento representado en términos de clases de ontologías.

3.2 Ontologías inducidas

Para realizar el mapping es necesario primero definir las ontologías, en este caso se introducen dos ontologías. La primera ontología se deriva del libro *Core J2EE Patterns* [Alur 03] y la segunda del libro *J2EE Design Patterns* [Crawford 03]. Se utiliza una representación textual de las ontologías en vez de una representación visual. En esta representación, tanto las capas, como los propios patrones se consideran clases de las ontologías. Las instancias de estas clases serían los casos concretos.

3.2.1 Ontología inducida por Core J2EE Patterns [Alur 03]

- Capa de presentación
  - Intercepting Filter
  - Front Controller
  - Context Object
  - Application Controller
  - View Helper
  - Composite View
  - Service to Worker
  - Dispatcher View

- Capa de negocio
  - Business Delegate
  - Service Locator
  - Session Façade
  - Application Service
  - Business Object
  - Composite Entity
  - Transfer Object
  - Transfer Object Assembler
  - Value List Handler

- Capa de integración
  - Data Access Object
  - Service Activator
  - Domain Store
  - Web Service Broker

3.2.2 Ontología inducida por J2EE Design Patterns

- Capa de presentación
  - Decorator
  - Front Controller
  - Model – View – Controller (MVC)
  - Composite View
  - Service to Worker
  - Dispatcher View
  - View Helper
  - Asynchronous Page
  - Caching Filter
  - Resource Pool

- Capa de negocio
  - Composite Entity
  - Domain Object Model
  - Is Dirty
  - Lazy Load
  - PK Block Generator
  - Serialized Entity
  - Stored Procedures for Primary Keys
  - Table Inheritance
  - Tuple Table
  - Business Delegate

  - Business Delegate Factory
  - Service Adapter
  - Service Locator
  - Session Façade
  - DAO Factory
  - Data Access Object (DAO)

- Capa de integración
  - Data Transfer Hash
  - Data Transfer Object (DTO)
  - Row Set DTO
  - Procedure Access Object

- Patrones de concurrencia
  - ACID Transaction
  - Transactional Context
  - Optimistic Concurrency
  - Pessimistic Concurrency
  - Lockable Object
  - Lock Manager
  - Version Number

- Patrones de mensajes
  - Malformed Message Channel
  - Point to Point Distribution
  - Publish – Subscribe
  - Competing Consumers
  - Event-Driven Consumer
  - Message Façade
  - Message Handler
  - Message Selector
  - Polling Consumer
  - Content Aggregator
  - Content – Based Routing
  - Control Bus
  - Pipes and Filters

3.3 Mapping entre las ontologías inducidas por Core J2EE Patterns y J2EE Design Patterns

Una vez identificadas las dos ontologías globales con sus patrones de diseño, se debe hacer un análisis para determinar relaciones y construir un mapping. Para desarrollar este trabajo se tomarán como base la mayoría de patrones definidos en *Core J2EE Patterns* [Alur 03].

Después de realizar el análisis se obtiene un mapa de relaciones, que se presenta a continuación en la Tabla 3.1. Con este mismo análisis se ha determinado que varios patrones de mensajería y algunos de concurrencia sólo existen en *J2EE Design Patterns* [Crawford 03].

En el siguiente mapping las letras “X” representan la no equivalencia de uno o varios patrones en cierta relación. Nótese también que algunos patrones de la ontología origen pueden verse como agregación de otros en la ontología destino (es decir, el mapping es *1 a n*).

Core J2EE Patterns [Alur 03]	**J2EE Design Patterns [Crawford 03]**
Capa de Presentación	
Intercepting Filter	Decorator
Front Controller	Front Controller
Context Object	X
Application Controller	MVC (*aproximado*)
View Helper	View Helper
Composite View	Composite View
Service to Worker	Service to Worker
Dispatcher View	MVC (*aproximado*) + Front Controller + View Helper
Capa de Negocio	
Business Delegate	Business Delegate
Service Locator	Service Locator
Session Façade	Session Façade
Application Service	X
Business Object	Domain Object Model
Composite Entity	Composite Entity
Transfer Object	Data Transfer Object
Transfer Object Assembler	X
Value List Handler	X
Capa de Integración	
Data Access Object	Data Access Object (Capa de Negocio)
Service Activator	Point to Point Distribution + Publish – Subscribe + Message Façade
Domain Store (incluye diferentes patrones definidos en [Fowler 02])	Is Dirty Lazy Load ACID Transaction Lock Manager Transactional Context Optimistic Concurrency Pessimistic Concurrency Version Number

Web Service Broker	X
Patrones de Mensajes	
X	Malformed Message Channel
X	Message Selector
X	Competing Consumers
X	Event-Driven Consumers
X	Message Handler
X	Polling Consumers
X	Content Aggregator
X	Pipes and Filters
X	Content Based Routing
X	Control Bus

Tabla 3.1. Mapping entre ontologías [Alur 03] y [Crawford 03]. Relación de patrones.

El mapping entre ontologías no sólo brinda el vocabulario común de discusión, pues aquí se puede ver como se relacionan temas de conocimiento. Las relaciones jerárquicas de conocimiento y el mapping entre ontologías permiten comparar conocimiento y determinar los requisitos para alcanzar niveles de conocimiento, como se menciona en el proyecto *OntoGlue* [Santacruz-Valencia 09].

3.4 Mapping entre capas

Para definir los mappings entre capas ha sido necesario definir primero los mappings entre los patrones asignados a cada capa (sección anterior).

Como tercera fuente de información se tiene el libro *"Patterns of Enterprise Application Architecture (PoEAA)"* [Fowler 02]. En este libro los autores han definido un mapping entre capas de aplicación con respecto al libro *"Core J2EE Patterns"* [Alur 03].

Es importante anotar que Alur et al. hace referencia en su trabajo a diferentes patrones de Fowler et al. Así, no es necesario hacer un mapping explícito entre [Alur 03] y [Fowler 02]. Las relaciones referenciadas se encuentran al final de la discusión de cada patrón de diseño en el libro de Alur et al.

En la Tabla 3.2 se presenta la relación de capas entre [Alur 03] y [Fowler 02].

Core J2EE Patterns [Alur 03]	**Patterns of Enterprise Application Architecture [Fowler 02]**
Capa de cliente	Capa de presentación (que se ejecuta en cliente)
Capa de presentación	Capa de presentación (que se ejecuta en servidor)
Capa de negocio	Capa de dominio
Capa de integración	Capa de fuentes de datos
Capa de recursos	Recursos externos (no está definido como capa)

Tabla 3.2. Relación entre capas de aplicación de los libros [Alur 03] y [Fowler 02].

Esta relación implícita permite entonces conocer la relación de patrones de Crawford y Kaplan y Fowler et al.

En la Tabla 3.3 se presenta la relación entre capas de Alur et al. y Crawford y Kaplan, como se puede observar, la relación entre capas en este caso no es uno a uno.

Core J2EE Patterns [Alur 03]	**J2EE Design Patterns [Crawford 03]**
Capa de cliente	Capa de presentación del cliente
Capa de presentación	Capa de presentación del lado del servidor
Capa de negocio & capa de integración	Capa de lógica del negocio del lado del servidor
Capa de negocio	Modelo del dominio del lado del servidor
Capa de integración & capa de recursos	Modelo de integración empresarial

Tabla 3.3. Relación entre capas de aplicación de los libros [Alur 03] y [Crawford 03]

4. Jerarquía de Aplicaciones

4.1 Tipos de Aplicaciones

En este capítulo se define la jerarquía de aplicaciones compuesta por tres tipos. Se clasificarán de acuerdo a su complejidad que será medida en base a los patrones de diseño que se requieran para su implementación.

Las aplicaciones descritas se adaptan a esquemas de diseño de múltiples capas y algunas tendrán implementación de patrones para más, o menos capas de acuerdo a su complejidad. La distribución entre capas es una de las técnicas más comunes que usan los diseñadores de software para separar por partes un sistema complicado [Fowler 02].

Empezamos esta jerarquía de aplicaciones distinguiendo entre los conceptos de *sitio web* y *aplicación web*. Un *sitio web* es un conjunto de páginas web que muestra información al usuario. Así, cuando éste realiza una petición al sitio web, se muestra un conjunto de información que puede estar almacenada en ficheros estáticos HTML, DHTML, XML o en una base de datos. Así, distinguimos entre sitios web *estáticos* y *dinámicos* [Conallen 99].

Cuando el usuario en sus peticiones envía datos (normalmente a través de formularios HTML) y éstos a su vez influyen en el comportamiento futuro del sistema, entonces estos sitios se convierten en aplicaciones web. Según Conallen, podemos definir las *aplicaciones web* como un sistema web (servidor web, red, HTTP, navegador) en el que la interacción del usuario (navegación y entrada de datos) afecta el estado del negocio [Conallen 99]. Así, las aplicaciones web contienen lógica de negocio y requieren implementar más patrones de diseño.

Las aplicaciones web están compuestas por lógica de procesamiento y también por estructuras de almacenamiento para persistir información. Este tipo de aplicaciones serán llamadas *aplicaciones con persistencia* y deben garantizar que la información almacenada sea íntegra en todo momento y que su codificación y mantenimiento no sean una labor tediosa.

Los *sitios web dinámicos* que describe Conallen solucionan problemas muy básicos. Sin embargo pueden implementar patrones de diseño como *Front Controller*, *Dispatcher View*, *Transfer Object* y *Data Access Object*. Las aplicaciones web con persistencia incluyen estos patrones; por esta razón no se analizarán los *sitios web* independientemente en este trabajo.

Debido al uso masivo de Internet en los últimos años, surge una gran demanda de usuarios que acceden y modifican simultáneamente la información de las aplicaciones web. Por tanto se deben proporcionar mecanismos que garanticen estabilidad y disponibilidad de la información [Crawford 03], por lo que se hace obligatorio introducir las *aplicaciones con persistencia y concurrencia.*

El tipo que se acaba de mencionar con concurrencia ya está enfocado en la construcción de sistemas *robustos* y ampliamente aceptados en la industria. Sin embargo, estas aplicaciones deben ser escalables para que se adapten a las nuevas exigencias como interoperabilidad entre plataformas. Así se llega al nivel más complejo de la jerarquía, las *aplicaciones con persistencia, concurrencia e intercambio de mensajes.*

Hay un tipo de aplicaciones que se construyen en la actualidad para ofrecer servicios Web. Estas a su vez pueden interactuar con otros servicios Web para responder a las solicitudes de las otras aplicaciones. Para diseñar este tipo de aplicaciones existen patrones de diseño [Monday 03] que no serán analizados en este trabajo. El estudio de estas aplicaciones y patrones queda como trabajo futuro.

A continuación se presentan ejemplos para los tres tipos de aplicación que serán analizados:

a. Aplicaciones con persistencia:

   NetCV.com [NET-CV 09] es una aplicación web que permite a los usuarios construir un currículum vitae en línea y publicarlo en la red en diferentes idiomas.

   Un currículum sólo puede ser modificado por el usuario propietario y nunca se presenta el caso de que varios usuarios pretendan editar un mismo registro a la vez. La información será accedida en modo lectura por los visitantes que desde diferentes países buscan empleados.

   Este es un caso típico de una aplicación con persistencia que no hace uso de sistemas de concurrencia.

b. Aplicaciones con persistencia y concurrencia:

   Sales Force [Sales-Force 09] es un famoso sistema CRM *(Customer Relationship Management)* donde varios usuarios pueden estar operando de forma simultánea.

   En un momento determinado, varios usuarios podrían acceder a modificar la información de un mismo cliente. Un sistema sin concurrencia actualizaría la información reemplazándola por la enviada por el último usuario en ejecutar la acción de "salvar" y no alertaría sobre una sobre-escritura/destrucción de información.

   En el diseño de este tipo de sistemas se deben incluir patrones de concurrencia para prevenir estas situaciones y garantizar que la información sea consistente después de las acciones que ejecuten los usuarios.

c. Aplicaciones con persistencia, concurrencia e intercambio de mensajes:

   Citibank [Citibank 09] es una entidad bancaria que ofrece a sus usuarios una banca en línea para realizar todas sus transacciones.

   Estas aplicaciones web empresariales deben poseer sistemas para intercambio de mensajes y así permitir a sus clientes hacer operaciones como transferencias a cuentas de otras entidades, pagar la factura del móvil, etc.

   Además, estas aplicaciones web deben ser muy escalables, almacenar grandes lotes de información y gestionar adecuadamente la concurrencia.

4.2 Relación entre Aplicaciones de Jerarquía y Patrones de Diseño

Una vez se ha definido la jerarquía y los patrones de diseño que analizarán en este trabajo, se debe establecer una relación entre cada una de las aplicaciones y los patrones requeridos.

Para esto se deben excluir patrones de concurrencia y mensajes para el tipo de aplicaciones más básico. Se debe tener en cuenta que hay diferentes patrones de diseño de la capa de negocio que deben estar incluidos en estas aplicaciones básicas.

En la Figura 4.1 se puede ver un esquema global de la propuesta.

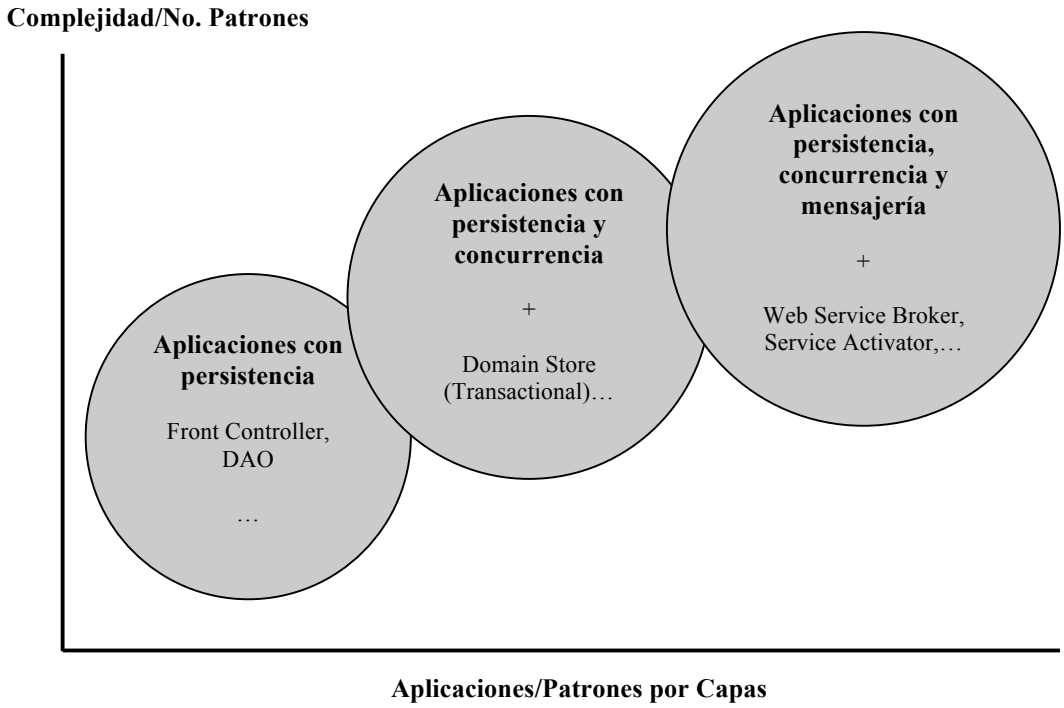

Figura 4.1. Esquema global de jerarquía vs. inclusión de patrones de diseño.

Cada aplicación, mientras avanza en su nivel de complejidad, introducirá nuevos patrones de diseño requeridos para su implementación. Más adelante se planteará una relación entre patrones de diseño que será útil para comprender mejor el alcance.

Cuando se piensa en la correcta evolución del software se debe crear una arquitectura que permita al programador extender y reorganizar los componentes del sistema. Se busca una arquitectura que combine flexibilidad, extensibilidad y rendimiento.

4.3 Patrones para aplicaciones con persistencia

Este tipo de aplicaciones agrupan una buena cantidad de patrones de diseño dado que la capa de presentación es responsable en una buena parte del rendimiento y flexibilidad de los sistemas.

El desarrollo de software no es una labor sencilla, un proyecto puede cambiar muchas veces en menos de un año y seguro la mejor solución no es volver a crearlo. De cierta forma la capa de presentación es la que se encuentra – a diferencia de las demás – relacionada directamente con el usuario final. Muchos de los cambios que se generan en un sistema tiene que ver con la parte gráfica, vinculación de nuevos elementos visuales, creación de interfaces para otro tipo de dispositivos móviles y más.

Todas estas consideraciones se deben tener en cuenta al momento de diseñar. Si se sabe que algunas características serán incluidas en seis meses, un año o incluso más, la mejor práctica es desde el mismo momento crear una arquitectura que pueda soportar los cambios sin problema [Crawford 03].

Por otro lado, uno de los mecanismos de persistencia más comunes en las aplicaciones Web son las bases de datos relacionales. Aunque por supuesto existen otros mecanismos de persistencia para las aplicaciones que siguen siendo usados, como archivos en formato de texto o binarios, y recientemente XML que se ha vuelto popular.

Los lenguajes de programación deben proveer mecanismos para representar objetos en un formato almacenable en disco. Luego podrán ser guardados y recuperados. Utilizar el disco directamente desde la aplicación puede ser una mala práctica porque los datos almacenados podrán sólo ser recuperados por la aplicación que los ha generado. Se podría escribir código que permita leer objetos guardados por otros lenguajes pero implicaría mucho esfuerzo.

Estas prácticas pueden generar diferentes problemas en las aplicaciones, el código será complejo y difícil de comprender. Debido a estos problemas conocidos es importante incluir patrones de persistencia en el diseño de la aplicación.

En la siguiente tabla se establecen los patrones disponibles para este tipo de aplicaciones.

Capa	**Patrones**
Capa de Presentación	Intercepting Filter
	Front Controller
	Context Object
	Application Controller
	View Helper
	Composite View
	Service to Worker
	Dispatcher View
Capa de Negocio	Transfer Object
	Transfer Object Assembler
	Value List Handler
	No aplican patrones de concurrencia
Capa de Integración	Data Access Object
	No aplican patrones de mensajes

Tabla 4.1. Patrones para aplicaciones con persistencia.

4.4 Patrones para aplicaciones con persistencia y concurrencia

Ya se ha mencionado en la descripción de los tipos de aplicación que la concurrencia es una de las diferencias entre sistemas regulares y sistemas empresariales robustos.

Los patrones que se introducen en este tipo de aplicaciones se basan en la gestión de concurrencia. Las transacciones que normalmente se delegan a los gestores de bases de datos son llamadas transacciones del sistema. Sin embargo, estas transacciones deben ser gestionadas por los elementos de la lógica del negocio de manera totalmente transparente de los almacenes de datos. Además, estas transacciones son operaciones que pueden ser vistas inmediatamente por el cliente, como la lógica para descargar un valor de una cuenta donde se afecta el estado de diferentes objetos.

En estas aplicaciones se incluye el patrón *Domain Store* que incluye el objeto *Transaction Manager* para gestionar transacciones a nivel de base de datos. La lógica del negocio debe estar aislada del mecanismo concreto de gestión de transacciones.

Nótese que la presencia de lógica distribuida no está necesariamente ligada a aplicaciones con fuerte concurrencia. En cualquier caso, para este tipo de aplicaciones, que pertenecen a un nivel de complejidad mayor, se ha decidido incluir los patrones que permiten acceder a los clientes a lógica distribuida.

Otro tipo son aquellas que requieren de un flujo de entradas y salidas y pueden interactuar con el usuario a través de diferentes interfaces Web, este tipo requieren nuevos elementos de diseño para solucionar el problema de concurrencia.

A continuación se introducen nuevos patrones. Tanto en la siguiente tabla como en las demás no serán incluidos los patrones de las aplicaciones de nivel de complejidad inferior, se asume que cada aplicación más compleja incluye todos los patrones de la anterior.

Capa	**Patrones**
Capa de Presentación	*Todos*
Capa de Negocio	*Igual al anterior, además:* Business Delegate Service Locator Session Façade Application Service Business Object Composite Entity Optimistic Concurrency Pessimistic Concurrency Version Number
Capa de Integración	Domain Store *No aplican patrones de mensajes*

Tabla 4.2. Patrones para aplicaciones con persistencia y concurrencia.

4.5 Patrones para aplicaciones con persistencia, concurrencia y mensajes

Los mensajes son un componente muy importante para las aplicaciones empresariales, estos permiten que las aplicaciones tengan facilidad de interactuar información con otros sistemas de cualquier tipo.

Los mensajes ofrecen nuevas oportunidades a los arquitectos. Por naturaleza los mensajes son asíncronos y esta situación puede beneficiar a las aplicaciones, desde que el remitente no requiera una respuesta inmediata se pueden construir sistemas que no están obligados a permanecer 100% procesando [Crawford 03]. Pueden ser usados esquemas de servidores centralizados para recibir peticiones.

Uno de los ejemplos que se plantean [Crawford 03] es: sí diez aplicaciones Web requieren enviar faxes, un servidor centralizado puede proveer el servicio a todos en vez de requerir un fax-módem en cada servidor Web. El servidor Web, independiente de si está ejecutando Solaris, Linux u otro sistema, puede comunicarse con el servicio de fax a través de un mensaje.

Capa	**Patrones**
Capa de Presentación	*Todos*
Capa de Negocio	*Todos*
Capa de Integración	Service Activator
	Web Service Broker
	Point-to-Point Distribution
	Publish-Subscribe
	Malformed Message Channel
	Message Façade
	Message Selector
	Competing Consumers
	Event-Driven Consumer
	Message Handler
	Polling Consumer
	Content Aggregator
	Pipes and Filters
	Content – Based Routing
	Control Bus

Tabla 4.3. Patrones para aplicaciones con persistencia, concurrencia y mensajes.

5. Aplicaciones con persistencia

En el capítulo anterior se describieron los diferentes patrones de diseño que se utilizan en las distintas aplicaciones de la jerarquía definida. Ahora se realizará un recorrido por cada patrón revisando sus necesidades tanto tecnológicas como de lenguaje de programación para su implementación. Los siguientes patrones están relacionados con las aplicaciones con persistencia.

Para este tipo de aplicaciones se enumerarán patrones que pueden pertenecer a diferentes capas. Para identificar rápidamente la capa a la que pertenece se ubicará una letra al lado derecho de cada nombre. La capa de presentación tendrá *(P)*, la capa de negocio tendrá *(N)*, la de integración *(I)*.

Como se observará en las próximas páginas, el soporte de persistencia del marco de Microsoft es adecuado para este tipo de aplicaciones. Por otro lado, cuando se usa una tecnología como PHP se encuentran bastantes alternativas. *Doctrine (PHP Object Relational Mapper)* es un marco para PHP que permite crear una capa de abstracción de base de datos. Este marco también incluye un componente clave relacionado con el patrón *Query Object* [Fowler 02] que establece un dialecto orientado a objetos para escribir *SQL* [Doctrine 09].

Otros marcos como *CakePHP* implementan mecanismos de persistencia transparentes y además cuentan con soporte para otros patrones de diseño que se mencionan en este trabajo [CakePHP 09]. Es importante destacar entonces que existen estas alternativas cuando se trata de desarrollar aplicaciones con una tecnología como PHP. Estos marcos de trabajo no serán tomados en cuenta para la implementación de estos patrones y se continuará analizando el soporte nativo de PHP y el soporte de Zend Framework.

5.1 Intercepting Filter (P)

Origen	Core J2EE Patterns [Alur 03]
Descripción	El objetivo principal de este patrón es interceptar y manipular solicitudes y respuestas, antes y después que éstas sean procesadas.
Requisitos de Implementación	- Soporte para creación de interfaces - Soporte de herencia
Comentarios	Algunas aplicaciones tienen requisitos como seguridad y autenticación que pueden ser aplicables a lo largo de todas las solicitudes que se realizan. Utilizando el patrón *Intercepting Filter* se podrá agregar este tipo de funcionalidad para que haya cierta lógica disponible para los componentes de aplicación que la requieran. También se tendrá el código separado y por lo tanto menos acoplamiento. Un objeto se encargará de recibir la solicitud y creará un objeto *FilterChain*. Este último objeto adjuntará dinámicamente los filtros necesarios para esta solicitud [Alur 03]. Los filtros deben tener comportamientos comunes y por esto se recomienda la creación de una interface. Este patrón se suele usar en combinación con *Front Controller*.

Tabla 5.1. Descripción del patrón.

5.1.1 Soporte de ASP.NET/C#

Hay tres formas de atacar el problema desde ASP.NET. Una de ellas es realizar una construcción personalizada, que suele ser más complicado pero ofrece la posibilidad de crear componentes flexibles y personalizados.

La segunda opción es usar el interfaz *IHTTPHandler*, que a través de un archivo de configuración permite que los filtros que la usen intervengan en una serie de eventos definidos por el marco [Msdn-Front 09].

La última opción y no tan útil para el caso es usar la característica de filtros *ISAPI* del servidor *Internet Information Service* (IIS) de Microsoft [Stanek 07]. Los filtros *ISAPI* se implementan a muy bajo nivel y son ideales para realizar funciones como manipulaciones de URL. Desafortunadamente los filtros *ISAPI* sólo pueden ser escritos en C++ y no tienen acceso a ninguna de las funcionalidades del Framework .NET [Msdn 09].

5.1.2 Soporte de PHP

En PHP se pueden crear los interfaces y las clases necesarias para crear y adjuntar los filtros a cada acción del usuario. PHP facilita reconocer el método por el cual se recibe una llamada, como *GET o POST*. También permite crear estructuras de objetos dinámicamente y posee métodos que permiten ejecutar acciones cuando un método es invocado o se cambian propiedades de los objetos.

5.1.3 Soporte de Zend Framework

Al combinar PHP con Zend Framework se puede realizar una implementación más rápida de este patrón. Para esto se deben usar controladores de aplicación y *plugins* (*Zend_Controller_Plugin)* [Allen 08]. Los plugins permiten sobrescribir algunos métodos como *preDispatch* y *postDispatch*, estos métodos son ejecutados respectivamente antes y después de ejecutar la lógica de la acción.

Lenguaje	Solución	Complejidad
ASP.NET/C#	Buena	Menor
PHP	Media	Mayor
PHP y Zend Framework	Buena	Menor

Tabla 5.2. Calificación de los lenguajes.

5.2 Front Controller (P)

Origen	Core J2EE Patterns [Alur 03]
Descripción	Facilita la creación de un punto de interacción inicial entre la interfaz de usuario y el resto del sistema que gestione todas las solicitudes.
Requisitos de Implementación	- Soporte para creación de interfaces - Soporte de herencia
Comentarios	*Front Controller* ayuda a reducir la cantidad de lógica que se requiere en las vistas. Este patrón normalmente se usa en combinación con *Application Controller* quien es el responsable de la gestión de acciones y vistas. También es útil integrarlo con *View Helper* para reducir el código requerido para generar vistas.

Tabla 5.3. Descripción del patrón.

5.2.1 Soporte de ASP.NET/C#

La centralización de todas las solicitudes través de un *Front Controller* se hace en casos muy específicos. ASP.NET tiene integrado el *Page Controller* que puede ser útil en muchos casos. Sin embargo la implementación de este patrón en ASP.NET se debe realizar de forma manual.

Una de las formas de implementar este patrón es creando un objeto *Handler* que se encarga de recibir los parámetros *GET* y *POST*. Basado en los parámetros se selecciona el comando apropiado a ejecutar (nótese que en este caso, el controlador jugaría también el papel de *application controller*). El comando es procesado a través de un procesador de comandos que se debe implementar. Este se encargará de hacer parte de la lógica y finalmente desplegar una vista para mostrar la página [Msdn 09]. El *Handler* se puede implementar utilizando el *API* de bajo nivel *HTTPHandler* que provee el Framework.

5.2.2 Soporte de PHP

Esta implementación se puede realizar en PHP de varias formas, se debe tener en cuenta que la construcción se debe hacer desde cero. Ya que la tecnología completa trae consigo un servidor Web que comúnmente es *Apache*, se usan los modos de sobre-escritura de URLs para permitir direcciones amigables en las peticiones [McCallum 04]. Se puede hacer una implementación similar a la mencionada en ASP.NET. También se puede crear un objeto principal que actúe como despachador hacia las acciones contenidas dentro del *Application Controller.*

5.2.3 Soporte de Zend Framework

El Framework oficial de PHP incluye el objeto *Zend_Controller_Front* que implementa este patrón de diseño. La implementación es muy sencilla y este mismo objeto está contenido dentro del modelo *MVC*.

Zend_Controller_Front implementa consigo un patrón *Singleton* para garantizar que sólo una instancia estará disponible en un momento dado [ZF 09].

Lenguaje	Solución	Complejidad
ASP.NET/C#	Media	Mayor
PHP	Media	Media
PHP y Zend Framework	Buena	Menor

Tabla 5.4. Calificación de los lenguajes.

5.3 Context Object (P)

Origen	Core J2EE Patterns [Alur 03]
Descripción	Encapsula datos provenientes de la interfaz para su uso en el resto del sistema. De esta forma se encapsulan estados de aplicación a lo largo de la ejecución.
Requisitos de Implementación	- Soporte de herencia - Soporte para creación de interfaces (dependiendo del tipo de implementación)
Comentarios	*Context Object* es muy útil cuando se quiere interactuar con los datos de la aplicación en una forma independiente a protocolos. Por

	ejemplo las variables de un formulario que se recibe por HTTP se guardan en el objeto y toda la aplicación interactuará con él. De esta forma si surge un cambio en el protocolo HTTP solo se deberá adecuar la lógica en un punto de la aplicación [Alur 03].

Tabla 5.5. Descripción del patrón.

5.3.1 Soporte de ASP.NET/C#

El Framework de ASP.NET trae diferentes objetos de contexto que resuelven este problema. *HttpContext* encapsula toda la información específica de HTTP y la deja disponible para toda la aplicación en una forma independiente de protocolo [Msdn 09].

También existe la clase *HttpApplicationState* donde su instancia se comporta como un diccionario de objetos que permanece disponible a lo largo de la aplicación e incluso entre diferentes solicitudes. Este objeto se crea la primera vez que un usuario hace una petición a cualquier URL de la aplicación. Tiene una referencia directa con *HttpContext*. Nótese que en este caso hay una ligera perversión del patrón, ya que el contexto, según Alur et al. debe ser independiente del protocolo utilizado en la aplicación.

Si el patrón se requiere implementar con soporte para otro tipo de protocolos personalizados se debe implementar manualmente. Existen unas clases libres que se pueden usar para construir objetos a partir del formulario [ASP-Beans 09].

5.3.2 Soporte de PHP

PHP facilita la implementación del patrón proporcionando métodos de lectura de diferentes protocolos. Desde cualquier lugar de la aplicación se puede acceder a información que indica el método de envío, cabeceras, versión de protocolo, datos y demás.

La implementación en PHP se debe realizar completa.

5.3.3 Soporte de Zend Framework

A diferencia de ASP.NET, Zend Framework no incluye clases de contexto que faciliten el acceso a los datos. La implementación se debe realizar completa pero teniendo en cuenta que el Framework de algunas clases como *Zend_Application* que permiten ejecutar acciones antes de que cualquier solicitud sea procesada por la lógica de la aplicación, un momento adecuado para crear el objeto de contexto.

Lenguaje	Solución	Complejidad
ASP.NET/C#	Buena	Menor
PHP	Media	Mayor
PHP y Zend Framework	Media	Media

Tabla 5.6. Calificación de los lenguajes.

5.4 Application Controller (P)

Origen	Core J2EE Patterns [Alur 03]
Descripción	Permite centralizar la gestión e invocación de componentes que procesen una solicitud como comandos y vistas de manera independiente del protocolo utilizado por los elementos de la interfaz de usuario (capturados por el *Front Controller*).

Requisitos de Implementación	- Soporte de herencia - Soporte para creación de interfaces
Comentarios	La combinación de este patrón con *Front Controller* permite crear una solución potente para la gestión de solicitudes. El objetivo es realizar una relación dependiendo de los parámetros de entrada para seleccionar las clases específicas que deben procesar la información y además seleccionar los componentes gráficos (vistas) adecuados para cada solicitud. Si se usa *Front Controller* se puede centralizar el lugar de recepción de solicitudes (dependientes del protocolo utilizado por la interfaz de usuario) y despachar la información al *Application Controller*. Esto es una buena práctica.

Tabla 5.7. Descripción del patrón.

5.4.1 Soporte de ASP.NET/C#

La solución recomendada para la implementación de este patrón en ASP.NET/C# es haciendo uso del patrón *Model-View-Controller* (MVC) que trae integrado el Framework. La implementación es relativamente sencilla porque el desarrollador no tendrá que escribir mucho código para lograrlo. Para esto se debe extender una clase de controlador de *System.Web.UI.Page* [Msdn 09].

Este patrón finalmente facilita el acceso a objetos del negocio para interactuar con lógica más avanzada de la aplicación. De esta misma forma se comportan los modelos en MVC.

5.4.2 Soporte de PHP

Este patrón se puede implementar en esta tecnología sin problema pero se debe escribir una buena cantidad de código para lograrlo. Cuando se usa PHP sin un Framework se debe diseñar y escribir el patrón, que incluye *Front Controller* (opcional), *Application Controller, Command Handler* y mecanismos para gestión de vistas [Alur 03].

Para la transformación y despliegue de vistas existen diferentes motores de plantillas que se pueden integrar [Sweat 05]. *WACT* [WACT 09] es un motor de plantillas que usa etiquetas personalizadas *Custom Tags* en las vistas. *PHP-TAL* [PHP-TAL 09] es un motor que permite definir plantillas con XML.

5.4.3 Soporte de Zend Framework

El Framework de PHP facilita bastante la creación de este patrón gracias a sus diferentes componentes que trae integrados. *Zend_Controller_Front* y *Zend_Controller* son usados para construir el núcleo.

Zend Framework incluye los componentes adicionales para implementar los modelos y así completar la misma implementación de MVC que se usa en ASP.NET. También se puede usar la clase *Zend_View* para crear objetos tipo vista y lanzarlos al cliente [ZF 09].

Lenguaje	Solución	Complejidad
ASP.NET/C#	Buena	Menor
PHP	Media	Mayor
PHP y Zend Framework	Buena	Menor

Tabla 5.8. Calificación de los lenguajes.

5.5 View Helper (P)

Origen	Core J2EE Patterns [Alur 03]
Descripción	Permite encapsular lógica para el procesamiento de elementos que son comúnmente usados en diferentes vistas. Un *View Helper* es un adaptador entre la vista y el modelo y su procesamiento está asociado a la lógica de formato, como generar elementos HTML.
Requisitos de Implementación	- Soporte de herencia - Soporte para creación de interfaces (opcional)
Comentarios	Hay diferentes beneficios al utilizar este patrón de diseño. Se pueden crear vistas basadas en plantillas y prevenir que se incluya lógica de aplicación en la vista. Al separar la lógica de la aplicación de la vista se facilita la labor de los diseñadores gráficos y desarrolladores de software [Alur 03].

Tabla 5.9. Descripción del patrón.

5.5.1 Soporte de ASP.NET/C#

El Framework provee *View Helpers* que facilitan la generación de HTML para las vistas [Msdn 09]. Actualmente hay métodos que permiten generar los elementos gráficos que son usados comúnmente en las aplicaciones Web.

Si se requieren usar *View Helpers* personalizados, estos se pueden crear dentro de un *namespace* y luego se podrán importar y usar en las diferentes vistas. ASP.NET AJAX Framework incluye también *Helpers* que permiten escribir código claro y resumido en *JavaScript* para describir comunicaciones asíncronas en las vistas [Wenz 07].

5.5.2 Soporte de PHP

PHP en sí es un ejemplo de un tipo específico de plantillas para vistas llamado "página de servidor" [Sweat 05]. La implementación del patrón *View Helper* es sencilla en PHP, aunque es mucho más recomendado usar sistemas de plantillas y *Helpers* para PHP como *Smarty* [Smarty 09] que cumplen con las especificaciones del patrón y pueden integrarse con otros patrones de diseño que se estén usando en la aplicación.

5.5.3 Soporte de Zend Framework

En Zend Framework existe *Zend_View*, los objetos de este tipo permiten gestionar vistas rápidamente. Además *Zend_View_Helper* incluye métodos para generar diferentes elementos gráficos, como en el caso de ASP.NET.

Se pueden crear *View Helpers* personalizados implementando *Zend_View_Helper_Interface* o extendiendo la clase *Zend_View_Helper_Abstract.* Zend Framework puede ser fácilmente integrado con *Smarty*.

Lenguaje	Solución	Complejidad
ASP.NET/C#	Buena	Menor
PHP	Media	Media
PHP y Zend Framework	Buena	Menor

Tabla 5.10. Calificación de los lenguajes.

5.6 Composite View (P)

Origen	Core J2EE Patterns [Alur 03]
Descripción	Permite construir vistas compuestas de múltiples vistas pequeñas. Estas vistas pequeñas pueden ser integradas dinámicamente en la plantilla principal.
Requisitos de Implementación	- No hay requisitos especiales
Comentarios	La mayoría de aplicaciones Web deben presentar cierto tipo de contenido que es igual en todas las interfaces a lo largo una sesión de usuario. Por ejemplo un menú de aplicación, una casilla de búsqueda u otro contenido que debe estar siempre en pantalla y cambia de acuerdo al comportamiento del usuario en el sistema. Al tener estas partes gráficas separadas se puede reutilizar el código y se garantiza un mejor mantenimiento de la aplicación.

Tabla 5.11. Descripción del patrón.

5.6.1 Soporte de ASP.NET/C#

Para resolver este problema se pueden usar páginas maestras (*Master Pages*) que permiten crear plantillas consistentes compuestas de múltiples vistas pequeñas [Msdn 09].

5.6.2 Soporte de PHP

Este lenguaje permite incluir contenido en los *Scripts* que se encuentren en otros ficheros. Así mismo se puede ver implementada la solución en *JSP*. De esta forma se puede crear una plantilla similar a la solución de páginas maestras de ASP.NET.

Se puede crear también una clase *Composite View* que cree la vista definitiva usando las pequeñas, tras cada solicitud del usuario. Esta solución requiere escribir más código.

5.6.3 Soporte de Zend Framework

Zend Framework cuenta con *Zend_Layout* que ha sido creado a partir de las ideas de los patrones *Composite View* y *Transform View* [ZF 09]. Es muy simple implementar vistas/plantillas que re-usen otras vistas de la aplicación.

Lenguaje	Solución	Complejidad
ASP.NET/C#	Buena	Menor
PHP	Buena	Menor
PHP y Zend Framework	Buena	Menor

Tabla 5.12. Calificación de los lenguajes.

5.7 Service to Worker (P)

Origen	Core J2EE Patterns [Alur 03]
Descripción	Es útil para centralizar la gestión de solicitudes y entregar un modelo de presentación antes de entregar el control a una vista. La vista se crea de acuerdo a la respuesta del modelo de presentación.
Requisitos de Implementación	- Soporte de herencia - Soporte para creación de interfaces
Comentarios	Este patrón requiere de otros para ser diseñado. El control centralizado se hace a través de *Front Controller*, el manejo de solicitudes a través de *Application Controller* y se usa *View Helper* en las vistas [Alur 03]. *Front Controller* crea un objeto *Context Object* para encapsular datos dependientes del protocolo y entregarlos a un objeto *Application Controller.* Este último se comporta como un gestor de comandos y delega responsabilidades de lógica a otro objeto. Finalmente se provee un modelo de presentación. *Model-View-Controller* y *Front Controller* permiten crear este patrón [Crawford 03].

Tabla 5.13. Descripción del patrón.

5.7.1 Soporte de ASP.NET/C#

Se ha mencionado el patrón MVC que trae integrado el marco de Microsoft. El uso de este patrón combinado entonces con *Front Controller* y con *View Helper* permite implementar *Service to Worker*.

5.7.2 Soporte de PHP

Si se están implementando soluciones propias y se cuenta con la solución MVC ó *Application Controller + View Helper*, entonces se podrá integrar con *Front Controller* para solucionar el problema. La implementación de este patrón en este lenguaje involucra la creación de otros para su funcionamiento. Tiene un nivel de complejidad más alto, a diferencia de las otras dos tecnologías analizadas.

5.7.3 Soporte de Zend Framework

De igual forma que en ASP.NET, Zend Framework incluye MVC y además *Zend_Controller_Front*. Fácilmente se puede implementar este patrón.

Lenguaje	Solución	Complejidad
ASP.NET/C#	Buena	Menor
PHP	Media	Media
PHP y Zend Framework	Buena	Menor

Tabla 5.14 Calificación de los lenguajes.

5.8 Dispatcher View (P)

Origen	Core J2EE Patterns [Alur 03]
Descripción	Permite que las vistas sean el punto de acceso principal de una solicitud, entonces éstas usarán lógica del negocio sólo cuando sea necesario y de una forma limitada.
Requisitos de Implementación	- Soporte para creación de interfaces
Comentarios	Este patrón es el caso contrario a *Service to Worker*, donde se despachaba un comando o acción y luego una vista. En este caso siempre se despachará una vista que tendrá autonomía de acceder a lógica del negocio. Del mismo modo requiere *Front Controller*, *Application Controller* y *View Helper* para su funcionamiento. Este patrón suele ser usado en casos donde el acceso a datos y lógica es mínimo. Por ejemplo cuando una respuesta es completamente estática (HTML) [Alur 03] o dinámica pero generada a partir de plantillas.

Tabla 5.15. Descripción del patrón.

5.8.1 Soporte de ASP.NET/C#

Este patrón tiene algunas consecuencias negativas, como obligar a tener poca separación entre vistas y lógica de aplicación [Alur 03]. Hay dos estrategias para implementar este patrón, una de ellas es convirtiendo el objeto *Application Controller* en un gestor de vistas. La otra es despachar las vistas directamente desde *Front Controller* [Alur 03].

En ASP.NET se puede utilizar *Page Controller* para despachar vistas directamente y se permite el acceso desde estas vistas a los modelos de MVC que procesan lógica del negocio [Msdn 09].

5.8.2 Soporte de PHP

Sí ya se han implementado los patrones relacionados es muy sencillo adaptar las clases de *Front Controller* o *Application Controller* para que despachen vistas en primera instancia. Desde los *Scripts* de las vistas se pueden acceder a datos de la aplicación (bases de datos) o directamente a objetos que procesen lógica del negocio.

5.8.3 Soporte de Zend Framework

Zend Framework permite adaptar *Zend_Controller* para que despache vistas directamente, éstas a su vez pueden usar objetos *View Helper*. Este tipo de objetos pueden acceder directamente los objetos *Zend_Model* pertenecientes al MVC. De esta forma desde las vistas se puede acceder a los datos del negocio.

Lenguaje	Solución	Complejidad
ASP.NET/C#	Buena	Menor
PHP	Media	Media
PHP y Zend Framework	Buena	Menor

Tabla 5.16. Calificación de los lenguajes.

5.9 Transfer Object (N)

Origen	Core J2EE Patterns [Alur 03]
Descripción	Permite transferir diferentes datos a través de las capas de aplicación, ocultando la representación interna de los datos en cada capa.
Requisitos de Implementación	- Soporte para serialización de objetos (puede ser opcional, hay casos en que un objeto *Transfer* se pasa entre objetos sin necesidad de usar persistencia)
Comentarios	En las aplicaciones multicapa el envío de información entre capas debe hacerse de la forma más eficiente posible. De esta forma, debe evitarse en la medida de lo posible la invocación múltiple de mensajes entre elementos de distintas capas, y la exposición de detalles internos de representación de los datos en cada capa. Así, por ejemplo, en las aplicaciones se debe acceder a elementos de la capa de integración como *Data Access Objects* desde la capa de negocio o incluso de presentación. *Transfer Object* permite encapsular los datos que se transmiten para prevenir que se hagan diferentes llamados para leer todas las propiedades de un objeto. Este mismo patrón permite transferir datos relacionados con objetos de negocio *Business Objects* que se usan en un tipo de aplicaciones con lógica más avanzada y pueden estar implementados como objetos remotos. En este caso este patrón reduce el tráfico de red.

Tabla 5.17. Descripción del patrón.

5.9.1 Soporte de ASP.NET/C#

La implementación de este patrón sugiere la construcción de clases para la transferencia de cada objeto de la aplicación. Estas clases se pueden implementar con C#.

Sin embargo, el marco de Microsoft contiene unos elementos llamados *DataSet* que son una subclase de *System.Data.DataSet*. Estos objetos son genéricos y evitan tener que crear las clases para transferencia de objetos. Los *DataSet* se pueden persistir en cadenas de caracteres [Msdn 09].

Para hacer persistencia en C# se puede especificar una *propiedad* en la declaración de las clases. Esta propiedad es *Serializable*, opcionalmente algunos atributos de la clase se pueden excluir de la serialización con *NotSerialized*. Si se requiere más control de la persistencia se debe extender el interface *ISerialized* [Mayo 01].

5.9.2 Soporte de PHP

En PHP se pueden implementar clases *Transfer Object* para encapsular los datos de los objetos que se desean transferir entre las capas.

PHP tiene integrado un mecanismo que permite persistir datos en disco, sesión y memoria. Los métodos *serialize* y *unserialize* son quienes proveen esta funcionalidad.

5.9.3 Soporte de Zend Framework

Del mismo modo que en PHP, las clases para encapsular la transferencia se deben implementar, Zend Framework no tiene un soporte como ASP.NET para facilitar la implementación de este patrón. La persistencia de información funciona igual que en PHP.

Lenguaje	Solución	Complejidad
ASP.NET/C#	Buena	Menor
PHP	Media	Mayor
PHP y Zend Framework	Media	Mayor

Tabla 5.18. Calificación de los lenguajes.

5.10 Transfer Object Assembler (N)

Origen	Core J2EE Patterns [Alur 03]
Descripción	Permite obtener un modelo de la aplicación compuesto por varios objetos *Transfer Object* que pueden contener datos a su vez de diferentes componentes del negocio.
Requisitos de Implementación	- Soporte para serialización de objetos
Comentarios	Los datos que este patrón ensambla se transmiten en un objeto *Transfer Object* que contiene entonces el modelo de datos de la aplicación. Estos datos son usados por el cliente sólo en modo de lectura. Los objetos del negocio pueden producir objetos de transferencia compuestos. Un *Transfer Object Assembler* puede contener datos de diferentes fuentes como *Data Access Objects, Business Objects, Application Services* [Alur 03]. Estos patrones se describirán en las aplicaciones con concurrencia, en el siguiente capítulo.

Tabla 5.19. Descripción del patrón.

5.10.1 Soporte de ASP.NET/C#

En el punto anterior se describe cómo pueden ser implementados los objetos *Transfer Object* en esta tecnología. El ensamblador se debe construir creando la clase *Transfer Object Assembler* que permite acceder a los diferentes componentes del negocio y crear el objeto de transferencia que se entregará al cliente.

5.10.2 Soporte de PHP

De la misma forma que en ASP.NET, se requiere tener implementado el patrón *Transfer Object.* De esta forma se puede implementar la clase ensambladora para compartir los datos entre las diferentes capas de la aplicación.

5.10.3 Soporte de Zend Framework

Debido a que Zend Framework no tiene un sistema similar a los *DataSet* de APS.NET, la implementación se debe hacer completa. Se debe crear la clase principal *Transfer Object Assembler* de la misma forma que en PHP.

Lenguaje	Solución	Complejidad
ASP.NET/C#	Buena	Menor
PHP	Media	Mayor
PHP y Zend Framework	Media	Mayor

Tabla 5.20. Calificación de los lenguajes.

5.11 Value List Handler (N)

Origen	Core J2EE Patterns [Alur 03]
Descripción	Ofrece funcionalidad de búsqueda y de iteración en resultados. A su vez guarda los resultados en caché para agregar eficiencia al proceso.
Requisitos de Implementación	- Soporte de herencia - Soporte para creación de interfaces
Comentarios	En *EJB* [Java-EJB 09] de *Java*, el uso de métodos de búsqueda *finder methods* o la ejecución de consultas *EJB queries* puede ser muy costoso si el número de objetos del negocio que se seleccionan es muy grande. Este patrón propone puentear el uso de estos mecanismos de selección de objetos del negocio interrogando directamente a objetos *Data Access Object*. En función del tamaño de la respuesta, podrían utilizarse objetos *Transfer Object* u otros recubrimientos del *ResultSet* para guardar la información. Si los resultados de la búsqueda se desean guardar en caché en la capa del servidor, sería necesario el uso de mecanismos como *EJBs* de sesión con estado para poder guardar los resultados durante las distintas iteraciones de la capa de presentación [Alur 03]. En adición, un cliente podría tener problemas manejando grandes resultados de datos, este mecanismo permite al cliente hacer búsquedas e iterar en los resultados más rápidamente.

Tabla 5.21. Descripción del patrón.

5.11.1 Soporte de ASP.NET/C#

Este patrón no está implementado en el marco de trabajo de ASP.NET. Para agregarlo se debe crear un interface *ValueListIterator*. Este interface será implementado por cada clase que representa un sistema de iteración de resultados, que normalmente vienen de un objeto *Data Access Object* [Alur 03].

C# incluye interfaces como *IEnumerator* y *IEnumerable* que son útiles cuando se busca agregar comportamiento iterativo a un objeto [Cooper 03].

ASP.NET posee *Session State* que permite almacenar información temporal que se mantiene tras diferentes solicitudes. Esta información puede ser almacenada en el espacio de memoria de ASP.NET, en SQL Server o en un servidor de estados *State Server* [Gibbs 03].

También, en la librería de clases de este marco existe la clase *DataPager* que permite paginar una colección de datos para posteriormente transmitir los resultados por partes [Msdn-DataPager 09]. Esta es otra solución para implementar este patrón.

5.11.2 Soporte de PHP

El interface y la clase de iteración se crean de la misma forma que se ha descrito para ASP.NET. PHP es un lenguaje menos exigente en la descripción de tipos, una variable de tipo arreglo puede contener cualquier tipo de información e incluso diferentes tipos de objetos.

PHP incluye en su *Standard PHP Library (SPL)* el interface *Iterator* que debe ser implementado cuando se requiere agregar un comportamiento iterativo a un objeto [Sweat 05]. Además se puede construir un objeto de iteración genérico que puede ligarse a cualquier objeto que implemente el interface *IteratorAggregate* [PHP 09].

PHP incluye soporte para sesiones de usuario, el almacenamiento de esta información siempre se realiza en disco [Gutmans 04]. No es recomendable almacenar mucha cantidad de datos en la sesión nativa de PHP. Se puede implementar un manejador de sesión personalizado que permita guardar y consultar datos de una base de datos. Para esto se debe cambiar el método encargado de gestionar la sesión a través del método *session_set_save_handler* [PHP 09].

En PHP se puede utilizar la clase de paginación *Pager* [Pear-Pager 09] que permite extraer resultados por partes de cualquier colección de datos. Esta clase hace parte de la librería *PEAR*, que contiene componentes extendidos para usar en aplicaciones PHP.

5.11.3 Soporte de Zend Framework

Se puede implementar teniendo en cuenta las mismas consideraciones de ASP.NET y PHP. Sin embargo, Zend Framework provee la clase *Zend_Session* que permite administrar la información contenida en la sesión [Allen 08]. En este marco de trabajo la solución recomendada para almacenar datos en caché es usando el componente *Zend_Cache* [Allen 08]. Este permite almacenar resultados de bases de datos y cualquier tipo de objeto fácilmente.

De otra forma, este patrón se podría implementar en este marco utilizando el componente *Zend_Paginator* que es flexible y permite paginar colecciones de datos para presentarlas a un cliente [ZF 09]. La paginación se puede realizar sobre cualquier colección de datos y no sólo sobre datos de una base de datos. Los datos se traen por partes y el cliente debe especificar a cual página de resultados quiere acceder.

Lenguaje	Solución	Complejidad
ASP.NET/C#	Buena	Menor
PHP	Buena	Media
PHP y Zend Framework	Buena	Menor

Tabla 5.22. Calificación de los lenguajes.

5.12 Data Access Object (I)

Origen	Core J2EE Patterns [Alur 03]
Descripción	Permite encapsular el acceso al mecanismo de persistencia. Un objeto de este tipo gestiona la conexión con la fuente de datos y permite leer y almacenar datos transparentemente, ocultando detalles de implementación al cliente.
Requisitos de Implementación	- Conexión con mecanismos de persistencia (RDBMS, LDAP, archivos, etc.) - Soporte para serialización de objetos
Comentarios	El cliente puede interactuar siempre con los objetos *Data Access Object (DAO)* y no debe tener conocimiento de la fuente de datos que se usa. Estos objetos no se encargan de guardar resultados en caché y por esto son ligeros [Alur 03]. Un DAO puede ser usado por objetos *Business Object, Application Services* y otros que requieran acceder o persistir datos.

Tabla 5.23. Descripción del patrón.

5.12.1 Soporte de ASP.NET/C#

Un objeto *Data Access Object* puede acceder a fuentes de información como bases de datos relacionales, *LDAP*, repositorios *XML*, archivos o incluso servicios web.

Este marco de trabajo permite usar *ADO.NET* que contiene componentes para permitir acceso a diferentes fuentes de datos *RDBMS* y *XML* [Worley 01].

Los resultados de una consulta a una fuente de datos a través de *ADO.NET* pueden ser procesados directamente o puestos en un *DataSet* [Msdn 09]. Este último es útil para la transferencia a través de *Transfer Object*.

Dos objetos importantes de *ADO.NET* son *Connection* y *Command.* El primero provee una conexión entre la aplicación y la fuente de datos. Por otro lado, para efectuar operaciones en la base de datos *ADO.NET* utiliza *Command* [Worley 01].

Usando *System.DirectoryServices* se puede acceder a servicios de directorio como *LDAP*.

5.12.2 Soporte de PHP

PHP incluye soporte para acceso a los diferentes mecanismos mencionados. Para bases de datos se pueden utilizar dos de las capas de acceso a datos más usadas en PHP, *ADOdb* o *PHP Data Objects (PDO)*. Estas capas facilitan un *API* para interactuar con motores como *Oracle*, *PostgreSQL*, *Sybase*, *MySQL* y otros [Sweat 05]. Este lenguaje también ofrece soporte para conexiones a LDAP y mecanismos para lectura/escritura eficiente de ficheros *XML*.

La persistencia de objetos en ficheros se puede lograr utilizando los llamados *serialize* y *unserialize* que son nativos del lenguaje. Sin embargo, estos métodos de persistencia no permiten almacenar objetos que representen recursos como conexiones a gestores de datos y manejadores de archivos [Gutmans 04]. Como en *Java*, PHP permite sobre-escribir métodos como *sleep* y *wakeup* que permiten especificar lógica que se debe realizar cuando un objeto se está persistiendo o recuperando.

5.12.3 Soporte de Zend Framework

Las mismas indicaciones de PHP aplican para Zend Framework. Sin embargo, este marco de trabajo incluye la clase *Zend_Db_Table* [Allen 08]. Esta clase puede ser heredada por otras que encapsulen el acceso a una tabla de un motor de base de datos. Finalmente toda la interacción se hace a través de la capa de acceso a datos *PHP Data Objects (PDO)*.

Zend Framework incluye una serie de adaptadores que se extienden de la clase abstracta *Zend_Db_Adapter_Abstract*. Cada motor de datos contiene su propia una implementación [Allen 08]. A través de la clase *Zend_Db* y otras derivadas se puede interactuar con un *RDBMS* escribiendo menos código *SQL*, de acuerdo al patrón *Query Object* [Fowler 02]. Esta clase facilita la implementación de los objetos *Data Access Object*.

El marco también incluye *Zend_Ldap* que facilita la comunicación con servicios de directorio [Allen 08].

Lenguaje	Solución	Complejidad
ASP.NET/C#	Buena	Menor
PHP	Buena	Media
PHP y Zend Framework	Buena	Menor

Tabla 5.24. Calificación de los lenguajes.

6. Aplicaciones con persistencia y concurrencia

En la introducción al capítulo 5 se mencionan algunas alternativas como marcos de persistencia para PHP. En este capítulo se mencionarán además alternativas para la comunicación remota con objetos como la extensión *Universe* [PHP-Universe 09].

Las únicas extensiones/librerías terceras que se considerarán favorables para la tecnología en este análisis serán aquellas que cuenten con soporte en la actualidad y tengan un desarrollo continuo.

En el caso de ASP.NET/C# se mencionará el soporte de otros marcos de Microsoft que se pueden integrar con esta tecnología para construir aplicaciones empresariales.

6.1 Business Delegate (N)

Origen	Core J2EE Patterns [Alur 03]
Descripción	Permite encapsular el acceso a los servicios del negocio. El objetivo es ocultar detalles de implementación de servicios como búsquedas, creación de datos, etc.
Requisitos de Implementación	- Soporte de herencia - Manejo de excepciones - Comunicación remota con objetos
Comentarios	Este patrón busca independencia entre los clientes y los mecanismos de acceso a objetos. Los clientes pueden estar en máquinas diferentes. Por esto es necesario que el lenguaje provea de mecanismos de comunicación remota. *Business Delegate* comporta diferentes beneficios al diseño de la aplicación: permite gestionar errores a nivel de aplicación, implementar mecanismos para reintentar una operación o recuperarse después de un fallo [Alur 03]. Usualmente requiere de un *Service Locator* para la localización de servicios del negocio. Hay dos estrategias para implementar este patrón. Una de ellas es implementándolo con un comportamiento igual al patrón *Proxy*, que permite que se traten objetos como si fuesen locales. La otra es usando un patrón *Adapter*, para proveer integración con sistemas incompatibles. Ambos patrones son definidos en el libro *Design Patterns* [Gamma 94].

Tabla 6.1. Descripción del patrón.

6.1.1 Soporte de ASP.NET/C#

Este patrón puede implementarse en este marco creando una clase *Business Delegate*, que ofrecerá el acceso a los objetos remotos de la capa de negocio. Haciendo uso de C# se puede escribir un código claro y de fácil mantenimiento. C# además permite manejar excepciones de aplicación.

El marco de trabajo de Microsoft incluye *.NET Remoting* [NET-Remoting 09], que provee diferentes utilidades para la gestión de objetos que residen en diferentes dominios de aplicación, procesos, y diferentes máquinas. El servicio *.NET Channel Services* provee el mecanismo de transporte. Por otro lado, *.NET* ofrece un canal HTTP que usa *SOAP* por defecto y TCP que usa una carga binaria [Evans 02]. Los objetos se pueden exponer en cualquiera de los dos protocolos.

6.1.2 Soporte de PHP

PHP tiene un buen sistema de manejo de excepciones. Es flexible y permite crear una clase que extiende a *Exception* para especificar un comportamiento global de la aplicación para excepciones que no sean capturadas en tiempo de ejecución [PHP 09]. Las excepciones se pueden traducir, es decir que capturando una excepción de conexión remota (por ejemplo *SOAP)* se puede lanzar otra que será manejada por los demás objetos de la aplicación.

PHP integra componentes para acceso a procedimientos remotos RPC a través de la extensión *XML-RPC*. También se puede acceder a objetos, datos y procedimientos remotos a través de la extensión *SOAP* [Vaswani 02]. *SoapServer* y *SoapClient* son las clases más importantes. Para llamadas a procedimientos remotos también existe el proyecto *Perfect High Performance Remote Procedure Call (PHPRPC)* que establece un protocolo ligero e independiente del lenguaje [PHPRPC 09].

Si este patrón se implementa basado en el protocolo HTTP se podría usar SOAP para exponer los objetos.

En adición, existe soporte para CORBA [PHP-Universe 09]. Sin embargo, esta extensión no está incluida en la distribución de PHP. La extensión para acceso a objetos CORBA es llamada *Universe* y permite crear instancias a través de su referencia *Interoperable Object Reference (IOR)* y tratarlos como si fuesen locales [Atkinson 03].

Si para la aplicación desarrollada se puede realizar comunicación remota vía SOAP, PHP permitiría implementar este patrón de diseño. En un caso contrario, este lenguaje no provee más mecanismos para comunicación remota con objetos.

6.1.3 Soporte de Zend Framework

Para comunicación remota, el marco de Zend incluye *Zend_Soap* y *Zend_XmlRpc*, que son basadas en las mismas utilidades descritas para PHP. Estos protocolos han sido diseñados para otros propósitos y no pueden ser comparados con *Java RMI* o *CORBA*. XML genera más tráfico de red para transmisión de datos. Del mismo modo que en PHP, no hay un soporte para comunicación remota con objetos diferente a SOAP.

Lenguaje	Solución	Complejidad
ASP.NET/C#	Buena	Menor
PHP	Mala	n/a
PHP y Zend Framework	Mala	n/a

Tabla 6.2. Calificación de los lenguajes.

6.2 Service Locator (N)

Origen	Core J2EE Patterns [Alur 03]
Descripción	Permite encapsular la ubicación de servicios y componentes. Oculta detalles de implementación del mecanismo de localización y centraliza la creación de servicios a través de un único objeto.
Requisitos de Implementación	- Soporte de herencia - Manejo de excepciones - Comunicación remota con objetos
Comentarios	Este patrón permite ocultar y reutilizar los mecanismos de búsqueda y acceso a servicios de aplicación como fuentes de datos y servicios web.

	Este patrón también es útil para localizar objetos remotos, y componentes de *Java Message Service (JMS)*. El soporte para transporte de mensajes será discutido en las *aplicaciones con persistencia, concurrencia y mensajes*.

Tabla 6.3. Descripción del patrón.

6.2.1 Soporte de ASP.NET/C#

En el patrón anterior se menciona el soporte para comunicación remota. Para este caso, este marco incluye la clase *Provider Model* que se puede extender para obtener un beneficio similar al que se obtiene implementando el patrón *Service Locator*.

El *Provider Model* es un simple contrato entre el cliente y la lógica de negocio o la capa de abstracción de datos [Msdn 09]. Si en ASP.NET se requiere una implementación más flexible de este patrón se puede construir uno propio o utilizar uno existente que se puede descargar de *Codeplex* [Codeplex-Locator 09].

6.2.2 Soporte de PHP

Se han mencionado los límites que tiene PHP para comunicación remota con objetos. La comunicación remota en PHP casi se limita a SOAP si se quieren usar extensiones soportadas por la distribución oficial.

Este patrón entonces no se puede implementar en PHP. Otra limitación es que PHP no posee un API como *JNDI* de Java para la localización de servicios. Sin embargo, se podría realizar una implementación básica de este patrón para la localización de servicios como conexiones a bases de datos, servicios web, LDAP. La implementación se tendría que hacer manual y se tendría que construir un API similar a la que provee *JNDI*.

PHP es una tecnología que por sí misma no permite compartir valores entre diferentes *Scripts* que estén ejecutándose en el servidor. ASP.NET en cambio, incluye un objeto *Application* que permite compartir datos entre toda la aplicación. Compartir datos entre *Scripts* es importante para gestionar un caché con servicios que ya antes han sido localizados.

6.2.3 Soporte de Zend Framework

La limitación de PHP para comunicación remota con objetos se refleja también en este marco y no hay soporte. En este marco también se podría hacer una implementación básica para la localización de algunos recursos que no involucren comunicación remota con objetos.

Lenguaje	Solución	Complejidad
ASP.NET/C#	Buena	Menor
PHP	Mala	n/a
PHP y Zend Framework	Mala	n/a

Tabla 6.4. Calificación de los lenguajes.

6.3 Session Façade (N)

Origen	Core J2EE Patterns [Alur 03]
Descripción	Permite exponer componentes de la capa de negocio y servicios, a clientes remotos.
Requisitos de Implementación	- Soporte para creación de interfaces - Soporte para almacenar sesión temporal del cliente - Comunicación remota con objetos (en el caso de agrupar servicios a clientes remotos)
Comentarios	Este patrón tiene una doble finalidad: (i) evitar que los clientes accedan directamente a componentes de la capa de negocio; y (ii) limitar el tráfico de red entre clientes remotos y componentes de negocio y servicios de grano fino. Permite crear una capa de acceso remoto y exponer servicios de aplicación a clientes remotos. La implementación se puede realizar utilizando un EJB de sesión con o sin estado [Alur 03]. Depende del caso en que se esté exponiendo un servicio. Si el cliente termina el proceso con una única invocación, no es necesario utilizar estados de sesión. En otro caso si el cliente debe hacer diferentes invocaciones para terminar el proceso, se requiere el uso de EJB de sesión con estado. El cliente que accede a un objeto del tipo *Session Façade* es normalmente un objeto *Business Delegate* [Alur 03].

Tabla 6.5. Descripción del patrón.

6.3.1 Soporte de ASP.NET/C#

Antes se describía *.NET Remoting* como un marco para gestión remota de objetos. Este marco permite crear objetos con y sin estado [McLean 02].

Normalmente las implementaciones de *Web services* carecen de un estado que se conserve a través de los llamados. ASP.NET permite crear *Web services* con sesión haciendo uso del componente *Session State.* Con estos elementos se puede crear la clase que se comportará como *Session Façade* que es el rol principal de este patrón.

6.3.2 Soporte de PHP y Zend Framework

PHP y Zend Framework carecen de un marco como *.NET Remoting*. Por esto no hay soporte para la implementación de este patrón.

Sin embargo, la extensión *SOAP* de PHP permite persistir objetos a través de una sesión. El método *setPersistence* de la clase *SoapServer* permite activarlo [Gilmore 08]. En Zend Framework se puede utilizar la clase *Zend_Soap*. Así entonces se puede hacer una implementación básica de este patrón.

Lenguaje	Solución	Complejidad
ASP.NET/C#	Buena	Menor
PHP	Mala	n/a
PHP y Zend Framework	Mala	n/a

Tabla 6.6. Calificación de los lenguajes.

6.4 Application Service (N)

Origen	Core J2EE Patterns [Alur 03]
Descripción	Permite centralizar lógica del negocio y apartarla de los objetos de negocio. De esta forma se provee una capa de servicio uniforme.
Requisitos de Implementación	- Soporte para creación de interfaces
Comentarios	Los objetos de negocio no deben tener a cargo demasiada lógica, y más cuando se trata de interactuar con otros objetos de su tipo. Parte de esta lógica se puede escribir en el *Application Service*, de esta forma se tendrán objetos de negocio de mejor reusabilidad. Desde el servicio de aplicación se puede acceder a los objetos de negocio e incluso a los objetos *Data Access Object.* Esta implementación se puede realizar utilizando el patrón *Command* [Gamma 94]. Sin embargo Alur recomienda usar el patrón *Strategy* [Gamma 94] que permite implementar múltiples variaciones de un mismo servicio.

Tabla 6.7. Descripción del patrón.

6.4.1 Soporte de ASP.NET/C#, PHP y Zend Framework

Este patrón de diseño no se encuentra disponible en las tecnologías analizadas. *Application Service* es un concepto elemental que debe implementar el desarrollador para encapsular lógica de aplicación. Las interfaces que son el único requisito, son soportadas en las tecnologías expuestas.

Con independencia de la complejidad de cada servicio de aplicación, en lo referente a los requisitos asociados a tecnologías específicas, se debe escribir poco código para implementar este patrón en cualquiera de las tres tecnologías.

Lenguaje	Solución	Complejidad
ASP.NET/C#	Buena	Menor
PHP	Buena	Menor
PHP y Zend Framework	Buena	Menor

Tabla 6.8. Calificación de los lenguajes.

6.5 Business Object (N)

Origen	Core J2EE Patterns [Alur 03]
Descripción	Permiten separar datos del negocio y lógica usando un modelo de objetos.
Requisitos de Implementación	- Conexión con mecanismos de persistencia - Carga dinámica (perezosa) de objetos - Soporte de transacciones
Comentarios	En aplicaciones menos complejas, como las *aplicaciones con*

	persistencia que fueron analizadas en el capítulo anterior, se accede directamente a objetos *Data Access Object* desde vistas o componentes del negocio. Esto normalmente se hace cuando el modelo de datos representa aproximadamente el modelo de dominio. En aplicaciones más complejas, donde hay más lógica y relaciones entre objetos de negocio es importante este patrón. Un *Business Object* permite aumentar la reusabilidad y separar lógica para crear un sistema de fácil mantenimiento. Los objetos del negocio implementan una capa reutilizable de entidades del negocio que describen un modelo de dominio [Alur 03]. En el capítulo anterior, en el patrón *Data Access Object* se puede ver el soporte de mecanismos de persistencia de las tecnologías analizadas. Este patrón se encuentra estrechamente relacionado con *Domain Store* que permite persistencia, carga dinámica de otros objetos y gestión de transacciones. Este patrón se puede implementar a través del almacén de dominio (ver sección 6.10) o utilizando frameworks de persistencia avanzados. En cualquier caso, si los objetos del negocio se desvinculan de su persistencia, su complejidad es menor. En otro caso, sería mayor.

Tabla 6.9. Descripción del patrón.

6.5.1 Soporte de ASP.NET/C#

Se debe elegir un mecanismo de persistencia, de acuerdo a los soportados por la tecnología, en este caso se podrían utilizar objetos *Data Access Object* combinados con *ADO.NET*.

Este patrón se puede implementar utilizando C# y su programación orientada objetos. Es necesario crear un modelo de objetos relacionados que representarán el comportamiento del negocio.

6.5.2 Soporte de PHP

Se debe elegir uno de los mecanismos de persistencia relacionados con esta tecnología y se utilizarían objetos *Data Access Object*. El soporte de PHP para programación orientada a objetos permite crear los objetos de negocio y sus relaciones.

6.5.3 Soporte de Zend Framework

En este marco de trabajo se deben tener las mismas consideraciones que en PHP. Como se menciona en el análisis del patrón *Data Access Object*, la persistencia se puede implementar más fácil en el DAO a través de las clases que incluye Zend Framework.

Lenguaje	Solución	Complejidad
ASP.NET/C#	Buena	Menor
PHP	Buena	Menor
PHP y Zend Framework	Buena	Menor

Tabla 6.10. Calificación de los lenguajes.

6.6 Composite Entity (N)

Origen	Core J2EE Patterns [Alur 03]
Descripción	Permite implementar objetos de negocio persistentes que estén compuestos por diferentes objetos relacionados.
Requisitos de Implementación	- No hay requisitos adicionales a los requeridos por: o Business Object o Session Façade o Transfer Object
Comentarios	Los objetos del negocio tienen relaciones entre sí que son necesarias para representar correctamente el modelo del dominio. Cuando se accede a objetos remotos se podría generar una sobrecarga de red al transmitir todos los objetos relacionados que pueden ser necesarios para el cliente. Este patrón se encuentra muy relacionado con los EJBs de entidad [Java-EJB 09] en las aplicaciones J2EE. Una entidad compuesta permite encapsular el diseño de la base de datos y usar implementaciones personalizadas de persistencia. Según *Design Patterns* [Gamma 94], el patrón *Composite* permite crear objetos en estructuras tipo árbol. Los clientes interactúan con el objeto padre sin distinguir si trabajan con objetos simples o agregados. Sin embargo, en el contexto J2EE, la entidad compuesta (p.e., una *Factura*) se encarga de agregar objetos dependientes (p.e., una *LíneaDeFactura*) que sólo son accesibles a través del objeto compuesto. En las aplicaciones empresariales, los objetos *Composite Entity* no deberían ser expuestos directamente al cliente. Para esto se puede usar un *Session Façade* [Alur 03]. Para hacer transferencia de datos se usa el patrón *Transfer Object*. El patrón *Lazy Load* [Fowler 02] es utilizado por *Composite Entity* para cargar progresivamente los objetos que se requieren y no hacer uso innecesario de recursos en la aplicación. Al igual que con los objetos del negocio, si se desliga de su mecanismo de persistencia, la implementación de una entidad compuesta no resulta excesivamente compleja.

Tabla 6.11. Descripción del patrón.

6.6.1 Soporte de ASP.NET/C#, PHP y Zend Framework

Este patrón no se encuentra incluido en ninguna de las tecnologías analizadas. El concepto detrás de este patrón consiste en crear un objeto mayor compuesto por otros objetos. Esta solución se puede implementar sin problema en las tres tecnologías. Para comunicación remota se deben tener en cuenta los límites de PHP. Sin embargo, en aplicaciones más pequeñas se podría acceder a una entidad compuesta desde un servicio de aplicación, *Application Service*.

Tanto en ASP.NET/C# como en PHP se puede hacer la implementación de la carga perezosa *Lazy Load* utilizando una estrategia como *Lazy initialization* [Fowler 02]. También se puede

implementar el patrón *Virtual Proxy* [Gamma 93] para resolver el problema de la carga de objetos por demanda.

Lenguaje	Solución	Complejidad
ASP.NET/C#	Media	Mayor
PHP	Media	Mayor
PHP y Zend Framework	Media	Mayor

Tabla 6.12. Calificación de los lenguajes.

6.7 Optimistic Concurrency (N)

Origen	J2EE Design Patterns [Crawford 03]
Descripción	Propone una solución para la concurrencia cuando hay muchos objetos que pueden estar bajo una transacción pero pocas transacciones simultáneas sobre los mismos objetos.
Requisitos de Implementación	- No hay requisitos particulares
Comentarios	Algunas aplicaciones tienen gran cantidad de datos y pocas transacciones afectando los mismos objetos. Estas aplicaciones requieren una buena solución para los problemas de concurrencia que no penalice el rendimiento del sistema y de los usuarios. El bloqueo optimista supone que las posibilidades de colisión son bajas. Si ocurre un problema al guardar la información se le informará al usuario para que corrija los datos. Se requiere conservar la versión del objeto que está modificando un cliente, si la versión original es la misma que la que tiene asignada el cliente entonces no hay problema. En caso de ser diferente, el cliente debe solucionar el problema y guardar la información actualizada.

Tabla 6.13. Descripción del patrón.

6.7.1 Soporte de ASP.NET/C#

Los objetos que controlan concurrencia normalmente representan una entidad en una base de datos. Este marco de trabajo provee un control llamado *SqlDataSource* que permite especificar por medio del asistente de creación si se quiere implementar concurrencia optimista. De esta forma el mismo control se encargará de validar la integridad de la información [Liberty 08]. Si se detectan cambios se lanzará un error que puede ser personalizado.

6.7.2 Soporte de PHP

Este patrón se debe implementar manualmente. Para implementarlo hay diferentes estrategias, una de ellas es que el objeto *Data Access Object* compare todos los campos en el momento de hacer la actualización del registro. Otra solución es que la entidad en la base de datos contenga un campo que especifique la fecha y hora de última modificación. Esta solución requiere escribir más código que en ASP.NET/C#, sin embargo, no es compleja la implementación.

6.7.3 Soporte de Zend Framework

Este marco no provee una solución instantánea como ASP.NET/C#. La solución se debe implementar como se ha descrito para el caso de PHP. En este caso sin embargo, se pueden utilizar las clases que se proveen para interactuar con la capa de datos.

Lenguaje	Solución	Complejidad
ASP.NET/C#	Buena	Menor
PHP	Media	Media
PHP y Zend Framework	Media	Media

Tabla 6.14. Calificación de los lenguajes.

6.8 Pessimistic Concurrency (N)

Origen	J2EE Design Patterns [Crawford 03]
Descripción	Propone una solución para la concurrencia cuando hay muchos usuarios trabajando sobre un conjunto de datos reducido.
Requisitos de Implementación	- No hay requisitos particulares, más allá de un mecanismo de bloqueo de tablas o filas en el sistema de gestión de bases de datos.
Comentarios	La concurrencia pesimista se puede utilizar para controlar acceso a recursos [Crawford 03]. El concepto se centra en obtener un recurso y a su vez un bloqueo sobre el mismo. El bloqueo se mantiene hasta que el usuario finalice la operación de actualización. De esta forma a diferencia de la concurrencia optimista, se podrá garantizar que la información no cambiará mientras el recurso se encuentre bloqueado. El recurso que se bloquea generalmente es un objeto que representa una fila en la entidad de la base de datos. La implementación de este tipo de concurrencia requiere de más cuidado. En las aplicaciones web puede haber operaciones que requieren del uso de más de una interfaz gráfica (varios formularios en diferentes páginas) y que interactúan sobre los mismos objetos. El mecanismo de bloqueo también debe estar preparado para el caso en que un usuario después de obtener un bloqueo, pierda la conexión o intencionalmente apague el ordenador. No se puede permitir que el recurso quede bloqueado indefinidamente.

Tabla 6.15. Descripción del patrón.

6.8.1 Soporte de ASP.NET/C#, PHP y Zend Framework

La solución para este tipo de concurrencia no viene incorporada en ninguna de las tres tecnologías o marcos.

Este patrón se debe implementar entonces manualmente y existen diferentes estrategias. Se pueden bloquear los registros en la base de datos, mientras alguno se encuentre bloqueado por un usuario, sólo podrá ser accedido en modo lectura por los demás [Liberty 08].

Lenguaje	Solución	Complejidad
ASP.NET/C#	Media	Media
PHP	Media	Media
PHP y Zend Framework	Media	Media

Tabla 6.16. Calificación de los lenguajes.

6.9 Version Number (N)

Origen	J2EE Design Patterns [Crawford 03]
Descripción	Permite asociar un número de versión a cada objeto que represente datos. La versión será incrementada cada que haya un cambio en los datos.
Requisitos de Implementación	- Soporte para creación de interfaces
Comentarios	Este patrón está relacionado con la concurrencia optimista y soluciona el mismo problema. Plantea la creación de una interface que sea implementada por las clases que tendrán objetos con control de versión. El número de versión puede ser usado por un objeto *Data Access Object* y la versión puede ser o no persistida dependiendo de la necesidad.

Tabla 6.17. Descripción del patrón.

6.9.1 Soporte de ASP.NET/C#

El problema de concurrencia optimista se puede resolver fácilmente en ASP.NET/C#, tal y como se indica en el patrón *Optimistic Concurrency*. Si se decide implementar esta solución con número de versión se debe escribir el código para soportarlo.

6.9.2 Soporte de PHP

Este patrón debe ser implementado manualmente. Puede ser usado por un objeto *Data Access Object* para controlar si se persisten o no los datos dependiendo si el cambio es válido o no.

6.9.3 Soporte de Zend Framework

Este patrón debe ser implementado manualmente. Como se ha descrito en los patrones anteriores relacionados con concurrencia, se pueden usar las clases que este marco provee para interactuar con la capa de datos.

Lenguaje	Solución	Complejidad
ASP.NET/C#	Buena	Menor
PHP	Media	Media
PHP y Zend Framework	Media	Media

Tabla 6.18. Calificación de los lenguajes.

6.10 Domain Store (I)

Origen	Core J2EE Patterns [Alur 03]
Descripción	Permite separar la persistencia del modelo de objetos.
Requisitos de Implementación	- Soporte para creación de interfaces - Soporte para persistencia de objetos - Gestión de hebras (no estrictamente necesario)
Comentarios	Las aplicaciones empresariales tienen un modelo de objetos complejo que requiere de un mecanismo de persistencia sofisticado. Así, entre otros, los clientes localizados en la capa de negocio tienen que utilizar objetos del negocio que soporten la carga dinámica de objetos, su acceso concurrente, y su participación en transacciones. Todo ello, sin que los objetos del negocio incluyan detalles específicos de persistencia. Precisamente, uno de los objetivos de este patrón es que los objetos de negocio no implementen detalles específicos de persistencia. Las tecnologías analizadas no poseen algo que reemplace a los EJB de entidad [Java-EJB 09]. *Core J2EE Patterns* [Alur 03] plantea que en el caso de no usar estas entidades se debe seleccionar una estrategia de persistencia separada del modelo de objetos. Este es el objetivo principal de *Domain Store*. Crear un marco de trabajo de persistencia es una tarea muy compleja. Este patrón tiene relación con los patrones: *Application Service, Business Object, Transfer Object y Session Facade*. Este patrón permite implementar una capa de persistencia más compleja, en el caso de que las tecnologías analizadas no permiten persistir modelos complejos de objetos (por ejemplo entidades representadas a través de herencia y con relaciones complejas entre sí). Este patrón es combinación de varios patrones de diseño expuestos en *Patterns of Enterprise Application Architecture* [Fowler 02]: *Unit of Work, Data Mapper, Data Gateway, Identity Map, Lazy Load y Query Object.* En la introducción al capítulo 5 se habla de algunos marcos de persistencia, y se recomienda su uso para la implementación de un almacén de dominio. Sin embargo estos marcos no se tienen en cuenta en este análisis.

Tabla 6.19. Descripción del patrón.

6.10.1 Soporte de ASP.NET/C#

Para los casos básicos de persistencia se ha hablado de *ADO.NET*. El marco de trabajo *.NET* incluye además el componente *Serviced Components (COM+)* que permite utilizar servicios como *Just-In-Time (JIT) Activation*, gestión de transacciones y otros [Guest 03, Msdn-COM+ 09].

Sin embargo la implementación de este patrón requiere un diseño propio, pero se recomienda hacerlo utilizando los componentes que proveen el marco de *ASP.NET y .NET.*

Un servicio de aplicación está encargado de crear un objeto *Business Object* y a su vez iniciar el gestor de persistencia que registrará una transacción. Se utiliza un objeto *State Manager* que se encarga de la carga de los atributos (estáticos o dinámicos) del objeto del negocio. Esta implementación utiliza una lista tipo caché en el administrador de persistencia *Persistence Manager*. *State Manager* puede usar un patrón *Identity Map* [Fowler 02], que se puede implementar en las tres tecnologías analizadas.

Finalmente, el *State Manager* se comunica con el DAO para hacer la escritura en disco, el cual y se encargará de crear un *Transfer Object* para entregar los resultados del *Business Object*.

Este gestor de almacenamiento se puede implementar utilizando *ADO.NET,* del mismo modo, este componente posee soporte para gestión de transacciones [Msdn-ADO 09].

Esta implementación requiere la definición de interfaces para especificar un comportamiento común de los objetos de negocio que se deben persistir. También se debe usar el patrón *Factory Method* [Gamma 93] para la creación del gestor de persistencia.

La interacción entre objetos de gestión de almacenamiento debe poseer una carga perezosa *Lazy Load* [Fowler 02], en el patrón *Composite Entity* se menciona que las tecnologías tienen soporte para su creación.

Por otro lado, este marco incluye soporte para hebras a través de las clases del *namespace System.Threading* [Msdn-Threading 09]. La gestión de revisión y recepción de mensajes se debe hacer en un proceso en paralelo para no afectar la ejecución del programa principal.

6.10.2 Soporte de PHP

En PHP también se requiere una solución propia, los patrones que se requieren para la interacción deben estar previamente implementados, como en ASP.NET/C#. La gestión de almacenamiento se debe realizar a través de un objeto *Data Access Object (DAO)*.

Normalmente las transacciones son delegadas a los sistemas RDBMS y en este caso *PDO* de PHP tiene soporte para interactuar con el mecanismo de transacciones del motor de datos [PHP-PDO 09]. Debe existir un objeto *TransactionManager* que permita hacer interacciones con múltiples objetos.

Aunque las hebras no son siempre necesarias para la implementación de este patrón, es necesario aclarar que PHP no tiene soporte para ellas. El entorno de ejecución de las aplicaciones PHP es diferente a J2EE y .NET. La solución para ejecución paralela de procesos es normalmente la creación de procesos de PHP que funcionen en segundo plano. Normalmente estos procesos se crean con *scripts* que se ejecutan desde una consola del sistema operativo.

6.10.3 Soporte de Zend Framework

Este marco incluye una implementación del patrón *Table Data Gateway* [Fowler 02] que está relacionado el gestor de almacenamiento de almacén de dominio. Las clases *Zend_Db_Table_Abstract, Zend_Db_Table_Rowset* y *Zend_Db_Table_Row* son las necesarias para hacer la implementación [Allen 08].

La clase *Zend_Db_Adapter* permite interactuar con diferentes sistemas RDBMS y provee los métodos para controlar las transacciones [ZF-DB 09]. Esta clase es útil en el momento de hacer la implementación del gestor de transacciones.

Este patrón se debe implementar bajo las mismas condiciones que en ASP.NET/C#, teniendo en cuenta los patrones relacionados que son necesarios. Este marco, al igual que PHP, no tiene soporte para gestión de hebras.

Lenguaje	Solución	Complejidad
ASP.NET/C#	Media	Mayor
PHP	Media	Mayor
PHP y Zend Framework	Media	Mayor

Tabla 6.20. Calificación de los lenguajes.

7. Aplicaciones con persistencia, concurrencia y mensajes

El marco .NET de Microsoft posee el componente de mensajería *MSMQ* que ofrece soporte para la implementación de muchos patrones relacionados con este tipo de aplicaciones [Msdn-MSMQ 09].

En el caso de PHP, no existe un componente con tal potencia, aunque existen algunos proyectos terceros como *Dropr* [Dropr 09], *Simple Asynchronous Messaging for PHP (SAM)* [SAM 09] y una herramienta de mensajería llamada *PHPMQ* que permite usar métodos JMS para envío y recepción de mensajes [PHPMQ 09].

En este análisis, estas librerías o extensiones de PHP serán irrelevantes, pues son proyectos obsoletos sin soporte en la actualidad. En el caso de *PHPMQ*, por ejemplo, su última versión fue liberada en 2004 y el desarrollo es realizado utilizando una extensión *experimental* de PHP que permite integrar código Java en PHP [PHP-Java 09].

7.1 Service Activator (I)

Origen	Core J2EE Patterns [Alur 03]
Descripción	Permite recibir solicitudes asíncronas e invocar servicios del negocio.
Requisitos de Implementación	- Soporte para creación de interfaces - Soporte para servicios de mensajería - Soporte para manejo de excepciones
Comentarios	En las aplicaciones empresariales se puede realizar procesamiento de manera asincrónica, un cliente puede solicitar un servicio y esperar su respuesta. Sin embargo, algunos servicios pueden tardar mucho porque incluyen una actividad compleja de procesamiento, o interacción con otras aplicaciones [Alur 03]. Precisamente por eso, J2EE dispone de los EJBs de mensaje, que a través de un servicio de mensajería, permiten ser invocados mediante mensajes asíncronos. La intención es que los clientes invoquen un servicio y no tengan que esperar hasta que finalice. La solución es que el servidor envíe un mensaje de notificación al cliente cuando haya finalizado el procesamiento. Después de que el servidor finalice el procesamiento enviará al cliente los resultados, o un manejador de resultados. En caso de error, debe enviar información del error. Un cliente podría agregar un identificador único a la petición que será devuelta en las respuestas. Este patrón es implementado como un servidor de mensajes que a su vez delega servicios. *Service Activator* procesará el mensaje e invocará los servicios requeridos para procesar la solicitud. Los servicios invocados por *Service Activator* pueden ser *Business Object, Session Façade o Application Services.* Este patrón tiene un comportamiento similar al patrón *Message Façade* [Crawford 03].

Tabla 7.1. Descripción del patrón.

7.1.1 Soporte de ASP.NET/C#

Este marco de trabajo cuenta con el componente *MSMQ* que permite establecer comunicación entre aplicaciones y componentes de ellas mismas a través de mensajes [Guest 03]. El marco de *.NET* ofrece varios objetos que permiten la interacción con *MSMQ* [Msdn-MSMQ 09].

Actualmente, los *Web services* ofrecen una gran oportunidad para interoperabilidad entre diferentes arquitecturas. *Microsoft* en conjunto con *IBM, BEA* y *TIBCO* han elaborado la especificación *WS-ReliableMessaging*, que define un protocolo para la comunicación con mensajes entre dos puntos a través de SOAP [Guest 03]. Esta opción crea posibilidades interactuar con otras plataformas.

ASP.NET/C# posee el soporte necesario para la implementación de este patrón de diseño gracias a estos componentes integrados. Antes se ha mencionado también, que C# tiene soporte para interfaces y excepciones.

7.1.2 Soporte de PHP y Zend Framework

PHP carece de un sistema de mensajes para comunicación entre aplicaciones. Esta es una limitación grande de este lenguaje en términos de interoperabilidad. PHP posee unos métodos para crear y gestionar colas de mensajes [PHP-Msg 09]. Sin embargo, este servicio de mensajería se limita al uso en la misma máquina.

Existe un proyecto de mensajería llamado *Dropr* [Dropr 09] pero todavía se encuentra en desarrollo. Existe otra iniciativa llamada *Simple Asynchronous Messaging for PHP (SAM)* [SAM 09]. Sin embargo *SAM* es un proyecto que ha estado olvidado desde el 2007. Existen otros proyectos que también carecen de mantenimiento en la actualidad, algunos, por ejemplo, para usar JMS desde PHP [PHP-JMS 09].

Para la construcción de aplicaciones empresariales se buscan librerías con soporte y en versiones estables. La construcción de un sistema de gestión de mensajes implicaría un trabajo titánico. Por lo tanto PHP y Zend Framework no tienen soporte para la implementación de este patrón.

Lenguaje	Solución	Complejidad
ASP.NET/C#	Buena	Menor
PHP	Mala	n/a
PHP y Zend Framework	Mala	n/a

Tabla 7.2. Calificación de los lenguajes.

7.2 Web Service Broker (I)

Origen	Core J2EE Patterns [Alur 03]
Descripción	Permite ofrecer acceso a uno o varios servicios utilizando XML y protocolos web.
Requisitos de Implementación	- Soporte para creación de interfaces - Soporte para creación de *web services* - Procesamiento de contenido XML
Comentarios	Las aplicaciones empresariales necesitan tener compatibilidad para interactuar con otros sistemas. Para esto se exponen servicios de la aplicación en forma de *web services*, estos servicios se pueden

	extraer de diferentes partes de la aplicación. Normalmente sólo se quieren hacer públicos algunos servicios de la aplicación. Este patrón de diseño permite reusar y exponer servicios a los clientes utilizando estándares abiertos que permitan integración con otras aplicaciones. El objeto *Web Service Broker* puede acceder a objetos y servicios de la aplicación como *Session Façade* y *Application Service*.

Tabla 7.3. Descripción del patrón.

7.2.1 Soporte de ASP.NET/C#

La creación de *web services* en ASP.NET/C# es sencilla. Se debe crear el servicio que extienda de la clase *System.Web.Services* que provee este marco. La implementación se puede realizar en C# y se debe asignar una extensión *.asmx* al fichero [Allen 02].

Se permite utilizar el atributo *WebMethod* en la descripción de los métodos. Este atributo permite especificar algunas propiedades como activación de transacciones, gestión de estado de sesión y buffer [Allen 02].

Este marco de trabajo genera automáticamente el *WSDL* con la descripción del servicio. Los detalles de persistir tipos de datos en paquetes *SOAP* están ocultos para los desarrolladores [Allen 02]. Estas herramientas permiten implementar este patrón de diseño sin complicaciones.

7.2.2 Soporte de PHP

PHP trae integrada la extensión *SOAP* que permite crear servicios para el transporte de mensajes. En PHP se pueden crear servicios *SOAP* con o sin *WSDL*. Se pueden agregar métodos dinámicamente al servidor *SOAP*, o agregar una clase que ofrecerá todos sus métodos como servicio web [Coggeshall 04]. La descripción *WSDL* del servicio debe realizarse manualmente. Sin embargo, si se está usando un editor como *Zend Studio for Eclipse* se pueden generar estos ficheros con una interfaz gráfica amigable [Zend-WSDL 09].

Este lenguaje incluye una extensión llamada *SimpleXML* que provee un API simple para la manipulación de datos XML [Coggeshall 04]. Existe también la extensión *XML-RPC* para llamar procedimientos remotos utilizando el protocolo *HTTP* como transporte y *XML* para la representación de datos. De acuerdo a esta información se puede concluir que hay soporte para la implementación de *Web Service Broker* en este lenguaje.

7.2.3 Soporte de Zend Framework

Este marco de trabajo se basa en las mismas extensiones que provee PHP aunque tiene sus propias clases que ofrecen un API aun más sencilla para la creación de estos servicios. *Zend_Soap_Server* y *Zend_Soap_Client* son dos de las clases más importantes. Este marco también provee la clase *Zend_Rest* que ha sido creada para ofrecer servicios web tipo *REST* [Allen 08].

El componente *Zend_XmlRpc* de este marco ofrece soporte para ofrecer y consumir servicios remotos XML-RPC. Esta tecnología también permite implementar este patrón de diseño.

Lenguaje	Solución	Complejidad
ASP.NET/C#	Buena	Menor
PHP	Buena	Menor
PHP y Zend Framework	Buena	Menor

Tabla 7.4. Calificación de los lenguajes.

7.3 Point-to-Point Distribution (I)

Origen	J2EE Design Patterns [Crawford 03]
Descripción	Permite enviar mensajes de un actor a otro. Es un método de distribución simple, el mensaje es enviado una sola vez y procesado una sola vez.
Requisitos de Implementación	- Soporte para creación de interfaces - Soporte para servicios de mensajería
Comentarios	En algunos casos es útil que entre los objetos de una aplicación se compartan mensajes, o que se envíen mensajes de un sistema a otro. Estos mensajes son puestos en una cola de mensajes para que el destinatario lo descargue y envíe un mensaje de confirmación. Los lenguajes deben poseer un servicio de mensajería que permita poner y leer mensajes de una cola.

Tabla 7.5. Descripción del patrón.

7.3.1 Soporte de ASP.NET/C#

Ya se ha hablado que este marco cuenta con *MSMQ* para la gestión de mensajes entre aplicaciones y componentes internos de una aplicación. Utilizando este servicio se puede usar una cola e implementar este patrón de diseño sin problema.

7.3.2 Soporte de PHP y Zend Framework

No hay soporte. La limitación de PHP en el uso de servicios de mensajería imposibilita la implementación de este patrón de diseño cuando se trata de aplicaciones en diferentes servidores. Si por ejemplo se requiere compartir mensajes entre la misma aplicación (entiéndase mismo servidor) se podrían usar los métodos nativos de PHP para gestión de colas de mensajes [PHP-Msg 09].

Lenguaje	Solución	Complejidad
ASP.NET/C#	Buena	Menor
PHP	Mala	n/a
PHP y Zend Framework	Mala	n/a

Tabla 7.6. Calificación de los lenguajes.

7.4 Publish-Subscribe (I)

Origen	J2EE Design Patterns [Crawford 03]
Descripción	Permite enviar mensajes a múltiples destinatarios usando un esquema que facilita el mantenimiento y la escalabilidad.
Requisitos de Implementación	- Soporte para creación de interfaces - Soporte para servicios de mensajería
Comentarios	En algunas aplicaciones empresariales se debe enviar el mismo mensaje o algunos similares a múltiples destinatarios. Utilizando el esquema de *Point-to-Point Distribution* se podrían enviar mensajes en una iteración a múltiples destinatarios. Eso trae consigo

	problemas de escalabilidad porque se deben conocer los destinatarios y además preocuparse por el envío de cada uno de los mensajes. Se puede crear un canal de mensajes para que los clientes se suscriban y vean los mensajes que allí se publican.

Tabla 7.7. Descripción del patrón.

7.4.1 Soporte de ASP.NET/C#

Este patrón se puede implementar en este marco de trabajo utilizando un objeto del tipo *TcpChannel*, y registrando los canales con *ChannelServices.RegisterChannel* [McLean 02]. Así se pueden enviar mensajes y los suscriptores se podrán leer mensajes de este canal.

7.4.2 Soporte de PHP y Zend Framework

A diferencia del marco de Microsoft, PHP y Zend Framework no permiten implementar este patrón con la misma facilidad. Para hacerlo se pueden utilizar sockets TCP para compartir mensajes entre los diferentes actores [PHP-Socket 09]. Los clientes se pueden conectar a un socket para leer los mensajes.

Lenguaje	Solución	Complejidad
ASP.NET/C#	Buena	Menor
PHP	Media	Mayor
PHP y Zend Framework	Media	Mayor

Tabla 7.8. Calificación de los lenguajes.

7.5 Malformed Message Channel (I)

Origen	J2EE Design Patterns [Crawford 03]
Descripción	Permite identificar los mensajes que contienen errores y guardarlos en una cola de mensajes mal formados.
Requisitos de Implementación	- Soporte para servicios de mensajería
Comentarios	Las aplicaciones reciben mensajes y deben procesar su contenido. En algunos casos el contenido de los mensajes está mal formado y la aplicación detectará fácilmente que no puede procesar el mensaje. Una aplicación fiable debe proveer un mecanismo para manejar condiciones de error y guardar mensajes. Este patrón permite especificar un destino al cual se pueden redireccionar los mensajes erróneos. Esto permitirá a un administrador del sistema revisar manualmente y determinar la causa de un error [Crawford 03]. Al implementar *Malformed Message Channel* se podría agregar algún comportamiento en aplicaciones críticas, como enviar una alerta para tomar una acción correctiva.

Tabla 7.9. Descripción del patrón.

7.5.1 Soporte de ASP.NET/C#

Antes se ha mencionado que el marco *.NET* trae integrado el servicio *MSMQ* para la gestión de mensajería y colas. Se debe crear una clase que provea los métodos para interceptar los mensajes recibidos y determinar si son procesables o no. Con *MSMQ* se podría crear una nueva cola de mensajes erróneos y así redireccionar allí los mensajes no procesables [Msdn-MSMQ 09].

7.5.2 Soporte de PHP y Zend Framework

Debido a que estas tecnologías no tienen soporte para mensajería, este patrón no puede ser implementado.

Lenguaje	Solución	Complejidad
ASP.NET/C#	Buena	Media
PHP	Mala	n/a
PHP y Zend Framework	Mala	n/a

Tabla 7.10. Calificación de los lenguajes.

7.6 Message Selector (I)

Origen	J2EE Design Patterns [Crawford 03]
Descripción	Permite procesar diferentes tipos de mensajes que se encuentren en la misma cola.
Requisitos de Implementación	- Soporte para creación de interfaces - Soporte para servicios de mensajería
Comentarios	Es normal que en grandes aplicaciones haya muchos tipos de mensajes para procesar. En algunos diseños se opta por crear colas separadas para cada tipo de mensaje. Sin embargo, la administración de diferentes colas o canales puede convertirse en una tarea muy complicada. Para implementar este patrón, la tecnología debe poseer un mecanismo para seleccionar mensajes de una cola dado un criterio de selección. Si no existe, debe poderse implementar.

Tabla 7.11. Descripción del patrón.

7.6.1 Soporte de ASP.NET/C#

A diferencia de *JMS* de J2EE, .NET no incluye los selectores en su API, para implementar este patrón se debe implementar el selector de mensajes desde cero utilizando los servicios que se proveen en este marco para la gestión de colas.

7.6.2 Soporte de PHP y Zend Framework

No hay soporte. En el caso de estarse utilizando la cola de mensajes nativa de PHP para aplicaciones dentro de una misma máquina [PHP-Msg 09], se pueden agregar/leer mensajes de la cola especificando un tipo.

Lenguaje	Solución	Complejidad
ASP.NET/C#	Media	Media
PHP	Mala	n/a
PHP y Zend Framework	Mala	n/a

Tabla 7.12. Calificación de los lenguajes.

7.7 Competing Consumers (I)

Origen	J2EE Design Patterns [Crawford 03]
Descripción	Permite procesar diferentes mensajes en paralelo.
Requisitos de Implementación	- Soporte para creación de interfaces - Soporte para servicios de mensajería
Comentarios	Cuando se conectan aplicaciones distribuidas, la mensajería es una buena solución para que un usuario continúe interactuando con el sistema incluso cuando se estén realizando operaciones complejas [Crawford 03]. Se pueden establecer varios consumidores que competirán para ganar el derecho a procesar los mensajes de la cola. Así cada consumidor lee el mensaje tan pronto como puede y lo procesa; si la demanda aumenta, simplemente se agregan más consumidores. La implementación de este patrón se puede realizar sin un servicio *middleware* orientado a mensajes *(MOM)*, pero esta implementación suele ser complicada [Crawford 03].

Tabla 7.13. Descripción del patrón.

7.7.1 Soporte de ASP.NET/C#

Con el servicio *MSMQ* que provee *Microsoft* en su marco se puede implementar este patrón de diseño sin complicaciones. La cola es responsable de garantizar que cada mensaje es entregado únicamente a un consumidor [Crawford 03].

7.7.2 Soporte de PHP y Zend Framework

Este patrón requiere el uso de servicios de mensajería, por lo tanto no hay soporte de estas tecnologías.

Para una aplicación básica se podría implementar en estas tecnologías sólo en el caso en que la cola de mensajes se encuentre en un servidor de correo. Cuando se está implementando *Competing Consumers* accediendo a un servidor de correo se debe tener en cuenta que no se puede tener más de una aplicación accediendo al mismo buzón de correo electrónico [Crawford 03]. PHP tiene soporte para conexión con servidores *POP3, IMAP* y *NNTP* [Wyke 00].

Zend Framework cuenta con un componente *Zend_Mail* [ZF-Mail 09] que permite hacer conexión con servicios de correo utilizando menos código fuente.

Lenguaje	Solución	Complejidad
ASP.NET/C#	Buena	Media
PHP	Mala	n/a
PHP y Zend Framework	Mala	n/a

Tabla 7.14. Calificación de los lenguajes.

7.8 Event-Driven Consumer (I)

Origen	J2EE Design Patterns [Crawford 03]
Descripción	Permite construir un mecanismo para que los mensajes se procesen de acuerdo a solicitudes del entorno.
Requisitos de Implementación	- Soporte para creación de interfaces - Soporte para servicios de mensajería
Comentarios	Este patrón permite crear un sistema de procesamiento de mensajes que responde a eventos. Al implementarlo se simplifica el trabajo que se desarrolla con el patrón *Polling Consumer*. Al utilizar *Event-Driven Consumer* se obtendrá como resultado un código menos complejo y más reusable. En el caso de *EJB 3.0* de *J2EE*, existe soporte para *message-driven beans* que permite implementar un método al que el servidor llama cada que detecta un nuevo mensaje.

Tabla 7.15. Descripción del patrón.

7.8.1 Soporte de ASP.NET/C#

En este marco de trabajo se puede realizar la implementación de este patrón, sin embargo, no hay un mecanismo que lo permita hacerlo de la misma forma que se puede hacer en J2EE. Una forma de resolverlo es creando una cola de mensajes asíncronos utilizando los métodos *BeginReceive()* y *EndReceive()* de *System.Messaging.MessageQueue*; se debe implementar un manejador de eventos *System.Messaging.receiveCompleteEventHandler* para notificar cuando el mensaje haya sido recibido [Fisher 06]. De esta forma se podría llamar un método común de los objetos que requieren recibir mensajes asíncronos.

7.8.2 Soporte de PHP y Zend Framework

No es soportado debido a la carencia de un sistema de mensajería.

Lenguaje	Solución	Complejidad
ASP.NET/C#	Buena	Media
PHP	Mala	n/a
PHP y Zend Framework	Mala	n/a

Tabla 7.16. Calificación de los lenguajes.

7.9 Message Façade (I)

Origen	J2EE Design Patterns [Crawford 03]
Descripción	Permite crear un objeto que recibe mensajes asincrónicamente de una fuente externa. Los mensajes son convertidos en llamadas a métodos de la lógica del negocio y opcionalmente se pueden enviar resultados de vuelta. Es apropiado para recibir mensajes de la capa de presentación.
Requisitos de Implementación	- Soporte para creación de interfaces - Soporte para servicios de mensajería

Comentarios	Las diferentes rutas a *Servlets*, *JMS*, *webservices* y más pueden ser desconocidas. Esas rutas a la lógica del negocio pueden ser encapsuladas a través de una fachada, esto tiene ventajas de reusabilidad. La lógica del negocio puede ser usada asincrónicamente desde la capa de presentación con el coste de escribir más código y requerir una mejor infraestructura [Crawford 03]. Este patrón tiene una relación con *Service Activator* [Alur 03].

Tabla 7.17. Descripción del patrón.

7.9.1 Soporte de ASP.NET/C#

Este patrón se puede implementar en este marco de trabajo utilizando los servicios de mensajería. La fachada recibe los mensajes por ejemplo utilizando otro patrón como *Event-Driven Consumer*. De esta forma procesa la acción y opcionalmente se retorna información al cliente.

7.9.2 Soporte de PHP y Zend Framework

Este patrón no es soportado en estas tecnologías.

Lenguaje	Solución	Complejidad
ASP.NET/C#	Buena	Media
PHP	Mala	n/a
PHP y Zend Framework	Mala	n/a

Tabla 7.18. Calificación de los lenguajes.

7.10 Message Handler (I)

Origen	J2EE Design Patterns [Crawford 03]
Descripción	Permite separar la lógica encargada de recibir el mensaje de la lógica encargada de procesar el contenido.
Requisitos de Implementación	- Soporte para creación de interfaces - Soporte para servicios de mensajería
Comentarios	En grandes aplicaciones, tener un sistema de recepción de mensajes y de procesamiento en el mismo punto afecta negativamente la escalabilidad y mantenibilidad del sistema. Al implementar un *Message Handler* se permitirá cambiar la forma en que se reciben mensajes sin alterar código encargado del procesamiento. La ejecución de pruebas al sistema también será más sencilla, debido a que se ejecutarán pruebas a cada componente y no a un gran componente integrado [Crawford 03].

Tabla 7.19. Descripción del patrón.

7.10.1 Soporte de ASP.NET/C#

En este marco de trabajo se puede implementar este patrón utilizando las interfaces que son soportadas en C# y los servicios de mensajería de *MSMQ* [Msdn-MSMQ 09].

7.10.2 Soporte de PHP y Zend Framework

No hay soporte para servicios de mensajería.

Lenguaje	Solución	Complejidad
ASP.NET/C#	Buena	Media
PHP	Mala	n/a
PHP y Zend Framework	Mala	n/a

Tabla 7.20. Calificación de los lenguajes.

7.11 Polling Consumer (I)

Origen	J2EE Design Patterns [Crawford 03]
Descripción	Permite revisar por mensajes periódicamente en vez de mantener una conexión constante con el servidor de mensajes.
Requisitos de Implementación	- Soporte para servicios de mensajería - Soporte para creación de interfaces - Gestión de hebras
Comentarios	Cuando se intentan recibir mensajes y el servidor de mensajes no se encuentra disponible, simplemente se reintenta la operación en otro momento. En el caso en que un cliente tenga un fallo y deba reiniciar su entorno de ejecución, el servidor mantendrá los mensajes y no se pierde información. Este patrón permite recibir mensajes de correo de Internet de una forma muy fiable [Crawford 03].

Tabla 7.21. Descripción del patrón.

7.11.1 Soporte de ASP.NET/C#

Los servicios de mensajería de este marco de trabajo no incluyen soporte nativo para *POP3* e *IMAP*. En el caso de requerirse consumir mensajes de un servidor de correo se necesita desarrollar un servicio que utilice el protocolo *TCP* para estas conexiones [Albahari 07].

Como se menciona en el patrón *Domain Store*, esta tecnología tiene soporte para gestión de hebras.

7.11.2 Soporte de PHP y Zend Framework

No hay soporte para la implementación de este patrón.

Lenguaje	Solución	Complejidad
ASP.NET/C#	Media	Mayor
PHP	Mala	n/a
PHP y Zend Framework	Mala	n/a

Tabla 7.22. Calificación de los lenguajes.

7.12 Content Aggregator (I)

Origen	J2EE Design Patterns [Crawford 03]
Descripción	Permite modificar el contenido de los mensajes para construir un formato común y redirigirlos al punto de procesamiento que puede ser otro servidor.
Requisitos de Implementación	- Soporte para creación de interfaces - Soporte para servicios de mensajería
Comentarios	Este patrón puede ser usado en combinación con *Content-Based Routing*. Cuando un sistema es alimentando con mensajes de diferentes fuentes, los mensajes son enviados a un *Aggregator* encargado de transformarlos y redirigirlos [Crawford 03].

Tabla 7.23. Descripción del patrón.

7.12.1 Soporte de ASP.NET/C#

Este patrón puede implementarse en esta tecnología usando *MSMQ*. Con este patrón se pueden transformar mensajes de todo tipo. Para el caso de XML se pueden usar los componentes de este marco para el manejo de estos datos.

7.12.2 Soporte de PHP y Zend Framework

No hay soporte por las limitaciones conocidas con mensajería. En entornos en los que se usen peticiones con *webservices* desde múltiples fuentes, podría usarse este patrón.

Lenguaje	Solución	Complejidad
ASP.NET/C#	Media	Mayor
PHP	Mala	n/a
PHP y Zend Framework	Mala	n/a

Tabla 7.24. Calificación de los lenguajes.

7.13 Pipes and Filters (I)

Origen	J2EE Design Patterns [Crawford 03]
Descripción	Permite enlazar diferentes *Message Handlers* en una tubería para el procesamiento de los mensajes.
Requisitos de Implementación	- Soporte para creación de interfaces - Soporte para servicios de mensajería
Comentarios	En las aplicaciones empresariales, el procesamiento de mensajes puede ser realizado por un conjunto de sistemas que se encuentren el mismo sitio o en localizaciones geográficas diferentes. Este patrón ha sido usado en otros contextos y el patrón como tal es antiguo [Crawford 03]. Cuando se implementa cada etapa de la gestión del mensaje como un proceso independiente, se obtiene un buen nivel de flexibilidad para modificar el orden de ejecución o

	agregar nuevos procesos intermedios. También se podrán ejecutar pruebas del sistema a cada proceso independientemente.

Tabla 7.25. Descripción del patrón.

7.13.1 Soporte de ASP.NET/C#

Este marco provee soporte para la implementación de este patrón de diseño. En un ejemplo del sitio oficial [Msdn Pipes-Filters 09] hay algunas recomendaciones para la implementación.

7.13.2 Soporte de PHP y Zend Framework

No hay soporte para servicios de mensajería.

Lenguaje	Solución	Complejidad
ASP.NET/C#	Buena	Media
PHP	Mala	n/a
PHP y Zend Framework	Mala	n/a

Tabla 7.26. Calificación de los lenguajes.

7.14 Content – Based Routing (I)

Origen	J2EE Design Patterns [Crawford 03]
Descripción	Provee un punto único para la recepción de mensajes. Redirige el mensaje basado en su contenido al sistema encargado de hacer el procesamiento.
Requisitos de Implementación	- Soporte para creación de interfaces - Soporte para servicios de mensajería
Comentarios	Este patrón permite que se puedan agregar nuevos servidores de procesamiento flexiblemente. Los mensajes que no se puedan redirigir a otro servidor pueden ser guardados utilizando un patrón como *Malformed Message Channel*. Este patrón también se puede usar en combinación con un *Service Activator*.

Tabla 7.27. Descripción del patrón.

7.14.1 Soporte de ASP.NET/C#

Este marco proporciona el soporte necesario para la implementación de este patrón. Con los servicios de mensajería *MSMQ* y *C#* es suficiente para implementar una clase *ContentBasedRouter* [Hohpe 03] que se encargue de recibir los mensajes, leer su contenido y redirigirlos.

7.14.2 Soporte de PHP y Zend Framework

No hay soporte de estas tecnologías. Si se están recibiendo los mensajes a través de un servidor de correo se podría implementar este patrón para lograr el mismo resultado.

Lenguaje	Solución	Complejidad
ASP.NET/C#	Buena	Media
PHP	Mala	n/a
PHP y Zend Framework	Mala	n/a

Tabla 7.28. Calificación de los lenguajes.

7.15 Control Bus (I)

Origen	J2EE Design Patterns [Crawford 03]
Descripción	Permite realizar actividades de monitorización de un bus de mensajes en el que puede haber varias aplicaciones involucradas.
Requisitos de Implementación	- Soporte para creación de interfaces - Soporte para servicios de mensajería
Comentarios	Este patrón facilita el proceso monitorización y permite identificar si un sistema se encuentra funcionando. Existen otros beneficios como identificar si hay retardos, si un canal se está llenando, etc. Los mensajes podrían utilizarse de hecho para enviar cambios en la configuración del sistema, esto involucra una gestión más estricta de seguridad [Hohpe 03].

Tabla 7.29. Descripción del patrón.

7.15.1 Soporte de ASP.NET/C#

Los servicios de *MSMQ* permiten implementar este patrón de diseño. Este patrón involucra la creación de uno o más canales para la transmisión de mensajes entre los diferentes componentes o aplicaciones.

7.15.2 Soporte de PHP y Zend Framework

No hay soporte en estas dos tecnologías.

Lenguaje	Solución	Complejidad
ASP.NET/C#	Media	Mayor
PHP	Mala	n/a
PHP y Zend Framework	Mala	n/a

Tabla 7.30. Calificación de los lenguajes.

8. Conclusiones y trabajo futuro

8.1 Conclusiones

Este trabajo presenta aportaciones en dos direcciones. La primera, realizar un mapping entre las ontologías de patrones inducidas por los libros *Core J2EE Patterns* [Alur 03] y *J2EE Design Patterns* [Crawford 03]. Al realizar este mapping, o equivalencia, se ha constatado que los patrones propuestos por Alur et al. son mucho más concretos y usables en la práctica. Esta conclusión no es casual, ya que estos patrones provienen de la experiencia en aplicaciones web de *Sun Microsystems*. Por ejemplo, cabe destacar la ausencia del patrón *Application Service* en el catálogo propuesto por Crawford y Kaplan. Eso sí, también debemos mencionar que el catálogo propuesto por Crawford y Kaplan incluye patrones de mensajería, casi inexistentes en el catálogo propuesto por Alur et al.

La segunda gran aportación se deriva de analizar los diferentes patrones por tipo de aplicación, ya que se ha logrado determinar la medida en la que cada una de las tecnologías puede dar soporte al desarrollo de los diferentes tipos de aplicación web. Además, el trabajo aporta la definición de una jerarquía de aplicaciones web para hacer el análisis.

Para establecer una diferencia global se presenta la Tabla 8.1, en la que se puede ver el alcance de cada tecnología con respecto a cada tipo de aplicación. Los porcentajes corresponden a la cantidad de patrones de diseño de alto nivel que permite implementar cada tecnología. Como base para la comparación se incluye la tecnología J2EE en la que se basan los patrones de los libros usados en este trabajo.

Aplicación/Tecnología	**J2EE**	**ASP.NET/C#**	**PHP**	**Zend F.**
Aplicaciones con persistencia	100%	100% (12/12)	100% (12/12)	100% (12/12)
Aplicaciones con persistencia y concurrencia	100%	100% (22/22)	86% (19/22)	86% (19/22)
Aplicaciones con persistencia, concurrencia y mensajes	100%	100% (37/37)	56% (21/37)	56% (21/14)
Total con base a todos los patrones	100%	100%	56%	56%

Tabla 8.1. Porcentaje de soporte.

ASP.NET/C# es parte del marco de trabajo *.NET*. Esta tecnología tiene el soporte de todo el marco de *Microsoft* que está orientado a la construcción de sistemas empresariales con interoperabilidad. Esta tecnología se aproxima a *J2EE* y puede ser comparada directamente con ella. Sin embargo *J2EE* es una tecnología que se destaca como superior basada en la seguridad, escalabilidad, fiabilidad y la posibilidad de usar entornos de ejecución en múltiples plataformas [Fisher 06].

Por el contrario PHP, se encuentra un escalón más abajo que ASP.NET, y por supuesto que J2EE. En el capítulo 6, se ha expuesto que PHP no tiene soporte integrado para acceso a objetos remotos, lo que restringe la implementación de algunos patrones de diseño de alto nivel que son usados en aplicaciones empresariales. Además, en el capítulo 7 se ha expuesto la carencia de un servicio integrado de mensajería en PHP. Esto hace imposible la implementación de la mayoría de patrones que están relacionados con la interoperabilidad entre diferentes plataformas.

PHP y el marco de Zend se encuentran adecuados para la implementación de cierto tipo de aplicaciones. Se establece un énfasis en aquellas que no requieran interoperabilidad.

Se ha visto que PHP tiene un soporte adecuado para la definición y uso de *web services*. Esto permite a las aplicaciones PHP compartir datos con servicios externos, y se pueden implementar patrones de alto nivel

relacionados con *web services*. Por tanto, una aplicación web desarrollada en PHP o Zend Framework podrá estar en contacto con otros sistemas, pero utilizando mecanismos como SOAP.

Para definir una tecnología a usar en el momento de construir una aplicación web, se deben tener en cuenta las ventajas y desventajas que puede poseer cada una. Del mismo modo, se debe estudiar el alcance de la aplicación web a desarrollar. J2EE ofrece soporte para implementar todos los patrones de alto nivel que se analizan en este trabajo, sin embargo su entorno de ejecución es más pesado y tiene por ejemplo una curva de aprendizaje mayor.

Los patrones de diseño que se han analizado en este trabajo pueden clasificarse de acuerdo a su complejidad, de esta forma se presenta la Tabla 8.3 que contiene algunos de ellos ordenados de mayor a menor complejidad.

Patrón de Diseño	Tipo de Aplicación	Capa	Complejidad	Facilidad de implementación
Domain Store	Persistencia y concurrencia	Integración	Mayor	ASP.NET/C# = 1 PHP = 1 Z. Framework = 1
Messaging Patterns	Persistencia, concurrencia y mensajes	Integración	Mayor	ASP.NET/C# = 4 PHP = 0 Z. Framework = 0
Optimistic Concurrency Pessimistic Concurrency Version Number	Persistencia y concurrencia	Negocio	Mayor	ASP.NET/C# = 5 PHP = 2 Z. Framework = 2
Application Controller	Persistencia	Presentación	Media	ASP.NET/C# = 6 PHP = 1 Z. Framework = 6
Transfer Object Transfer Object Assembler	Persistencia	Negocio	Media	ASP.NET/C# = 6 PHP = 1 Z. Framework = 1
Data Access Object	Persistencia	Integración	Menor	ASP.NET/C# = 6 PHP = 4 Z. Framework = 6
View Helper Composite View Service to Worker	Persistencia	Presentación	Menor	ASP.NET/C# = 6 PHP = 5 Z. Framework = 6
Application Service Business Object	Persistencia y concurrencia	Negocio	Menor	ASP.NET/C# = 6 PHP = 6 Z. Framework = 6

Tabla 8.2. Complejidad de un grupo de patrones.

Es importante destacar que los patrones que se clasifican con alta complejidad son aquellos que requieren bastante codificación y/o carecen de soporte en alguna tecnología. Para implementar estos patrones, se recomienda el apoyo de los diferentes marcos de trabajo que posean las tecnologías.

La complejidad de cada patrón se calcula en base a su facilidad de implementación en las tecnologías analizadas. Con este fin se han asignado equivalencias cuantitativas a cada tipo de calificación. Para calcular la facilidad de implementación del patrón con respecto a la tecnología se hace una multiplicación *S* x *C* (ver Tabla 8.3). Para calcular la facilidad de implementación del patrón se suman las puntuaciones

anteriores para cada tecnología. Las mayores puntuaciones indican que hay más soporte de la tecnología, las menores que hay menos soporte y de paso mayor complejidad para alcanzar la solución. Así, por ejemplo, un patrón con complejidad de 6, será un patrón más sencillo de implementar que un patrón con puntuación de 4, ya que dispondrá de soluciones mejores y/o de complejidades menores por tecnologías de implementación.

Solución *(S)*	**Complejidad *(C)***
Buena = 2	Menor = 3
Media = 1	Media = 2
Mala = 0	Mayor = 1

Tabla 8.3. Equivalencia cuantitativa de calificaciones.

Finalmente, con base en este análisis este trabajo hace recomendaciones tecnológicas de acuerdo a los diferentes tipos de aplicación. La elección de una u otra tecnología depende de los ingenieros de software y el ámbito de cada proyecto. La escalabilidad, mantenibilidad e interoperabilidad son conceptos claros que deben ser tomados en cuenta desde el diseño del sistema. La Tabla 8.4 recoge estas recomendaciones.

Tipo de aplicación	**Recomendaciones**
Persistencia	Si la interoperabilidad con otros sistemas a futuro no es un punto crítico, PHP/Zend Framework es una buena elección. PHP tiene una curva de aprendizaje baja y su entorno de ejecución es multiplataforma. Esta solución es *Open Source* y por tanto no exige inversiones en licenciamiento. En otro caso, para aplicaciones que proyectan a futuro necesidades de interoperabilidad (con mecanismos diferentes a servicios web), se recomienda el uso de ASP.NET/C# o J2EE.
Persistencia y concurrencia	Para estas aplicaciones se debe considerar la misma anotación de interoperabilidad del tipo anterior. Tanto PHP/Zend Framework como ASP.NET/C# y J2EE son tecnologías apropiadas para la construcción de una aplicación web de este tamaño. En este tipo de aplicaciones puede ser necesaria la comunicación con objetos remotos. De ser requerida, ASP.NET/C# o J2EE es más apropiado. Sin embargo, si la comunicación se puede realizar utilizando servicios web, se puede usar PHP/Zend Framework y hacer implementaciones alternativas (es decir, basadas en estos servicios) de *Business Delegate* y *Service Locator*.
Persistencia, concurrencia y mensajes	ASP.NET/C# o J2EE es la solución recomendada para estas grandes aplicaciones empresariales. PHP/Zend Framework desaparece de las candidatas por el hecho de no poseer un buen sistema de mensajería ni soporte para objetos remotos.

Tabla 8.4. Recomendaciones para la jerarquía de aplicaciones

8.2 Trabajo Futuro

Aunque este trabajo proporciona una visión global de la capacidad de cada tecnología, es importante ejemplificar los patrones y profundizar en más conceptos técnicos de cada tecnología. Así, los ingenieros y desarrolladores tendrán una comprensión mejor de los problemas que plantea y resuelve cada patrón. Se sugiere además crear una discusión más extensa sobre las tecnologías en sí. Esta discusión se puede basar en temas tales como: la productividad de cada marco de trabajo, cantidad de documentación en la red, proyectos exitosos, porcentajes de uso a nivel mundial, curvas de aprendizaje, interoperabilidad y otros que se consideren importantes como punto de comparación.

Por último, resultaría muy interesante considerar un cuarto tipo de aplicaciones basado en patrones para servicios web, y llevar a cabo un análisis similar al realizado en este trabajo.

9. Referencias

[Albahari 07]

Joseph Albahari, Ben Albahari. C# 3.0 in a Nutshell. 3rd Edition. O'Reilly Media, Inc. September(2007).

[Alexander 77]

Christopher Alexander. A Pattern Language. Oxford University Press, New York, (1977).

[Allen 08]

Rob Allen, Nick Lo, Steven Brown. Zend Framework in Action. Manning Publications. December (2008).

[Alur 03]

Deepak Alur, John Crupi, Dan Malks. Core J2EE Patterns: Best practices and design strategies. Sun Microsystems, Prentice Hall (2003).

[ASP-Beans 09]

http://aspbeans.sourceforge.net

[ASPNET 09]

http://asp.net

[Atkinson 03]

Leon Atkinson. Core PHP Programming, Third Edition. Prentice Hall. August (2003).

[Beck 87]

Kent Beck, Apple Computer, Inc., Ward Cunningham, Tektronix, Inc., Using Pattern Languages for Object-Oriented Programs. September 17 (1987).

[Blackboard 09]

http://www.blackboard.com/

[Booch 05]

Grady Booch, James Rumbaugh, Ivar Jacobson. Unified Modeling Language User Guide. Second Edition. Addison-Wesley Professional. May (2005).

[Booch 07]

Grady Booch, Robart A. Maksimchuk, Michael W. Engle, Bobbi J. Young Ph.D, Jim Conallen, Kelli A. Houston. Object Oriented Analysis and Design with Applications. Third Edition. Addison-Wesley Professional. April (2007).

[CakePHP 09]

http://cakephp.org

[Citibank 09]

http://online.citibank.com

[Codeplex-Locator 09]

http://www.codeplex.com/CommonServiceLocator

[Coggeshall 04]

John Coggeshall. PHP 5 Unleashed. Sams. December (2004).

[Conallen 99]

Jim Conallen. Modeling Web Application Architectures with UML. Rational Software. Communications of the ACM, volume 42, number 10, 63-70. October (1999).

[Cooper 03]

James W. Cooper. C# Design Patterns, a tutorial. Addison Wesley. August (2003).

[Crawford 03]

William Crawford, Jonathan Kaplan. J2EE Design Patterns. 1st Edition. O'Reilly. California (2003)

[Darie 08]

Cristian Darie, Wyatt Barnett. Build your own ASP.NET 3.5 Website using C# & VB. 3rd Edition. Sitepoint Pty. Ltd. (2008).

[Doctrine 09]

http://www.doctrine-project.org

[Dropr 09]

https://www.dropr.org/

[Eclipse 09]

http://www.eclipse.org/

[Evans 02]

Kirk Allen Evans, Ashwin Kamanna, Joel Mueller. XML and ASP.NET. Sams. April (2008).

[Fisher 06]

Marina Fisher, Ray Lai, Sonu Sharma, Laurence Moroney. Java EE and .Net interoperability: integration strategies, patterns, and best practices. Pretince Hall. April(2006).

[Fowler 02]

Martin Fowler, David Rice, Matthew Foemmel, Edward Hieatt, Robert Mee, Randy Stafford. Patterns of Enterprise Application Architecture. Addison-Wesley Professional. November (2002).

[Gamma 94]

Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software. Addison Wesley. November 10 (1994).

[Gibbs 03]

Matthew Gibbs, Rob Howard. Microsoft ASP.NET Coding Strategies with the Microsoft ASP.NET Team. Microsoft Press. October (2003).

[Gilmore 08]

W. Jason Gilmore. Beginning PHP and MySQL. Third Edition. Apress. March (2008).

[GlassFish 09]

https://glassfish.dev.java.net/

[Gruber 93]

Thomas R. Gruber. A translation approach to portable ontologies. Knowledge Acquisition, 5(2): 199-220, 1993.

[Guest 09]

Simon Gest. Microsoft .NET and J2EE Interoperability Toolkit. Microsoft Press. April (2003).

[Gutmans 04]

Andi Gutmans, Stig Sæther Bakken, Derick Rethans. PHP5 Power Programming. Pretince Hall. October (2004).

[Hohpe 03]

Gregor Hohpe, Bobby Woolf. Enterprise Integration Patterns. Addison-Wesley Professional. October(2003).

[Java-EJB 09]

http://java.sun.com/products/ejb/

[Java-J2EE 09]

http://java.sun.com/javaee

[Liberty 08]

Jesse Liberty, Dan Maharry, Dan Hurwitz. Programming ASP.NET 3.5, 4th Edition. O'Reilly. October(2008).

[Mayo 01]

Joseph Mayo. C# Unleashed. Sams. November (2001).

[McCallum 04]

Ethan McCallum. Building a PHP Front Controller. O'Reilly PHP Devcenter. Oreillynet.com. August (2004).

[McLean 02]

Scott McLean, James Naftel, Kim Williams. Microsoft .NET Remoting. Microsoft Press. October (2002).

[Metsker 06]

Steven John Metsker, William C. Wake. Design Patterns in Java. Pearson Education (2006).

[Monday 03]

Paul B. Monday. Web Services Patterns: Java Edition. 1st Edition. Apress. April (2003).

[Monnox 05]

Alan Monnox. Rapid J2EE Development: an adaptative foundation for enterprise applications. Prentice Hall. March (2009).

[Mono-Project 09]

http://mono-project.com

[Moodle 09]

http://moodle.org/

[Msdn 09]

MSDN. Microsoft Developer Network. Microsoft Partners and Practices Developer Center. http://msdn.microsoft.com/en-us/library/ms998572.aspx . (2009)

[Msdn Pipes-Filters 09]

http://msdn.microsoft.com/en-us/library/ms978599.aspx

[Msdn-ADO 09]

http://msdn.microsoft.com/en-us/library/ms254973.aspx

[Msdn-COM+ 09]

http://msdn.microsoft.com/en-us/library/3x7357ez(VS.71).aspx

[Msdn-DataPager 09]

http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.datapager.aspx

[Msdn-Front 09]

http://msdn.microsoft.com/en-us/library/ms998532.aspx

[Msdn-Messaging 09]

http://msdn.microsoft.com/en-us/library/aa480027.aspx

[Msdn-MSMQ 09]

http://msdn.microsoft.com/en-us/library/ms978425.aspx

[Msdn-Threading 09]

http://msdn.microsoft.com/en-us/library/system.threading.thread.aspx

[Net 09]

http://www.microsoft.com/NET/

[NetBeans 09]

http://www.netbeans.org/

[NET-CV 09]

http://www.netcv.com

[NET-Remoting 09]

http://msdn.microsoft.com/en-us/library/ms973864.aspx

[Papazoglou 07]

Michael Papazoglou. Web Services: Principles and Technology. 1st Edition. Prentice Hall. September (2007).

[Pear-Pager 09]

http://pear.php.net/manual/en/package.html.pager.intro.php

[Perl 09]

http://www.perl.org/about.html

[PHP 09]

http://php.net

[PHP-Java 09]

http://il.php.net/manual/en/intro.java.php

[PHP-JMS 09]

http://onjava.com/pub/a/onjava/2004/10/27/php-jms.html

[PHPMQ 09]

http://sourceforge.net/projects/phpmq/

[PHP-Msg 09]

http://us3.php.net/manual/en/function.msg-get-queue.php

[PHP-PDO 09]

http://es.php.net/manual/en/pdo.transactions.php

[PHPRPC 09]

http://www.phprpc.org

[PHP-Socket 09]

http://us3.php.net/manual/en/intro.sockets.php

[PHP-TAL 09]

http://phptal.motion-twin.com

[PHP-Universe 09]

http://universe-phpext.sourceforge.net/

[Sakai 09]

http://sakaiproject.org

[Sales-Force 09]

http://www.salesforce.com

[SAM 07]

http://pecl.php.net/package/sam

[Santacruz-Valencia 08]

Santacruz-Valencia, L. P., Navarro, A., Delgado Kloos, C., & Aedo, I. (2008). ELO-Tool: Taking Action in the Challenge of Assembling Learning Objects. Educational Technology & Society, 11 (1), 102-117.

[Santacruz-Valencia 09]

Liliana Patricia Santacruz-Valencia, Antonio Navarro, Ignacio Aedo, Carlos Delgado Kloos. Comparison of knowledge during the assembly process of learning objects. Springer Science. DOI 10.1007/s10844-009-0088-5 (under publication process).

[Silva-Rocha 03]

Silva, N., & Rocha, J. Semantic web complex ontology mapping. In Proc. web intelligence 2003 (pp. 82-88). Los Alamitos: IEEE Computer Society.

[Smarty 09]

http://www.smarty.net

[Stanek 07]

William R. Staneck. Internet Information Services (IIS) 7.0. Microsoft Press. December (2007).

[Sweat 05]

Jason E. Sweat. Guide to PHP Design Patterns. First Edition. Marco Tabini & Associates, Inc. July (2005).

[Tomcat 09]

http://tomcat.apache.org/

[Vaswani 02]

Vikram Vaswani. XML and PHP. Sams. June (2002).

[WACT 09]

http://wact.sf.net

[Wenz 07]

Christian Wenz. ASP.NET AJAX. O'Reilly. (2007).

[Worley 01]

Scott Worley. Inside ASP.NET. Sams. November (2001).

[Wyke 00]

R. Allen Wyke, Michel J. Walker, Robert Cox. PHP Developer's Dictionary. Sams. December (2000).

[Yogesh 99]

Yogesh Deshpande, Steve Hansen, Webengineering. http://www.webengineering.org. (1999).

[Zandstra 07]

Matt Zandstra. PHP Objects, Patterns and Practice. Second Edition. Apress. December (2007).

[Zend-Web 09]

http://www.zend.com

[Zend-WSDL 09]

http://files.zend.com/help/Zend-Studio-Eclipse/generate_wsdl_files.htm

[ZF 09]

Zend Framework Documentation. Programmer Reference Guide. http://framework.zend.com/.

[ZF-DB 09]

http://framework.zend.com/manual/en/zend.db.html

[ZF-Mail 09]

http://framework.zend.com/manual/en/zend.mail.html

10.Glosario

A

ADOdb:

Librería de abstracción de base de datos para PHP.

D

DHTML:

Dynamic HTML, es una combinación de tecnologías para crear sitios web interactivos.

H

HTML:

HyperText Markup Language, lenguaje de etiquetas que se usa normalmente para pintar páginas web.

J

JavaScript:

Lenguaje de programación que se ejecuta en el cliente (navegador web).

JSON:

JavaScript Object Notation, una alternativa para transmitir datos.

M

Mono:

Proyecto apoyado por *Novell*, implementación *Open Source* del Framework .NET. Permite instalar aplicaciones en Linux, Solaris y Mac OS X.

Memcached:

Sistema de caché de memoria distribuida de propósito general. Es un proyecto *Open Source* creado por *Danga Interactive* y ahora es usado por *Youtube, Slashdot, Wikipedia, Facebook, Twitter, NYTimes,* y otros sitios con gran demanda.

P

PDO:

PHP Data Objects. Interfaz ligera para acceder bases de datos en PHP.

PEAR:

PHP Extension and Application Repository. Un repositorio de librerías para PHP.

R

RDBMS:

Relational database management system. Sistema para gestión de bases de datos relacionales.

S

SQL:

Structured Query Language. Lenguaje para gestionar datos en bases de datos relacionales.

X

XML:

Extensible Markup Language. Conjunto de reglas para codificar documentos electrónicamente.