Ingeniare. Revista chilena de ingeniería, vol. 14 Nº 3, 2006, pp. 265-275

INTEGRACIÓN DE TECNOLOGÍAS EN UNA PLATAFORMA J2EE DIRIGIDA POR MODELOS

TECHNOLOGIES INTEGRATION IN A MODEL DRIVEN J2EE PLATFORM

David Colque C.[1] **Ricardo Valdivia P.**[2]

Recibido 5 de diciembre de 2005, aceptado el 11 de septiembre de 2006
Received: December 5, 2005 Accepted: September 11, 2006

RESUMEN

Este documento presenta una propuesta de cómo integrar adecuadamente diversas tecnologías de persistencia, de negocios y web en una plataforma J2EE. Esto involucra una arquitectura de múltiples capas, que considera para cada capa el uso de soluciones prácticas efectivas en el desarrollo de software, también se considera el uso del paradigma de desarrollo dirigido por modelos (MDA) para facilitar su integración, implementado en el *Framework Open Source* AndroMDA, y el proceso de desarrollo propuesto por Larman. De este modo se pretende fortalecer el desarrollo orientado a objetos al incorporar una herramienta MDA que genera código a partir de los modelos de cada plataforma específica e incentivar el uso de modelos en el proceso de desarrollo del software, mejorando así la calidad del software, la portabilidad, la interoperabilidad y la reusabilidad, como también la independencia de las tecnologías usadas en cada capa.

Palabras clave: Arquitectura dirigida por modelos, plataforma J2EE, framework, patrones de diseño, AndroMDA.

ABSTRACT

This work presents a proposal for integrating properly diverse technologies of persistence of business and Web in a J2EE platform. This involves an architecture of diverse layers, which considers the use of effective practical solutions for each layer in the development of the software. Also are considered the use of the Model Driven Architecture Paradigm, for facilitating integration performed in the Open Source AndroMDA Framework, and the development process proposed by Larman. This process is aimed at strengthening the development oriented to objects, when incorporating a MDA tool, which generates codes from the models of every specific platform; encouraging the use of models in the process of software development, improving the quality, the portability, the interoperability, as well as reusing the software, and also the self sufficiency of the technologies used in every layer.

Keywords: Model driven architecture, platform J2EE, framework, design patterns, AndroMDA.

INTRODUCCIÓN

En la actualidad, la adopción de la plataforma J2EE es una realidad consolidada. Pero presenta complejidad al integrar diversas tecnologías que cambian frecuentemente, lo que se traduce en un problema de adaptación de los desarrolladores menos experimentados a esta plataforma. La cantidad de modelos de desarrollo, *frameworks*, y APIs presentes en la construcción de una aplicación añade mayor confusión. A esto se suma la existencia de estándares impulsados por el grupo de expertos de J2EE, y los estándares de facto utilizados por los desarrolladores; particularmente en la capa de la lógica de negocio e integración.

El proceso de construcción de software actualmente tiene que adecuarse a los continuos cambios en las tecnologías de implementación, lo que implica un mayor esfuerzo en el diseño de la aplicación a fin de satisfacer las exigencias de las nuevas tecnologías que la incorporan. Cada vez es más difícil de atender y satisfacer los requisitos de escalabilidad, seguridad y eficiencia de un software, más aún, si vemos como aplicaciones *Bussines to Bussines* (B2B) y *Client to Bussines* (C2B), son cada vez más comunes.

[1] Magíster Ingeniería de Software. Escuela Universitaria de Ingeniería Industrial, Informática y de Sistemas. Universidad de Tarapacá, Arica-Chile, davidcolque@yahoo.es
[2] Escuela Universitaria de Ingeniería Industrial, Informática y de Sistemas. Universidad de Tarapacá. Arica-Chile, rvaldivi@uta.cl

Como consecuencia de lo anterior y mediante la exploración previa de diferentes proyectos que incorporan estrategias de persistencias, se identificó que el *Framework* de mapeo objeto relacional (ORM) *Hibernate* es más cercano a una arquitectura dirigida por modelos conocida como MDA (*Model Driven Architecture*), a diferencia de los proyectos que utilizan el estándar *Java Data Object* (JDO) y *Enterprise Java Beans* (EJB). Según las estrategias de persistencia analizadas EJB, JDO y ORM, es posible establecer que estas, por sí solas, no son suficientes en el desarrollo de software, debido a la complejidad de tratar la correspondencia entre el modelo orientado a objetos y el modelo relacional, en dominios grandes y complejos.

La información existente referente a un proceso guiado por modelo no detalla la integración de diversas tecnologías en la capa de la lógica de negocio, integración y de los sistemas legados. Además, hay que destacar los esfuerzos de varias organizaciones por construir una herramienta de desarrollo que incorpore tecnologías de persistencia más livianas que los EJB. Sin embargo, pocas de ellas se ajustan al paradigma MDA y, por ende, son diferentes entre sí.

Una comparación entre diferentes procesos de desarrollo basados en MDA, considerando el propuesto en este artículo, se muestra en la tabla 1.

La tabla 1 expresa lo siguiente:

- Todos los procesos analizados son de tipo ágil, es decir, usan métodos de desarrollos ágiles, sencillos y de iteración corta, no como los procesos RUP.
- Sólo AndroMDA y ArcStyler son procesos universales, ya que no están restringidos a un proceso en particular.
- La etapa de requerimiento de un desarrollo con MDA no es considerada en el ciclo de desarrollo por ningún proceso. Por otro lado, nuestra propuesta de desarrollo no abarca las etapas de requerimiento y de análisis.
- El proceso subyacente a OptimalJ solamente considera la generación de PSMs, de manera automática. Donde los modelos de la aplicación son generados a partir del PIM, mediante patrones de transformación, correspondientes a los PSMs de la capa de negocio, presentación y de la base de datos.
- Los procesos basados en una arquitectura CARAT[3] solamente permiten varias implementaciones de tecnologías.
- La implementación y unión de los componentes de cada capa de la arquitectura J2EE es parte del proceso de desarrollo. Por el contrario, el proceso de OptimalJ no considera la implementación de uniones de los componentes, ya que la herramienta genera los diversos PSMs y las uniones entre ellos.

Tabla 1. Evaluación de los procesos de desarrollo.

Proceso de desarrollo	**C1**	**C2**	**C3**	**C4**	**C5**	**C6**
Según OptimalJ	Ágil	No	A, D, I	Sí	No	auto-mático
Según AndroMDA	Ágil	Sí	A, D, I	No	Sí	Manual**
Según ArcStyler	Ágil	Sí	A, D, I	No	Sí	Manual**
Nuestra propuesta con AndroMDA	Ágil	No	D, I	No	Sí*	Manual**

* Limitada tan sólo para una arquitectura de múltiples capas de J2EE.

** Los elementos necesarios para la interacción de las capas los genera la herramienta.

Glosario: R : Requerimiento
A : Análisis
D : Diseño
I : Implementación

Donde se consideraron los siguientes criterios de evaluación:

C1	Tipo de proceso
C2	Proceso universal
C3	Ciclo de vida
C4	Generación de PSM
C4	Permite varias implementaciones
C5	Integración de modelos

Así, se puede afirmar en los siguientes apartados que a continuación están expuestos y específicamente detallados.

ASPECTOS GENERALES

Nuestra propuesta considera el uso del *Framework Spring* en la capa de la lógica de negocio, el que permite a objetos de negocio y de acceso a datos ser reusables, independizando de esta forma los objetos de Java de los servicios específicos de J2EE y de entornos administrados de JDO. Estos objetos de negocio persistentes pueden ser

[3] La arquitectura CARAT se basa en cartuchos que contienen un conjunto de reglas de transformación y se instala como un *plugin*.

reutilizados tanto en entornos J2EE (web o EJB), y en aplicaciones *standalone*, de una manera sencilla. En otras palabras, los objetos persistentes pueden ser manipulados por la capa de presentación mediante la implementación de los métodos de los servicios de *Spring* que manipulan a modo *back-end* los objetos de negocio de la capa de integración, desacoplando así la capa de la lógica de negocio de la tecnología usada en la capa de integración.

Para la capa de presentación, diversos *frameworks* de aplicaciones Web están disponibles, entre ellos: *Struts*, *WebWork* y *JavaServer Faces* (JSF). De los cuales se ha considerado la variante de *Struts, bpmb4struts,* el cual implementa el Modelo 2 del MVC (*Model-View-Controller*).

La tendencia de los métodos de desarrollo apunta a considerar:

- Un enfoque industrial para la producción de software: que tiene como lema "capacidad de producir software de alta calidad a bajo costo" que incluye automatización, estándares, reutilización de componentes, patrones, *frameworks* y configuración de soluciones.
- MDA como un nuevo paradigma de desarrollo dirigido por modelos.

Un método agrupa: tecnología, proceso y organización, los cuales se clasifican en Métodos Pesados y Ágiles, tales como:

- Procesos pesados: El estándar de facto *Rational Unified Process*, RUP.
- De desarrollo ágil: UP, *Extreme Programming* XP, *Scrum*, DSDM entre otros.

Desde esta perspectiva MDA [1] puede ser visto como una nueva tecnología que permite transformar un modelo independiente de la plataforma (PIM) a modelos específicos para una plataforma (PSM), cuyo proceso de desarrollo se puede dividir en tres fases:

- Construcción de un modelo independiente de la plataforma, de alto nivel del sistema.
- Transformación del modelo anterior a uno o varios modelos específicos de plataforma.
- Generación de código a partir de cada PSM.

Nuestra propuesta de desarrollo de software basada en un enfoque industrial y dirigido por modelos para una arquitectura J2EE es una personalización del proceso de desarrollo propuesto por Larman [1] y el proceso basado en UML de García Molina [5], en el que nos centramos en las etapas de diseño y construcción como se muestra en la figura 1.

Este plan de trabajo, es útil para cualquier metodología de desarrollo RUP, XP entre otros; que incluye una organización, tecnologías y un proceso de desarrollo, obteniéndose ventajas cuantitativas respecto a un desarrollo manual [4].

Estas ideas son ejemplificadas mediante la implementación de un sistema de bitácora web conocido como *Blog*, que permite la publicación y discusión de temas clasificados por categorías.

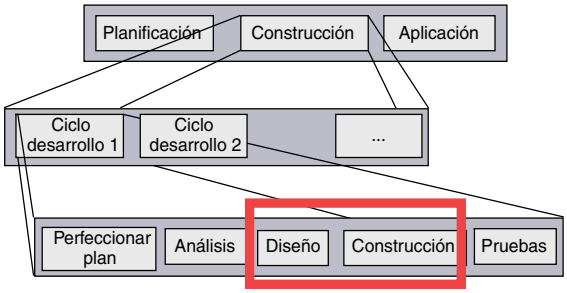

Figura 1. Proceso de desarrollo de interés.

PROPUESTA DE DESARROLLO

Esta propuesta de desarrollo se centra en las etapas de Diseño y Construcción del proceso de desarrollo propuesto por Larman (figura 1), considerando además un enfoque industrial guiado por modelos MDA. Por otra parte, considera el uso de *frameworks* de aplicación para cada capa de la arquitectura J2EE (figura 2), donde cada *framework* superior usa los servicios de las capas inferiores.

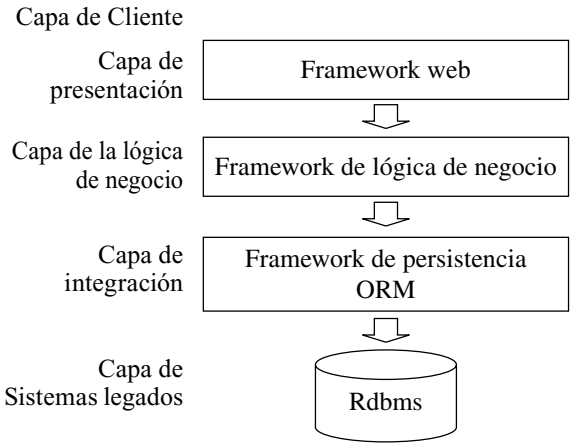

Figura 2. *Frameworks* utilizados.

El paradigma de desarrollo guiado por modelos se aplica de la siguiente manera como se aprecia en la figura 3.

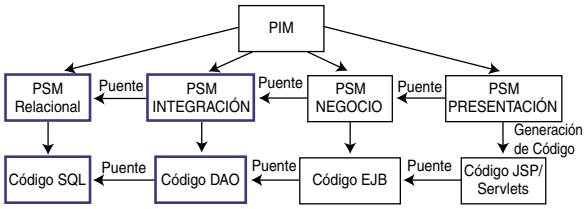

Figura 3. Modelo de transformación en MDA.

Para la construcción de cada PSM se utilizarán perfiles de tipo estereotipos y valor etiquetados que permitirán personalizar el modelo y por ende el código resultante para cada plataforma específica.

Esta propuesta se centra en la etapa de diseño y codificación respectivamente, requiriéndose como entrada los resultados de las tareas de la etapa de análisis resumidas en:

- El documento de requisitos.
- Diagrama de casos de uso.
- Modelo conceptual.
- Diagramas de secuencia, que identifican los eventos y las operaciones del sistema.
- Los contratos, que describen lo que hacen las operaciones del sistema.

Para luego iniciar la etapa de diseño de la aplicación considerando los siguientes aspectos:

1. Definición de las operaciones a implementar de los servicios del *framework* de la capa de la lógica de negocio que corresponde a las operaciones de sistema identificados en la etapa de análisis.

2. Definir los diagramas de diseño de clases.
   Involucra la construcción del modelo independiente de la plataforma con sus atributos, e identificación de las entidades persistentes mediante un estereotipo, es decir, aquellas entidades que persisten más allá de la ejecución de la aplicación.

3. Determinar los casos reales de uso.
   - Identificación de sólo un caso de uso de inicio o de arranque de la aplicación mediante un estereotipo.
   - Identificación de los casos de uso frontales de la aplicación, mediante el estereotipo de inicio *Front-end*, que podrán ser ejecutados después del caso de uso de inicio de la aplicación.

4. Determinar la secuencia de pantallas y reportes para la interfaz de usuario.
   - Construcción de los procesos del negocio, a través de un diagrama de actividad por paquete, mediante los estados de acción propias del servidor y las que se traducirán en una página JSP por el uso de estereotipos.
   - Involucra además agregar las operaciones de negocio al diagrama de actividad. Estas operaciones serán almacenadas en la clase control (*controller*) respectivo, propio del *framework* de la capa web según el patrón de asignación de responsabilidades denominado controlador [1].

5. Fijar los diagramas de interacción correspondientes a los de colaboración y de secuencia, para identificar las operaciones de cada objeto persistente, propio de la capa de integración.

6. Definir el esquema de la base de datos.
   - Esta tarea recae en el desarrollador encargado de montar la persistencia de la capa de integración, mediante el *framework* de persistencia objeto relacional (ORM).
   - Especificar las transformaciones de un modelo de objeto a uno relacional en el PSM de la capa de integración mediante el uso de perfiles, tipo estereotipos y el uso de valores etiquetados. Esto significa hacer persistente una entidad y sus asociaciones, requiriendo conocer:
     – Tipos de generador de identidad de una entidad.
     – Cardinalidad de las asociaciones.
     – Transformación de herencias.
     – Relaciones de agregación, composición.
     – Clases de asociación.

7. Perfeccionar la arquitectura
   - Involucra identificar operaciones que no tengan el soporte adecuado, por ejemplo validando que todas las operaciones de negocio puedan ser satisfechas por las operaciones del servicio, y estas estén soportadas por las operaciones de las entidades a nivel de la capa de integración.
   - Eliminar inconsistencias y redundancias.

8. Generar código y validar el modelo
   - Generación de los componentes de la aplicación asociados a los diferentes modelos.
   - Validación de los modelos, mediante la instalación del prototipo de la aplicación en el servidor de aplicaciones[4].

[4] En el caso particular de AndroMDA se genera un estereotipo de la aplicación el cual permite apreciar el proceso de negocio, aun cuando éste no es operable.

La tarea 1 es obtenida de la etapa previa de análisis. Las tareas 3 y 4 son consecutivas propias de la capa de presentación. Las tareas 2, 5 y 6 también son consecutivas pero involucran mayoritariamente a la capa de integración. Ambos grupos de tareas pueden ser desarrollados paralelamente, para finalmente realizar las tareas 7 y 8.

Una vez acabado el diseño y generado exitosamente el código base del prototipo de la aplicación, es posible pasar a la etapa de construcción del software. La estructura de los componentes de la aplicación generada determina regiones de componentes de código que pueden ser modificados.

En términos de funcionalidad, se sugiere construir el sistema de la siguiente forma:

1. Crear el esquema de la base de datos e implementar los métodos de las clases para la capa de integración.

   Los métodos de las clases se clasifican en dos tipos según su ámbito: de clasificación y de instancia a un objeto. Los de ámbito de clasificación están asociados a la implementación de un objeto DAO asociado a la entidad o clase persistente identificados en la etapa de diseño, y los de instancia a la implementación de la clase persistente.

   Un objeto DAO abstrae y encapsula el acceso a datos desacoplando las capas de lógica de negocio y de persistencia, permitiendo de manera sencilla migrar de una estrategia de persistencia a otra (ORM, EJB, JDO).

2. Implementar las operaciones del servicio de la capa de la lógica de negocio haciendo uso del objeto de servicio que usa los objetos DAO.

   Los métodos de la clase de servicio del *framework* de la lógica de negocio se valen de la implementación del patrón inyección de dependencia [3] que se basa en el principio del patrón de diseño Método de plantilla o *Hollywood principle* y su lema es: “no nos llame, nosotros lo llamaremos” [2], estos métodos en el *framework Spring* corresponden a la clase con la sigla Service<nombreServicio>Impl.java.

3. Implementar la capa de presentación.
   Esto involucra la implementación de los métodos de las clases que heredan e implementan a la clase de control (*Controller*) de la capa de presentación. Con el fin de poder enviar información a una página JSP o recibir los datos de un formulario y ejecutar un método de la lógica de negocio, esto comúnmente requiere usar un método de la clase *Controller* que retorne un objeto de servicio a fin de utilizar los métodos de la clase de servicio.

4. Si desea modificar el código específico a una página JSP, se debe editar la página JSP respectiva.

5. Finalmente, es posible instalar la aplicación en el servidor de aplicaciones.

En las próximas secciones de Diseño y Construcción se ejemplifica la propuesta de desarrollo considerando un sistema de bitácora web conocido como *Blog.*

El sistema *Blog* permite a miembros de una organización publicar diversos temas clasificados por categorías, que podrán ser vistos por los demás miembros. El autor o propietario de un *Blog* se le conoce como *Blogger* y es quien lo escribe.

ETAPA DE DISEÑO

En la construcción de los modelos de esta etapa se hará mención de los estereotipos y valores etiquetados específicos a cada cartucho (*cartridge*) del *framework Open Source* AndroMDA, en esta etapa usaremos la herramienta *case* Magic Draw 9.5 [7] por ser una de las más populares y recomendada por el grupo de desarrollo de AndroMDA. Los *cartridge* de AndroMDA corresponden a los *frameworks Open Source bmp4struts, Spring* e *Hibernate.* También, se hace mención de las diversas formas de generar los componentes de la aplicación y su instalación en el servidor de aplicación mediante el comando *maven*.

Las tareas que se tratarán en esta sección y que fueron detalladas en la propuesta, se muestran en la figura 4.

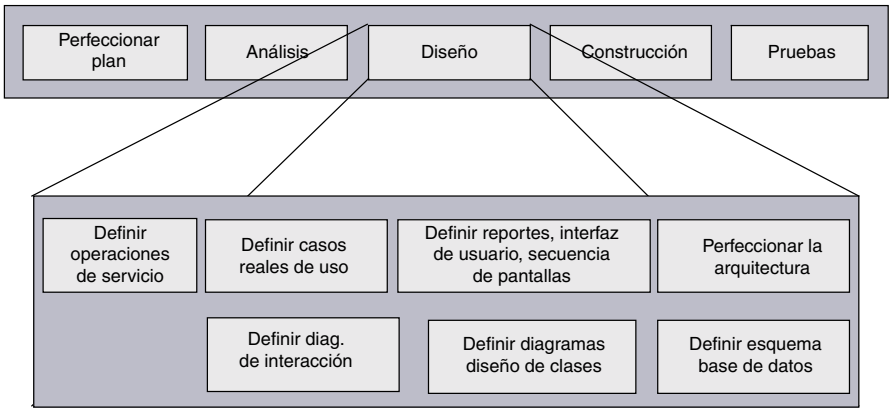

Figura 4. Tareas a realizarse en la etapa de diseño.

Definir operaciones de servicio

Corresponde a identificar las operaciones de la clase de servicio denominada ServicioBlog mediante el estereotipo <<*Service*>>, estas operaciones de sistema que a continuación se listan son implementadas posteriormente en la etapa de construcción mediante el *framework Spring*:

- Mensaje(mensajeBienvenida)
- verificacionLogin(nombreBlog, password)
- buscarCategorias(blog)
- buscarTemas(blog,categoría)
- buscarTema(tema, blog,categoría)
- buscarBlogs()
- registrarBlog(nombreBlog,nombrePropietario,email,password).
- registrarCategoria(nombreCategoría,blog)
- registrarTema(categoría,blog,tema,cuerpoTema, fechaPublicacion)

Definir diagramas de diseño de clases

Concerniente al modelo conceptual o de dominio independiente a una plataforma (PIM) para el sistema de ejemplo corresponde a la figura 5, este modelo PIM debe tener al menos el nombre de la clase, sus atributos y relaciones.

Determinar los casos reales de uso

Esta es una refinación de los casos esenciales de uso de la etapa de análisis previa. Debiéndose indicar qué caso de uso iniciará la aplicación mediante el estereotipo <<FrontEndApplication>>, y los casos de uso frontales de la aplicación que pueden ejecutarse una vez iniciada la aplicación mediante el estereotipo de inicio *Front-end* <<FrontEndUseCase>>, el diagrama de casos de uso reales corresponde a la figura 6.

Determinar la secuencia de pantallas y reportes para la interfaz de usuario

Está dada por la construcción del proceso de negocio, mediante un diagrama de actividad por paquete, identificando los estados de acción del servidor y del cliente que se traducirán en una página JSP mediante el uso del estereotipo <<FrontEndView>>. Las operaciones de negocio de este diagrama se agregarán a la clase de control (*controller*) que soporta a este diagrama de actividad. Por ejemplo, un diagrama de actividad para el proceso de autentificación corresponde a la figura 7.

Fijar los diagramas de interacción correspondiente a los de colaboración y de secuencia

Estos describen gráficamente cómo los objetos interactúan y son útiles para identificar las operaciones de las clases persistentes a implementar en la capa de integración. En la figura 8 se muestran las operaciones identificadas correspondientes a la capa de integración.

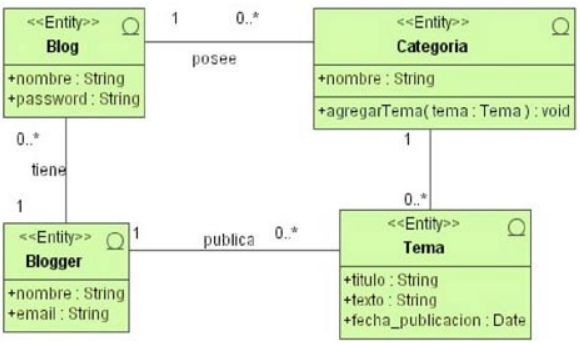

Figura 5. PIM de sistema *Blog*.

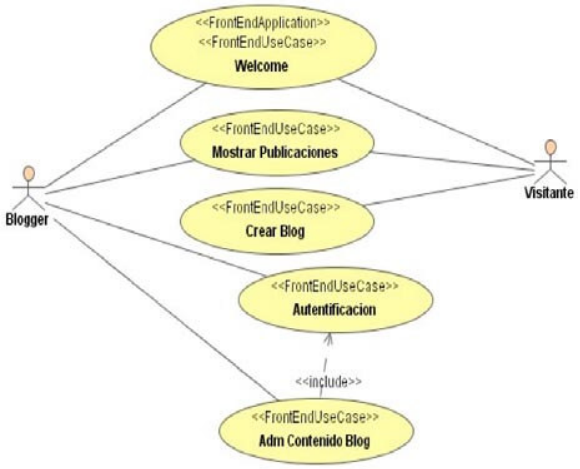

Figura 6. Caso real de uso.

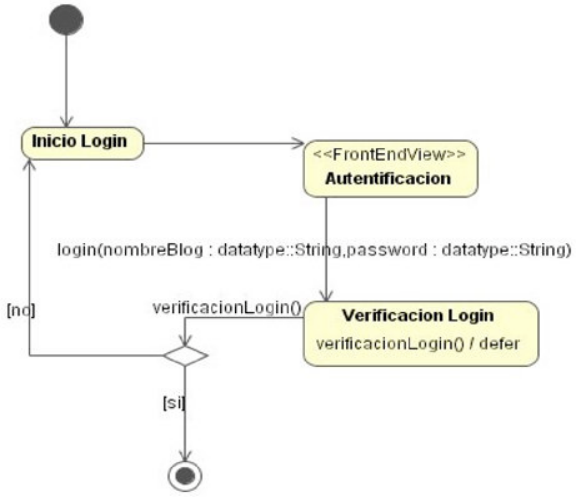

Figura 7. Diagrama de actividad de autentificación.

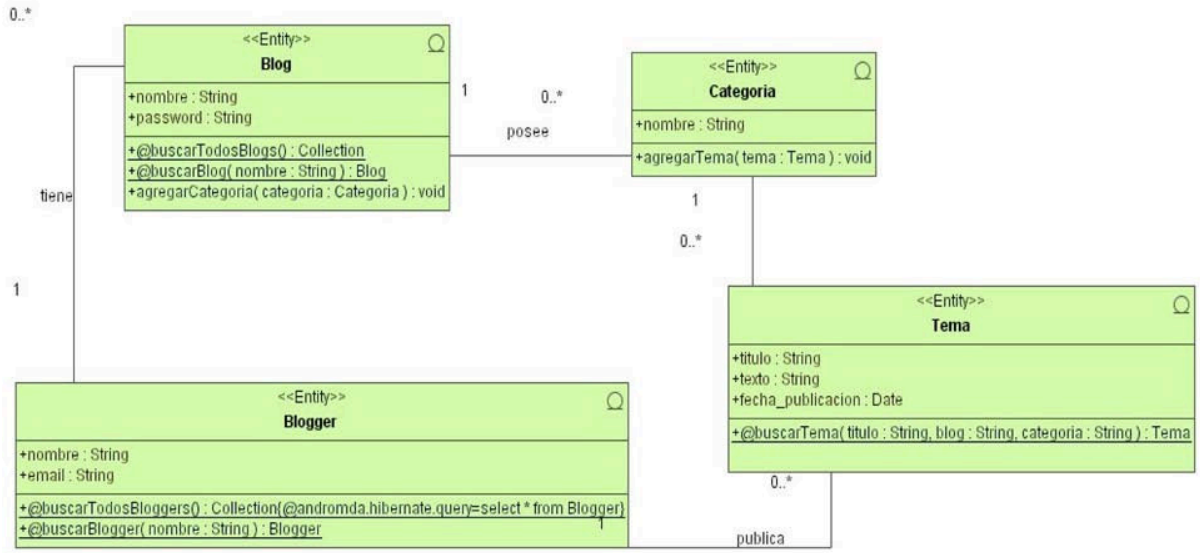

Figura 8. PSM de la capa de integración.

Definir el esquema de la base de datos

Es una tarea que recae en el desarrollador encargado de montar la capa de integración o de persistencia que involucra todas las clases persistentes mediante el estereotipo <<Entity>>, en este caso particular nos referimos al *framework Open Source* de mapeo Objeto Relacional (ORM) *Hibernate*. La especificación de las transformaciones al modelo PSM de la plataforma específica a la capa de integración, requiere el uso de perfiles, de tipo estereotipos y de valores etiquetados, para poder transformar un modelo de objeto a uno relacional, esto requiere conocer las estrategias de persistencia propias del *framework*, para una entidad y sus asociaciones.

Para la generación de la identidad del objeto se debe escoger entre la identidad manejada por el usuario que considera ciertos atributos del objeto como campo clave, o usar el tipo de identidad manejado por el *framework* de persistencia que en este caso corresponde a *Hibernate*; esta última forma de identificar un objeto fue la que se utilizó, en donde a cada entidad del modelo relacional el *framework* de persistencia agrega una columna adicional para identificar a cada objeto inequívocamente, el valor de esta columna está dado internamente por *Hibernate*. El diagrama conceptual resultante del esquema de la base de datos corresponde a la figura 9.

Perfeccionar la arquitectura

Requiere validar que todas las operaciones de negocio puedan ser satisfechas por las operaciones del servicio. De igual forma se debe validar que las operaciones del servicio estén soportadas por las operaciones de las entidades a nivel de la capa de integración.

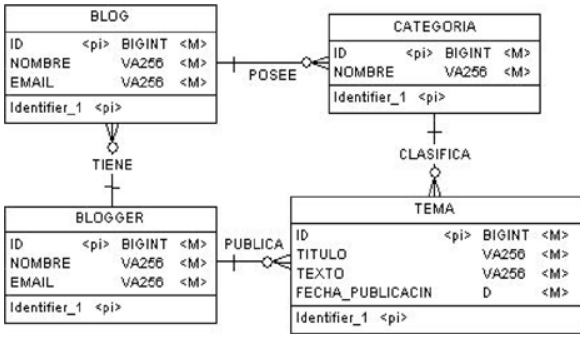

Figura 9. Modelo conceptual de la base de datos.

Generar código y validar el modelo

Una vez construidos todos los modelos, se puede generar el código de cada componente del software mediante el comando *maven install,* y luego instalar este prototipo de la aplicación en el servidor de aplicación mediante el comando *maven -o deploy.*

El modelo específico a la plataforma de la lógica de negocio construido corresponde a la figura 10, donde la clase ServicioBlog (<<Service>>) corresponde a un servicio y este a su vez depende de la implementación de las clases persistentes (<<Entity>>) de la capa de integración.

Por otro lado el modelo resultante al final de la capa de presentación involucra a varias clases *Controller* que soportan cada proceso de negocio, como se muestra en la figura 11, este incluye una clase de tipo Sesión denominada EstadoSesion mediante el estereotipo <<FrontEndSessionObject>>, para almacenar las variables de la sesión del dueño de un *Blog.*

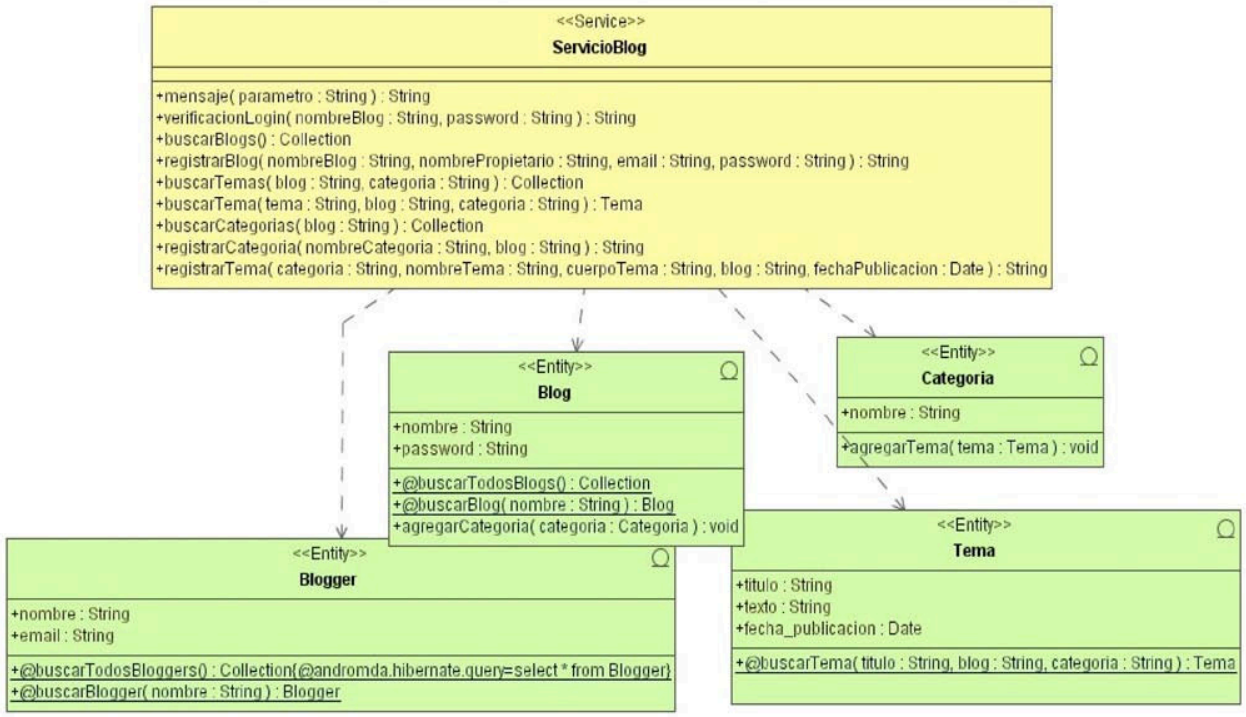

Figura 10. PSM de la capa de la lógica de negocio.

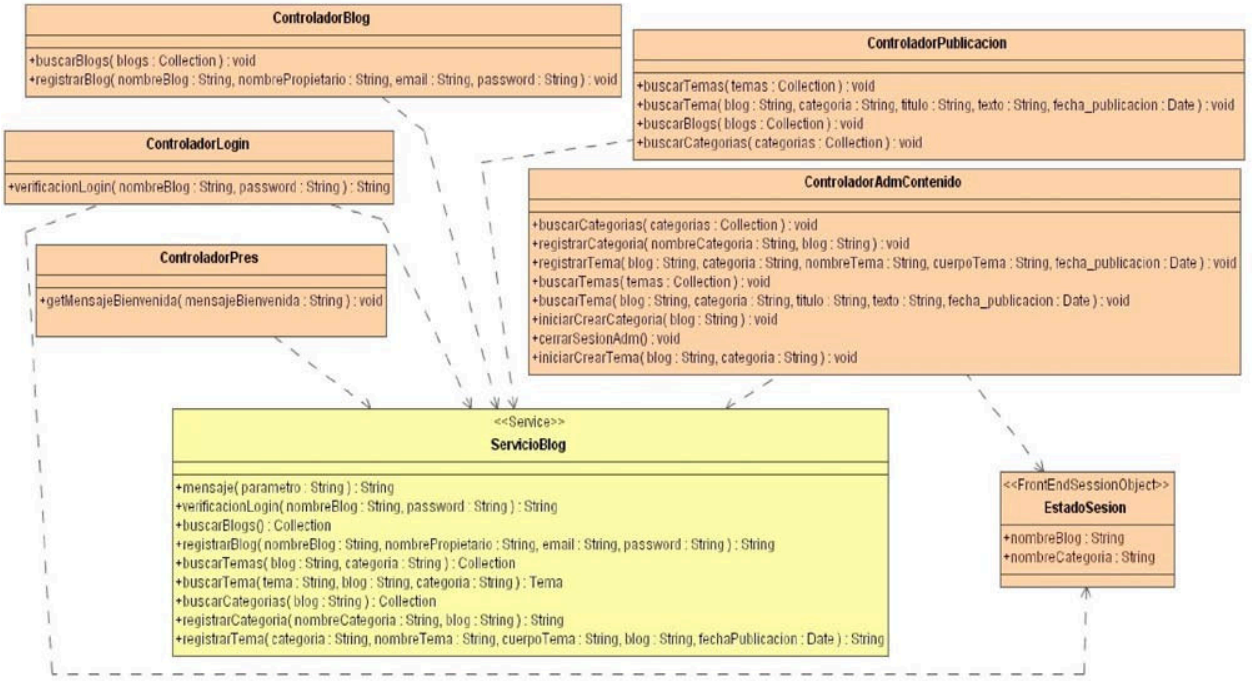

Figura 11. PSM de la capa de presentación.

ETAPA DE CONSTRUCCIÓN

En esta etapa fue necesario conocer cómo trabajan e interactúan los diversos *frameworks bmp4struts, Spring* e *Hibernate*, para saber qué clases modificar. A continuación se ejemplifica cómo nuestra propuesta de desarrollo para la etapa de Construcción fue aplicada en la construcción del Sistema *Blog.*

1. Capa de integración.

La construcción del esquema en la base de datos se realizó mediante la ejecución del *script* de creación del esquema mediante el comando *maven -o create-schema*.

Luego se realizó la implementación de los métodos de las clases persistentes, es decir, aquellas con estereotipo <<Entity>>, considerando el ámbito de las operaciones. Si el alcance de una operación es una instancia, se implementó en la clase del objeto en particular. Asimismo, si su ámbito es una clasificación de objetos se implementó en la clase del objeto DAO.

2. Capa de la lógica de negocio.

Aquí correspondió implementar todos los métodos de la lógica de negocio perteneciente a la implementación de la clase de servicio denominada ServicioBlog, utilizando los objetos de negocio de la capa de integración a modo *back-end*. Se utilizó un método implementado en la clase de servicio que retorna un objeto DAO, para a partir de este objeto obtener y manipular los objetos persistentes.

Una vez implementada esta capa se compiló el *core* o núcleo de la aplicación mediante la sentencia *maven core.*

3. Capa de presentación.

Aquí se implementó cada método de las clases de control (*Controller*) de cada paquete, a fin de poder desplegar información de los *Blogs* en las páginas JSP, y de recibir los campos de los formularios. También se utilizaron variables de sesión en aquellas clases de control dependientes de la clase EstadoSesion.

Para hacer efectivos los cambios se debió compilar tan sólo la capa web mediante el comando *maven web.*

4. Modificación de páginas JSP

En este paso, se modificaron algunas páginas JSP, añadiendo argumentos a los hipervínculos. Posteriormente para hacer efectivos los cambios se debió ejecutar nuevamente el comando *maven web.*

5. Instalación de la aplicación

Por último, se prosiguió con la construcción de los componentes *web* y *ear*, e instalación de la aplicación en el servidor de aplicación JBOSS 3.2.7, mediante el comando *maven -o deploy.*

El código y documentación del sistema *Blog* puede ser descargado para su evaluación desde el servidor del Departamento de Computación e Informática[5]. La página de inicio del *Blog* se muestra en la figura 12.

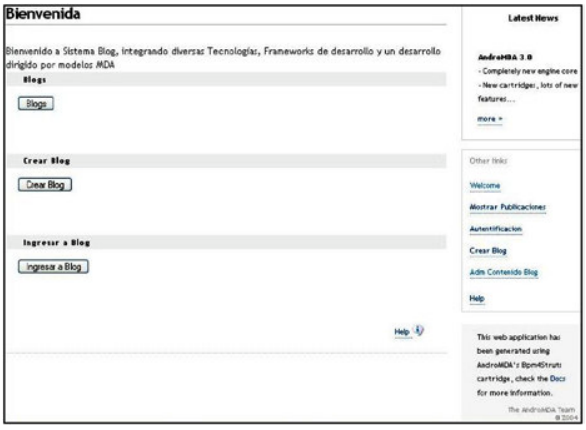

Figura 12. Página de inicio del sistema *Blog.*

CONCLUSIONES

La propuesta presentada es válida para sistemas que posean una arquitectura de múltiples capas como la plataforma J2EE, ejemplificada en el sistema *Blog*.

Esta propuesta incentiva el uso de modelos en el desarrollo de software, al incorporar en el proceso de desarrollo un enfoque industrial guiado por modelos que añade valor semántico, consistencia y facilidades de uso, obteniendo así ventajas frente a un proceso manual. En particular:

- El uso de perfiles de tipo estereotipo y valor etiquetado añade valor semántico a los modelos.
- La generación del código específico a los modelos de cada plataforma es generado por la herramienta.
- Se logra tener una correspondencia de los modelos de desarrollo.

Esta propuesta establece una guía de cómo usar el paradigma de desarrollo MDA en el proceso de desarrollo propuesto de Larman, a fin de obtener un producto de mejor calidad, facilitando la portabilidad, interoperabilidad y reusabilidad.

Se fortalece la independencia de tecnologías y el desarrollo orientado a objetos, al incorporar *frameworks* de aplicación para cada capa, que implementan diversos patrones de diseño. Así, el *framework Spring* de la capa de la lógica de negocio soporta diversas tecnologías web y de persistencia como ORM, EJB y JDO; de igual forma cada tecnología de persistencia soporta una vasta gama de gestores de base de datos relacionales propios de la capa de los sistemas legados.

Por otro lado, es posible que objetos normales de Java conocidos como POJO's sean persistentes y estén disponibles para la capa de presentación de manera sencilla y rápida a modo de servicios anteponiendo el estereotipo <<Entity>> en el modelo PIM. Esta perspectiva permite obviar las disputas entre las especificaciones *Enterprise Java Beans* (EJB) y *Java Data Object* (JDO).

Respecto a la tabla 1, de evaluación de los procesos de desarrollo, podemos señalar que nuestra propuesta ajusta el proceso de desarrollo universal de una herramienta MDA, en un proceso ágil, iterativo e incremental, basado en Larman [2], logrando así un proceso formal de desarrollo para las etapas de diseño y construcción. A diferencia del proceso utilizado en OptimalJ, nuestra propuesta requiere más código para la integración de los PSMs. Sin embargo, permite el uso de diferentes tecnologías de persistencia y web.

REFERENCIAS

[1] Object Management Group. "Model Driven Architecture Guide". 2003.

[2] C. Larman. "Applying UML and patterns". Prentice-Hall. Segunda edición. 2002.

[3] Rod Johnson, Juergen Hoeller et al. "Spring java/j2ee Application Framework". Documentación de referencia versión 1.2.1. 2005.

[4] Martin Fowler. "Inversion of Control Containers and the Dependency Injection pattern". Actualizado el 23 de Junio del 2004. Consultado en noviembre del 2005. http://www.martinfowler.com/articles/injection.html

[5] A. Kleppe, J. Warmer, and W. Bast. "Chapter 1: The MDA Development Process". MDA Explained. Addison-Wesley, pp. 1-12. 2003.

[5] http://www.decom.uta.cl

[6] García Molina Jesús, Ortín M. José, Moros Begoña, Nicolás Joaquín y Toval Ambrosio. "De los procesos de negocio a los casos de uso". JISBD 2000. Valladolid, España. Noviembre 2000. http://dis.um.es/~jmolina/jis2000modelado negocio.pdf

[7] AndroMDA tool. Actualizado 21 octubre 2005. Consultado en octubre del 2005. http://www.andromda.org

[8] "Magic Draw. User's Manuals, versión 9.5". No Magic, Inc. Abril 2005.