CONTENIDO

INTRODUCCIÓN

El desarrollo de software es actualmente uno de los pilares en el desarrollo tecnológico a nivel mundial. Como proceso, ha sufrido variaciones a través de la historia, aprovechándose de los progresos que la tecnología misma ofrece, haciendo posible la evolución de los mecanismos empleados para generar software con mayor calidad.

Uno de los mayores esfuerzos y a la vez anhelos que por años ha tenido la comunidad de desarrolladores de software está orientado a la construcción de software de manera automática, permitiendo al desarrollador concentrarse en la definición de las reglas de negocio y en el diseño arquitectónico de la solución, y desligarse un poco de la implementación propia de ellas.

Tratando de lograr este objetivo, surgen las llamadas Arquitecturas Dirigidas por Modelos (MDA, por sus siglas en inglés).

MDA en sí, es un estándar definido por el OMG (Object Management Group) bajo el cuál se plantea un esquema de reglas de transformación de varios niveles para obtener código fuente a partir de un modelo de diseño.

Para el objetivo de este trabajo de grado, nos apoyaremos en una herramienta llamada AndroMDA (http://www.andromda.org). Esta herramienta permite generar aplicaciones completas para las plataformas Java y .Net basados en modelos UML especificados en archivos XML.

AndroMDA está soportada por una serie de “cartridges” que se encargan de manera independiente de generar aplicaciones y ensamblar cada una de sus partes, dando como resultado una aplicación completa y funcional.

En este sentido, existen componentes encargados de generar de manera independiente las diferentes capas de aplicación, tales como persistencia, lógica de negocio, presentación, etc.

Con la versión actual de AndroMDA es posible generar una aplicación completa de administración de las tablas que componen una base de datos (CRUD (Create, Read, Update and Delete)). La limitación que se tiene actualmente es que esta generación está basada en un framework de presentación llamado Struts (http://struts.apache.org). Este framework ha sido por varios años un estándar de referencia dentro de la comunidad de desarrolladores de aplicaciones WEB bajo la plataforma J2EE. Sin embargo, las herramientas, patrones y frameworks han evolucionado, dejando a Struts como un framework obsoleto.

OBJETIVOS

A continuación se enumeran los objetivos del proyecto.

1. Objetivo General

Extender la funcionalidad de AndroMDA de generación de aplicaciones CRUD para permitir la construcción de una aplicación Web con esquema de presentación basado en el framework Spring MVC, integrándose a los componentes ya existentes, que permiten la generación de la capa de negocio y persistencia basados en Spring Framework e Hibernate.

2. Objetivos Específicos

2.1. Construir un generador de capa de presentación que utilice el modelo propuesto por Spring MVC.

2.2. Experimentar con las diferentes alternativas que se pueden emplear para extender la funcionalidad de AndroMDA, permitiendo hacer uso de esta herramienta de acuerdo a las necesidades que se tengan en un determinado caso.

2.3. Generar un documento en español donde se especifique clara y detalladamente el proceso de instalación e integración de AndroMDA

con los nuevos componentes que son desarrollados en este proyecto de grado.

2.4. Construir un proyecto ejemplo, que permita validar el correcto funcionamiento de los componentes desarrollados en este proyecto.

MARCO TEÓRICO

Las empresas desarrolladoras de software han tomado mayor interés e importancia el modelado en el desarrollo de cualquier tipo de software, debido a la facilidad que ofrece un buen diseño tanto a la hora de desarrollar como al hacer la integración y mantenimiento de sistemas de software.

En la actualidad una de las soluciones que existen para desarrollar productos de software, consiste en apoyarse en el concepto de MDA el cual plantea la posibilidad real de generar código fuente a partir de modelos.

1. Esquemas De Transformación De Modelos UML A Código Fuente

MDA propone basar el desarrollo de software en modelos especificados utilizando UML, para que, a partir de esos modelos, se realicen trasformaciones que generen código u otro modelo, con características de una tecnología particular (o con menor nivel de abstracción). MDA define un framework para procesar y relacionar modelos, el cual permite visualizar la estructura de desarrollo de aplicaciones de software que apoyan las diferentes áreas de una organización, centrando su mayor esfuerzo en la etapa de diseño, ya que a partir de ésta las demás etapas de desarrollo son automatizadas por los diferentes frameworks de desarrollo utilizados para lograr los objetivos funcionales de la aplicación.

En la *figura 1*, se ilustra el modelo general del concepto MDA con las áreas y tecnologías que abarca.

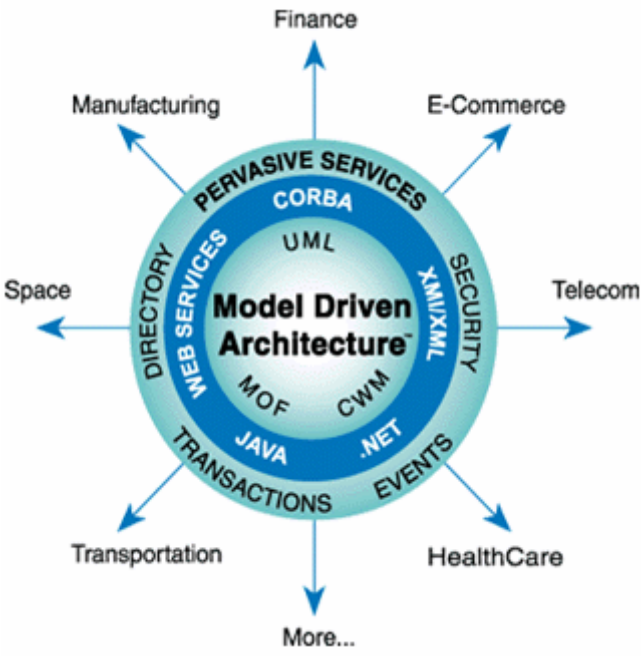

Figura 1: OMG´s Model Driven Architecture[1]

Usando la metodología MDA, la funcionalidad del sistema será definida en primer lugar como un modelo independiente de la plataforma (Platform-Independent Model o PIM), el cual puede traducirse en uno o más modelos específicos de la plataforma (Platform-Specific Models o PSM´s) para su implementación. La transformación entre el PIM y los PSM´s se realizan normalmente utilizando herramientas automatizadas, como herramientas de transformación de modelos. En este caso aquellas herramientas que cumplen con el estándar de OMG denominado QVT (Queries/Views/Transformations).

[1] http://www.omg.org/mda/

El estándar QVT pretende establecer un lenguaje para consulta de modelos (Q), un lenguaje para la definición y generación de vistas (V) y un lenguaje de transformación de modelos (T), que facilite el análisis de modelos desde las diferentes perspectivas de las personas que intervienen en el desarrollo de las aplicaciones

Uno de los principales objetivos de MDA es separar el diseño de la arquitectura y de las tecnologías de construcción, facilitando que el diseño y la arquitectura puedan ser alterados independientemente. El diseño alberga los requerimientos funcionales (casos de uso) mientras que la arquitectura proporciona la infraestructura a través de la cual se hacen efectivos requerimientos no funcionales como la escalabilidad, fiabilidad o rendimiento. MDA se asegura de que el modelo independiente de la plataforma (PIM), el cual representa un diseño conceptual que concreta los requerimientos funcionales, sobrevive a los cambios que se produzcan en las tecnologías de fabricación y en las arquitecturas software.

"MDA representa una nueva manera de organizar y administrar arquitecturas empresariales, basada en la utilización de herramientas de automatización de etapas en el ciclo de desarrollo y servicios. De esta forma, permite definir los modelos y facilitar trasformaciones paulatinas entre diferentes modelos. Algunos ejemplos de modelos son: el modelo de análisis, el de diseño y el de comportamiento, entre otros. Es decir que, a partir de uno de ellos, podemos generar otro de menor abstracción."[2]

[2] http://www.epidataconsulting.com/tikiwiki/tiki-read_article.php?articleId=55

1.1. Modelos MDA

"Los modelos juegan un rol trascendental en MDA. Como un framework para construir sistemas, MDA abstrae el sistema a construir en distintas capas de abstracción (layers). Tradicionalmente, el OOAD (Object Oriented Analysis and Design) contiene, entre otros, una vista de análisis, una vista de diseño detallado y código (representando la vista de negocios de un sistema), la vista de arquitectura y la vista de implementación. MDA agrega una capa de abstracción más, que representa el contexto de negocio del sistema."[3]

La capa de abstracción que se agrega se refiere al modelo CIM (Computational-Independent Model) el cual caracteriza el dominio del problema.

En la *Figura 2* se muestran las diferentes capas de abstracción (layers) que intervienen en el desarrollo de un producto de software adoptando la metodología MDA, las cuales se van haciendo mas concretas de izquierda a derecha.

[3] http://www.epidataconsulting.com/tikiwiki/tiki-read_article.php?articleId=55

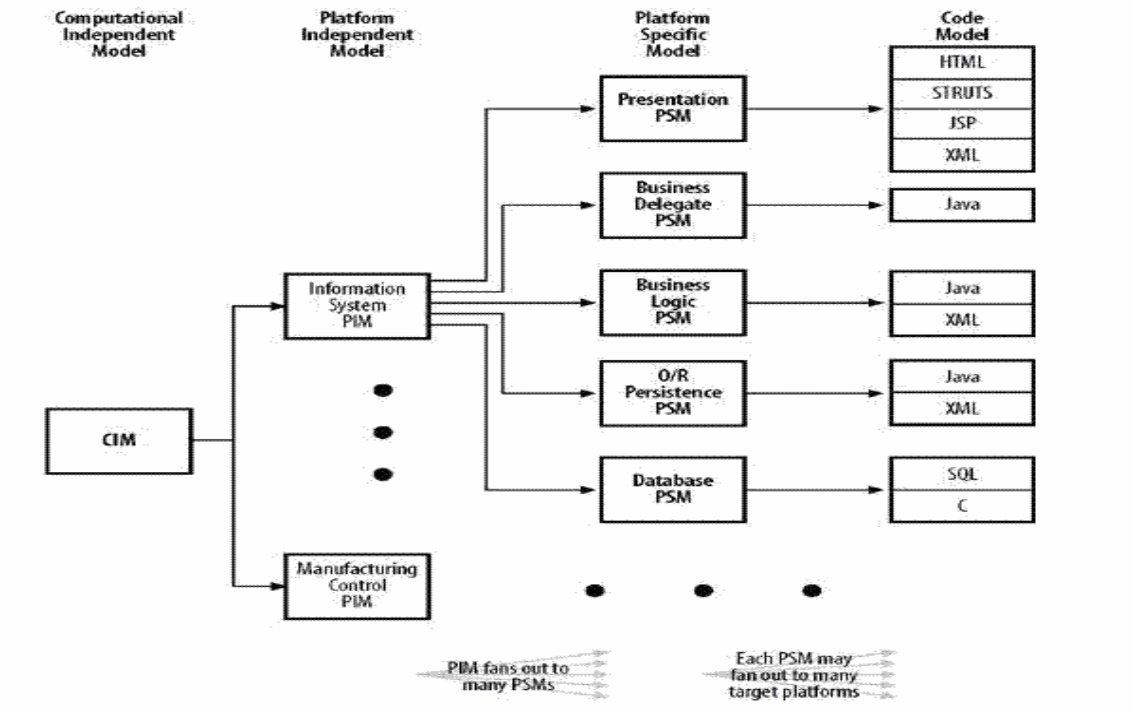

Figura 2: Un ejemplo de modelo MDA y sus relaciones"[4]

Como se observa en la *figura 2,* de acuerdo a la arquitectura propuesta por MDA se deben separar cada una de las capas o modelos que intervienen en el desarrollo de las aplicaciones de software (CIM, PIM, PSM y Code Model) con el fin de obtener mejores resultados. Cada una de estas capas o modelos se relacionada con la capa o modelo siguiente como fuente de entrada.

Por ejemplo, la capa o modelo CIM genera una idea inicial de los aspectos generales que el sistema de información debe proveer en su implementación, la capa o modelo PMI describe una solución de software que no contiene detalles de la plataforma en la cual será implementado, este a su vez sirve de

[4] http://www.epidataconsulting.com/tikiwiki/tiki-read_article.php?articleId=55

entrada para la generación de los modelos PSM los cuales definen los componentes arquitectónicos de la aplicación en la que será implementada la solución. Una vez están identificados estos componentes, se define el framework de desarrollo de la aplicación llamado Code Model.

"Los modelos concretos exceden en número a los modelos abstractos. A medida que avanzamos en las transformaciones, los modelos se vuelven más concretos, transformando al modelo abstracto en uno compatible con una tecnología o plataforma. Esto se produce debido a que MDA promueve la fuerte separación entre las responsabilidades de requerimientos del negocio y las responsabilidades tecnológicas. La ventaja de esta separación de responsabilidades es que ambos aspectos pueden evolucionar individualmente sin generar dependencias entre sí. De esta manera, la lógica de negocio responderá a las necesidades del negocio y no dependerá de vicisitudes técnicas."[5]

1.2. Decisiones De Diseño

Se ha visto cómo, de manera automática y paulatina, MDA promueve la transformación de modelos que representan lógicas de negocios complejas (CIM), hasta llegar al código ejecutable y desplegable (Code Model). Pero, ¿qué pasa con las decisiones de diseño, aquellas decisiones que tomamos cuando, por ejemplo, tenemos que desarrollar un sistema donde la mayoría de las transacciones sólo leen datos, pero diez de ellas hacen uso intensivo del

[5] http://www.epidataconsulting.com/tikiwiki/tiki-read_article.php?articleId=55

procesador? ¿Qué pasaría si todos los mapeos se hicieran de igual manera? Se es dado a pensar que esto degradaría la calidad final percibida de la aplicación. Para corregir este problema, MDA promueve el uso de Marks (marcas), las cuales indican aspectos específicos para tener en cuenta en cada transformación. Un PIM generado a partir de otro PIM se denomina PIM Marcado o Marked PIM. La utilización de las marcas establece que las decisiones respecto a aspectos tecnológicos queden fuera de los modelos principales como se muestra en la *figura 3*.

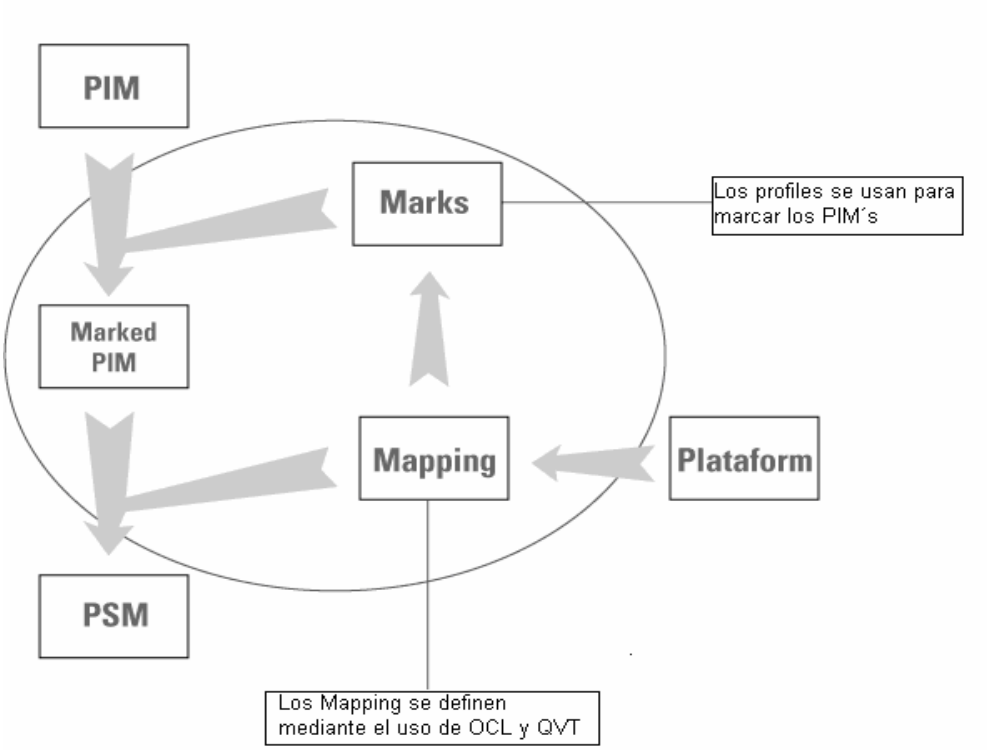

Figura 3: Utilización De Marcas Y Mapeos[6]

Ahora bien, para realizar el mapeo entre un PIM marcado y por ejemplo un PSM, es necesario detallar cómo se mapean esas marcas; para eso se definen

[6] www.omg.org/docs/omg/03-06-01.pdf

los Mappers (mapeadores). Puede verse la relación entre los mapeadores, las marcas y los PSM en la figura 3. Los mapeos se hacen utilizando un QVT.

1.3. Ventajas De MDA

La ventaja principal de MDA radica en la clara y estricta separación de responsabilidades. Por un lado, modelaremos los PIM´s, que representan los modelos de nuestro negocio, y por otro lado, los PSM´s con las preocupaciones tecnológicas. Esto permitirá que ambos modelos puedan evolucionar por separado. De esta manera, si quisiéramos, por ejemplo, modificar un aspecto técnico, bastará con modificar el PSM sin que estos tengan impacto en la lógica de negocios. Esta idea viene de un concepto que en Ingeniería de Software, se llama “Guías de Diseño”. Particularmente, una de esas guías dice que el modelado de la solución debe ser dirigido por el negocio. Esta guía se basa en la afirmación de que un cambio en el negocio seguramente produzca un cambio en el código, pero no lo inverso: los cambios en el código no deberían impactar en el negocio. MDA también permite lidiar con la complejidad del negocio, modelando a éste por separado y permitiendo su análisis y mejora; disminuir costos, si se cuenta con una herramienta MDA adecuada a las necesidades; mejorar la calidad de los modelos y procesos, mediante su análisis y la separación de responsabilidades.

1.4. Estándares Involucrados en MDA

Las tecnologías más importantes involucradas para poder llevar a la práctica los conceptos subyacentes en MDA, son:

1. Meta Object Facility (MOF): Es un modelo de integración extensible que impulsada la definición, manipulación e integración de datos y meta-datos en una plataforma independiente. MOF esta basado en estándares para la integración de herramientas, aplicaciones y datos.
2. Unified Modeling Language (UML): Es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema de software.
3. XML Model Interchange (XMI): XMI es un modelo manejado por el framework XML para la definición, intercambio, manipulación e integración de datos XML y objetos.
4. Common Warehouse Meta-Model (CWM): Especifica una interfaz estándar que puede utilizarse para facilitar el intercambio de Warehouse y metadatos de inteligencia de negocio entre las herramientas de Warehouse, plataformas Warehouse y repositorios Warehouse distribuidos en entornos heterogéneos.
5. Software Process Engineering Meta-Model (SPEM): Este metamodelo se utiliza para describir procesos concretos de desarrollo de software o una familia de programas relacionados con los procesos de desarrollo.
6. Query-View-Transformation (QVT).

2. Niveles De Definición De Modelos

A continuación se relacionan los modelos que son utilizados por MDA:

2.1. CIM (Computational-Independent Model)

El modelo CIM que se traduce como: “Modelo Independiente de la Computación” representa el nivel más alto del modelo de negocios, no muestra detalles de la estructura del sistema. A veces es llamado modelo de dominio o modelo de negocio. En él se modelan los requisitos que deberá satisfacer el sistema, describiendo la situación en la cual el sistema será usado. Es muy útil tanto para ayudar a comprender el problema como para ejercer de fuente de vocabulario compartido para el uso en otros modelos. El CIM juega un papel importante como puente entre los que son unos expertos en el dominio del problema y sus requisitos y aquellos que son expertos en el diseño y construcción de artefactos software.

2.2. PIM (Platform-Independent Model)

El modelo PIM, que se traduce como “Modelo Independiente de la Plataforma”, representa el modelo de procesos de negocio a ser implementado. Comúnmente se usa UML o un derivado de UML para describir el modelo PIM. Un modelo PIM muestra el grado de independencia de plataforma necesario para poder ser usado en diferentes plataformas tecnológicas de un tipo similar. Este modelo debe tener tal nivel de abstracción que no cambie, sea cual sea la plataforma elegida para su implementación. Con este modelo se representa la lógica del sistema y sus interacciones con el mundo exterior, sin entrar en

detalle de que tipo de tecnología implementará cada parte y cómo se adapta a una plataforma específica.

2.3. PSM (Platform-Specific Model)

El modelo PSM, que se traduce como “Modelo Específico de la Plataforma”, es una vista del sistema para una plataforma específica. Éste combina la especificación del sistema hecha en el PIM, con los detalles que especifican la plataforma o tecnología con que se implementará la solución.

En la *figura 4* se ilustra la transformación de los modelos de entrada llamados PIM u otro tipo de modelo cualquiera que al ser tratados bajo reglas de transformación puede generar uno o mas modelos PSM.

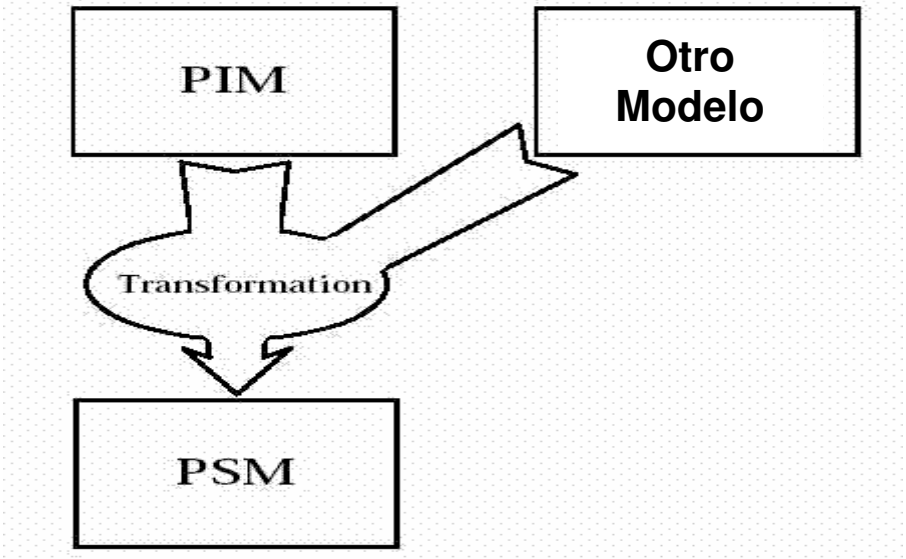

Figura 4: Los PIM´s Se Transforman En PSM´s[7]

[7] www.omg.org/docs/omg/03-06-01.pdf

2.4. Code Model

"El Modelo de Código representa el código desplegable (deployable), normalmente en un lenguaje de programación de alto nivel, como Java, C#, C++, VB, JSP, etc. Idealmente, el modelo de código está listo para compilar y no debería requerir la intervención humana; el despliegue de la aplicación podría ser automatizado. Según los puristas y algunos fanáticos de MDA, en un ambiente MDA maduro no deberíamos pensar en el código mas que como simples archivos, o como un mero objeto intermedio para generar el ejecutable final."[8]

MDA identifica tres tipos de modelos principales, sin incluir el modelo CIM, como se muestra en la *Figura 5*, donde se representan los artefactos centrales en el ciclo de vida de desarrollo en un ambiente de software dirigido por modelos.

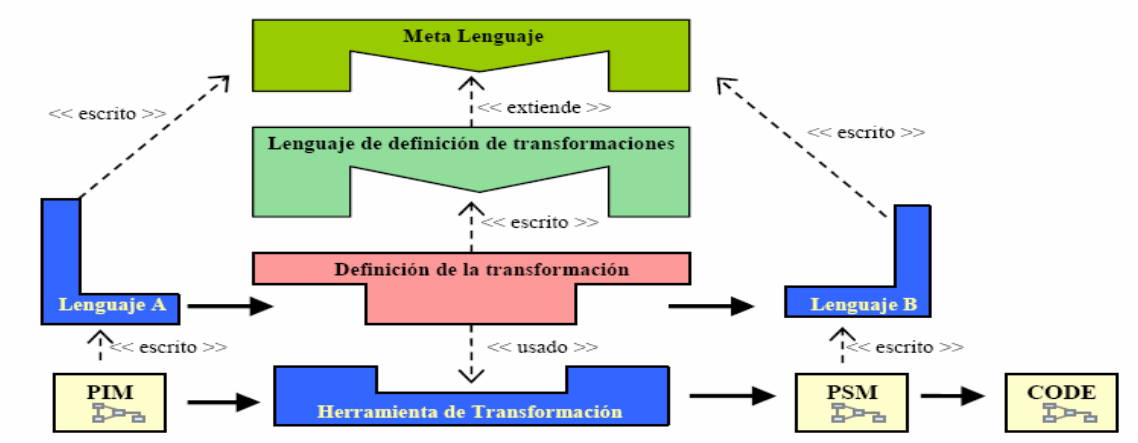

Figura 5: Integración De Los Componentes Del Framework MDA[9]

[8] http://www.epidataconsulting.com/tikiwiki/tiki-read_article.php?articleId=55
[9] Herramientas de soporte al proceso de desarrollo dirigido por modelos y su implementación con DSL Tools

Los modelos son considerados conductores primarios en todos los aspectos del desarrollo de software. Un PIM puede ser transformado en uno o más PSM´s, es decir, que para cada plataforma tecnológica específica se genera un PSM preciso. La transformación entre modelos constituye el motor de MDA y de esta manera los modelos pasan de ser entidades meramente contemplativas a ser entidades productivas.

Los modelos anteriormente mencionados pueden ser enfocados a los diferentes roles que desempeñan los participantes en un proyecto de software, como son:

- El analista de negocio, experto en el dominio del problema, modela una realidad por medio de CIM
- El arquitecto y el diseñador que definen una propuesta de solución (transformación del CIM en PIM) y progresivamente la concretan (transformación de PIM a PIM), hasta llegar a un diseño detallado.
- El desarrollador o el probador que tiene amplio conocimiento de la tecnología y decide la manera más adecuada como el diseño detallado se implementará en una plataforma particular (transformación del PIM a PSM)

TRANSFORMACIÓN DE MODELOS UML A CÓDIGO FUENTE EXTENDIENDO ANDROMDA

AndroMDA es un programa informático de tipo framework, utilizado para la generación extensible de código que se adhiere al paradigma de la arquitectura dirigida por modelos.

Al contrario de otros entornos de desarrollo MDA, AndroMDA ofrece una ventaja funcional que incluye un conjunto de "cartridge" enfocados a los paquetes de desarrollo actuales como son Axis, jBPM, Struts, JSF, Spring e Hibernate.

1. Historia Del Proyecto AndroMDA

El 3 de julio de 2002, Matthias Bohlen, fundador y creador del proyecto UML2EJB el precursor de AndroMDA, que estaba basado en la generación de código para programar con EJB 2.0, creó un generador de código con el fin de acompañar la publicación del libro "Enterprise JavaBeans GE-PACKT" que permitiera la automatización de procesos manuales en el desarrollo de aplicaciones. A medida que se fue desarrollando el proyecto, otros desarrolladores se fueron interesando en éste, como es el caso de Stefan Kuehnel y Ralf Wirdemann.

El comienzo de AndroMDA se dio el 3 Feb de 2003. En esta fecha el proyecto de AndroMDA fue aprobado en SourceForge.net (http://sourceforge.net/),

teniendo en su arquitectura algunos cambios significativos al proyecto inicial con el fin de que AndroMDA pudiese crecer en el futuro.

El 23 de julio de 2003 se liberó AndroMDA 2.0.2 final con la implementación de "cartridge" para EJB´s, Hibernate, Struts y plain Java.

Más desarrolladores se unieron al proyecto, se generó un plugin de AndroMDA para Poseidon, el cual fue vendido como extensión comercial.

Tres días después de la salida de la versión 2.0.2 final, AndroMDA comenzó a ganar ímpetu de una manera inimaginable. Una gran comunidad comenzó a formarse alrededor de la fuente abierta MDA. TheServerSide y otros agencias de noticias comenzaron a escribir artículos acerca de AndroMDA.

El 26 de julio de 2003, uno de los desarrolladores que se unieron al grupo, se dio cuenta de algunas inconsistencias que tenía el proyecto y comenzó a solucionarlas de manera que AndroMDA fuera un proyecto más seguro y confiable.

El 27 de Julio de 2003 se publica un documento con la visión de AndroMDA para los siguientes meses llamado "AndroMDA 3.0, la generación siguiente". Esto era importante para el proyecto porque permitió discutir la arquitectura de AndroMDA más fácilmente y con eficacia.

En el año 2004, se comenzaron a implementar casos de uso completos para J2EE a partir de modelos UML. Los usuarios de AndroMDA comenzaron a utilizar el generador para proyectos de desarrollo. Se liberaron tres lanzamientos 3.0M1 a 3.0M3 en mayo, agosto y diciembre del mismo año con el fin de permitir a los usuarios probar nuevas funcionalidades presentes en la nueva versión.

A medida que crecía el proyecto, los usuarios comenzaron a quejarse por la poca documentación, para lo cual se plantearon metas fundamentales en la utilidad del producto: Documentación y nuevas características de “cartridge”.
En la conferencia de JAX 2005, se anunció el lanzamiento de la versión 3.0 final la cual salió el 9 de mayo de 2005. AndroMDA 3.0 ahora era un producto más estable, usable y documentado. En este mismo año se implemento el “cartridge” para Spring que permitia la generación de servicios de fachada como Spring beans.

La versión 3.3 Final de AndroMDA (última versión) se encuentra disponible desde el 21 de abril de 2008. Al igual que las versiones anteriores, ha recibido una retroalimentación de la comunidad involucrada en el proyecto, la cual contiene principalmente correcciones de errores y pequeñas mejoras, tales como:

- Mejor soporte para las últimas versiones de Maven (Maven 2.0.8)
- Capacidad para leer archivos *.emx de RSA directamente, sin necesidad de convertirlos a *.uml2.
- Permite controlar la especificación de Meta-Modelos de IBM para los elementos gráficos dados en los diagramas UML.

2. Arquitectura general de las aplicaciones desarrolladas con MDA

Las aplicaciones empresariales modernas se construyen utilizando varios componentes relacionados entre sí, cada uno proporcionando una

funcionalidad específica. Componentes que realizan similares tipos de funciones generalmente se agrupan en capas. Estas capas son organizadas como una pila, en la que los componentes de una capa superior utilizar los servicios de componentes de una capa inferior. Un componente en una determinada capa podría generalmente utilizar la funcionalidad de otros componentes que se encuentran en su misma capa o en las capas inferiores. La *figura 6* muestra una estructura de capas utilizada para una aplicación empresarial.

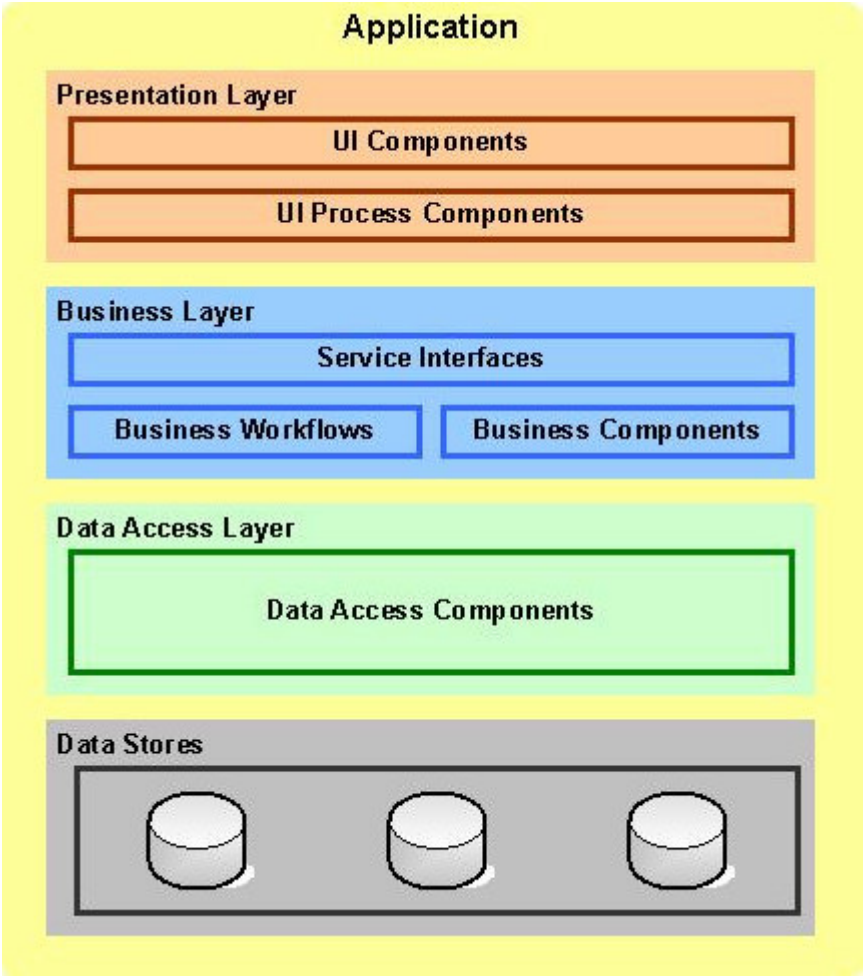

Figura 6: Diagrama general de las capas de una aplicación empresarial[10]

[10] http://galaxy.andromda.org/docs/getting-started/java/index.html

- **Presentation Layer (Capa de presentación):** La capa de presentación contiene los componentes necesarios para interactuar con el usuario de la aplicación. Ejemplos de esos componentes son las páginas Web, la interacción del usuario con los componentes del proceso, etc.
- **Business Layer (Capa de negocio):** La capa de negocio encapsula la funcionalidad del core del negocio en la aplicación. Las funciones del negocio pueden ser implementadas utilizando componentes complejos, mientras que las transacciones de mayor peso pueden ser implementadas utilizando flujos de trabajo (Workflow). Los componentes de negocio son por lo general front-ended, es decir, un servicio de interfaz que actúa como una fachada para ocultar la complejidad de la lógica de negocio. Esto se conoce comúnmente como SOA (Service-Oriented Architecture).
- **Data Access Layer (Capa de acceso a los datos):** La capa de acceso a los datos provee un API para permitir el acceso y la manipulación de los datos. Los componentes de esta capa abstraen la semántica de los datos que son solicitados, permitiendo así que la capa de negocio se enfoque en la lógica de negocio. Cada componente proporciona métodos para llevar ejecutar los CRUD´s, es decir, las operaciones de las entidades específicas del negocio.
- **Data Stores (Almacenes de datos):** Es el conjunto de aplicaciones empresariales que permiten el almacenamiento de los datos, como por ejemplo, las bases de datos, sistemas de archivos, entre otros.

3. Arquitectura de aplicaciones generadas con AndroMDA

Ahora que se conocen los conceptos básicos detrás de las aplicaciones empresariales modernas, vamos a analizar cómo AndroMDA implementa estos conceptos.

AndroMDA toma como entrada un modelo de negocio con especificaciones realizadas en UML y genera importantes porciones de las capas necesarias para construir una aplicación Java. AndroMDA tiene la capacidad de convertir automáticamente altos niveles de las especificaciones del negocio en código de calidad, lo cual significa un ahorro de tiempo en la generación de la aplicación final en Java. La *figura 7* presenta las diferentes capas de una aplicación desarrollada con AndroMDA para JAVA.

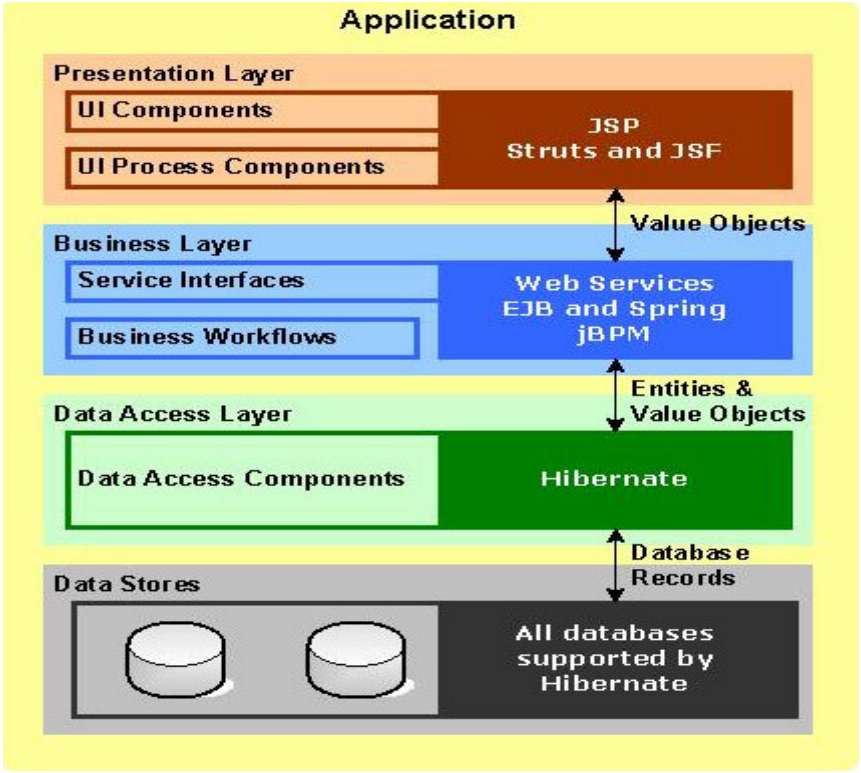

Figura 7: Capas de una aplicación JAVA desarrollada mediante AndroMDA[11]

[11] http://galaxy.andromda.org/docs/getting-started/java/index.html

- **Presentation Layer (Capa de presentación):** AndroMDA actualmente ofrece dos opciones de tecnología Web para construir la capa de presentación: Struts y JSF. Esta acepta los diagramas UML como entrada para poder generar los componentes Web que se ajustan a los framework de Struts o JSF.
- **Business Layer (Capa de negocio):** La capa de negocio generada por AndroMDA consiste principalmente en configurar los servicios usados por el framework de Spring. AndroMDA crea métodos vacíos en los cuales se puedan implementar clases a la lógica del negocio. Servicios opcionales pueden ser generados como front-ended mediante EJB´s. En este escenario los servicios deben ser desplegados dentro de un contenedor EJB como JBoss. Los servicios pueden ser expuestos como Web Services (Servicios Web), proporcionando una plataforma independiente para que los clientes puedan acceder a su funcionalidad. AndroMDA también pueden generar procesos de negocio y flujos de trabajo (Workflow) para el JBPM Workflow engine.
- **Data Access Layer (Capa de acceso a los datos):** AndroMDA emplea Hibernate para generar la capa de acceso a datos de la aplicación. Genera objetos de acceso a datos (DAO´s (Data Access Objects)) para las entidades definidas en el modelo UML. Estos objetos de acceso a datos utilizan el API de Hibernate para convertir los registros de la base de datos en objetos y viceversa.

- **Data Stores (Almacenes de datos):** AndroMDA genera aplicaciones con Hibernate para el acceso a los datos. Para el almacenamiento de datos se utilizan bases de datos soportadas por Hibernate.

4. Propagación de datos entre las diferentes capas

Además de los conceptos anteriormente mencionados, es importante comprender cómo se propagan los datos entre las distintas capas de una aplicación.

Las bases de datos relacionales guardan los datos como registros en tablas. La capa de acceso a datos obtiene estos registros de la base de datos y los transforma en objetos que representan entidades en el dominio del negocio. De ahí que estos objetos son llamados entidades del negocio.

La capa de acceso a datos transporta las entidades del negocio a la capa de negocio la cual realiza la lógica del negocio usando estas entidades.

Por último, para completar la propagación de datos, está la transferencia entre la capa de negocio y la capa de presentación.

Existen 2 formas de realizar esta transferencia de datos. Algunos expertos en el tema recomiendan que la capa de presentación deba tener acceso directo a las entidades del negocio. Otros recomiendan todo lo contrario, es decir, las entidades del negocio deben estar restringidas para ser accedidas por la capa de presentación y que la capa de negocio es la que debe suministrar este servicio de transferencia de estos objetos llamados “objetos de valor” a la capa de presentación. Se tienen pros y los contras de estos dos enfoques:

- El primer enfoque es más sencillo de implementar, ya que no se tienen que crear objetos de valor o implementar algún tipo de código que permita la transferencia de información entre las entidades y los objetos de valor. Esto se puede implementar en aplicaciones sencillas, en las cuales la capa de presentación y la capa de servicio se encuentran alojadas en la misma máquina física, lo cual no se recomienda para aplicaciones más complejas por la escalabilidad que se presenta.
    - Como la lógica del negocio no está totalmente definida en la capa de negocio, la libre manipulación de las entidades extendería la lógica de negocio a múltiples capas, lo cual representaría un incremento en la mantenibilidad de la aplicación. En caso de que existan múltiples front-ends para un servicio, la lógica de negocio podría ser duplicada en todos los front-ends. Además, no se garantizaría protección contra la corrupción de las entidades.
    - Cuando la capa de presentación se está ejecutando en un equipo diferente (como en el caso de un cliente Web), sería ineficiente mostrar toda la información que no se necesite en determinado momento, como por ejemplo, en el caso de órdenes de servicio mostrar todo el detalle de una orden, sin necesidad.
    - Pasar las entidades reales a los clientes puede poner en riesgo la seguridad de la aplicación.
- En el segundo enfoque, con la utilización de “objetos de valor” se le da una solución a los problemas anteriormente mencionados. Es necesario implementar más líneas de código, pero a cambio se obtiene una capa

de protección implementada en la capa de negocio, que es la que se comunica eficientemente con la capa de presentación.

5. Definición del problema.

Actualmente, AndroMDA está en capacidad de generar una aplicación JEE completa para la administración de tablas de base de datos. Basta con definir un modelo de clases UML, marcando las clases que se deseen incluir dentro de esta aplicación con un estereotipo denominado <<Manageable>>. Este estereotipo, le indica a AndroMDA y a los diferentes "cartridge" configurados dentro del proyecto, que generen las piezas de software necesarias para la construcción del sistema de administración de tablas.

En el uso normal de AndroMDA, existe un "plugin" de MAVEN, para la generación del esqueleto del proyecto. Este "plugin" guía al desarrollador en la definición de los componentes que desea utilizar para la generación de la aplicación. En uno de los puntos por los cuales el "plugin" va guiando al desarrollador, cuestiona a éste sobre cual framework de presentación desea utilizar, teniendo como posibles opciones únicamente a Struts y JSF. Sin embargo, hasta la versión 3.2 de AndroMDA, si se seleccionaba como opción JSF, no era posible generar la capa de presentación para las clases marcadas como <<Manageable>>. La adición de esta característica fue realizada solo hasta la versión 3.3 de AndroMDA que fue lanzada como versión estable solo hasta Abril del año 2008.

Entendiendo, que cada empresa, organización o persona, que desarrollan software, puede definir su propia arquitectura de desarrollo, es importante que existan diferentes alternativas a la hora de generar aplicaciones con AndroMDA. Es por esto que se abre un gran mundo de posibilidades de generación de aplicaciones usando AndroMDA, tomando en cuenta la evolución de las tecnologías orientadas a las WEB y que cada día aparecen nuevas posibilidades para ser usadas como capa de presentación, entendiendo eso sí, que cada una de ellas logre que el desarrollo de aplicaciones sea cada vez más simple y a la vez, genere una experiencia de usuario final, más amigable.

Como AndroMDA es una herramienta de software libre, existen actualmente varios grupos o personas, desarrollando diferentes “cartridge” compatibles con AndroMDA. Sin embargo, aun no existe una implementación que permita la generación de aplicaciones Web que se guíen por la arquitectura propuesta por el modulo de Spring llamado Spring MVC. Este modulo define una arquitectura bastante simple a partir del componente C de Controller, basado en la implementación de una interfaz que se encarga de gestionar las peticiones http que se le hacen a la aplicación. El componente M de Model, puede estar gestionado por Spring, o por cualquier otra tecnología Java. El componente V de View, puede ser implementado a través de simples páginas JSP o cualquier otro tipo de tecnología que pueda ser desplegada como resultado final a través de un browser de Internet, como son, salidas en XML, archivos PDF, archivos de Microsoft Excel, etc.

6. Propuesta a desarrollar.

El objetivo general de este proyecto plantea la posibilidad de extender la funcionalidad de AndroMDA para generar componentes personalizados en el contexto de Spring MVC.

Para lograr este objetivo existen varias alternativas descritas a continuación.

- Sobrescribir las plantillas de generación de componentes, existentes en algún “cartridge”: Esta alternativa es útil cuando se desea personalizar parte de los componentes generados por algún “cartridge”. Por ejemplo, el “cartridge” de Struts, genera varias páginas JSP, cada una de estas páginas puede ser personalizada a gusto o necesidad de un proyecto especifico.
- Modificar un “cartridge” existente: Esta alternativa es útil cuando se desea aprovechar las características existentes en un “cartridge”, pero se desea adicionar más características y componentes generados.
- Crear un nuevo “cartridge”: Esta alternativa es útil cuando alguna de las dos anteriores no es viable para cumplir con los objetivos que se deseen tener en un proyecto.

Evaluando las distintas alternativas, se encuentra que para cumplir con el objetivo de este proyecto, la mejor posibilidad es las segunda de ellas, modificando el “cartrigde” existente de spring, adicionando las características necesarias para la generación de la aplicación Web requerida para la administración de tablas.

7. AndroMDA Spring cartridge.

Este “cartridge” está basado en el framework Spring y debe ser usado en conjunto con el “cartridge” de Hibernate, para aprovecharse de las características de su integración para la capa de persistencia.

Spring es un contenedor JEE liviano. Esta creado como una alternativa para las aplicaciones que se construyan bajo el estándar JEE sin necesidad de contar con un contenedor EJB.

La idea principal es que spring es capaz de comunicarse directamente con servicios JEE como pueden ser JNDI, JTA, JMS, etc.

El núcleo del framework esta basado en el concepto de inyección de dependencias. La inyección de dependencias radica en resolver las dependencias de cada clase (atributos) generando los objetos cuando se arranca la aplicación y luego inyectarlos en los demás objetos que los necesiten a través de métodos set o bien a través del constructor, pero estos objetos se instancian una sola vez y son compartidos por toda la aplicación. De este modo, se elimina la necesidad de que el código (que generalmente se debe encargar de especificar unas reglas de negocio) deba encargarse de localizar los demás componentes con quien tenga algún tipo de relación.

La inyección de dependencias también es conocida como **Inversión de Control.** Este concepto se basa en la idea de que en la ejecución de alguna clase, esta no tenga necesidad de hacer algún tipo de petición por alguno de los componentes que este requiera, sino, que sea el contenedor de inversión

de control, quien se encargue de llamar el componente para asignarle las referencias a las dependencias que este tenga.

Soportado en esta teoría, spring provee de manera transparente integración con muchas tecnologías entre las cuales se destacan:

- Hibernate (y otros frameworks de persistencia)
- Enterprise Java Beans (Locales o Remotos)
- Servicios de Transaccionalidad
- Servicios de correo electrónico
- Web Services
- AOP

La ventaja principal de utilizar spring, es que permite realizar una implementación completa de un sistema basado en el concepto de POJO (Clases planas), es decir, que no tengan ninguna dependencia del framework.

Esto permite trasladar la ejecución de un componente, de un contexto a otro, sin generar problemas de compatibilidad por especificaciones de las marcas de los contenedores donde estos se alojan.

Al ejecutar AndroMDA, usando el “cartridge” de Spring, se produce como resultado la generación de las clases que conforman la lógica de negocio y la capa de persistencia. Adicionalmente, se crean todos los archivos XML necesarios para la definición de los Application Context requeridos para el correcto funcionamiento de la aplicación generada.

Por ejemplo, para la clase Color, descrita en la *figura 8*, se producen los archivos listados a continuación de la *figura 8*.

Figura 8: Diagrama de clase ejemplo

- Color.java: Este archivo contiene la definición de la clase Color.
- ColorDao.java: Este archivo contiene la definición de la interfaz para los métodos de acceso a la capa de persistencia.
- ColorDaoBase.java: Este archivo contiene la clase que implementa la interfaz definida en el archivo Colorado.java
- ColorDaoImpl.java: Este archivo contiene una clase que extiende de la clase ColorDaoBase. Esta clase solo se genera una vez, por lo cual, puede ser manipulada posteriormente por si se desea cambiar el comportamiento de sus métodos.
- ColorImpl.java: Este archivo contiene una clase que extiende de la clase definida en el archivo Color.java. Esta clase solo se genera una vez, por lo cual, puede ser manipulada posteriormente por si se desea cambiar el comportamiento de sus métodos.
- Color.hbm.xml: Este archivo contiene la configuración XML requerida para que la clase Color pueda ser manipulada por Hibernate.

- ColorManageableDao.java: Este archivo contiene la definición de la interfaz requerida para los métodos que permiten la manipulación de la clase Color en un esquema CRUD.
- ColorManageableDaoBase.java: Este archivo contiene la clase que implementa la interfaz definida en el archivo ColorManageableDao.java.
- ColorManageableServiceBase.java: Este archivo contiene la clase que implementa la interfaz definida en el archivo ColorManageableService.java
- ColorManageableService.java: Este archivo contiene la definición de la interfaz requerida para los métodos de la capa de negocio para manipular objetos de tipo Color.
- ColorValueObject.java: Este archivo contiene la definición del objeto que será transportado entre las diferentes capas de la aplicación.

Adicionalmente, para la clase Color, se generan diferentes entradas en los archivos de configuración del Application Context de spring. Estos archivos son:

- applicationContext.xml
- applicationContext-dataSource.xml
- applicationContext-import-remoteServices.xml
- applicationContext-localDataSource.xml
- applicationContext-manageable.xml

8. Hibernate

Hibernate es un framework de persistencia de objetos, de código abierto, que encaja dentro del concepto de ORM (Object Relational Mapping).
Es usado, para realizar un mapeo entre entidades de base de datos y objetos Java, encargándose este de convertir un objeto en un registro de base de datos y viceversa.

La idea general de usar una herramienta de persistencia, se reduce a lo complicado que resulta manejar el API JDBC para manipular los objetos Java con el fin de almacenarlos en una base de datos. Sin embargo, cualquier herramienta de persistencia que se use, sigue haciendo uso de JDBC porque a fin de cuentas, es el API establecido por Sun Microsystems para manipular bases de datos desde aplicaciones Java. Lo que en realidad hacen estos frameworks es crear capas de abstracción que evitan el uso directo de los componentes definidos por el API JDBC.

9. Desarrollo de la propuesta.

Un “cartridge” de AndroMDA, está compuesto por 3 componentes principales.

Estos componentes son los siguientes:

- Definición del cartridge: Este componente, esta implementado a través de un archivo XML llamado cartridge.xml. Este archivo contiene la definición de

todas las plantillas utilizadas para la generación de los componentes de una aplicación.

- Espacio de nombres: Este componente esta implementado a través de un archivo XML llamado namespace.xml. Este archivo contiene la definición de todas las variables utilizadas en las plantillas de generación de componentes utilizadas por el “cartridge”. Cada una de estas variables, pueden tener un valor por defecto que puede ser sobrescrito en cada proyecto, según las necesidades que se tengan. Otras, no tienen valor por defecto, así que por obligación deben ser especificadas en los archivos de configuración requeridos en un proyecto gestionado por AndroMDA.
- Metamodelo PSM del “cartridge”: Cada “cartridge” debe ser representado por un modelo UML que defina su comportamiento. Este modelo es usado para la generación del código fuente que compone el “cartridge” y los componentes que da como resultado son utilizados en conjunto con las plantillas de generación, para el procesamiento de un modelo UML de una aplicación especifica. Este es uno de los puntos fundamentales a destacar en el desarrollo de un “cartridge”, ya que el mismo, es generado a su vez por AndroMDA.

Teniendo en cuenta estos 3 componentes, la siguiente tarea es detectar de qué manera se deben modificar para hacer posible la extensión del “cartridge”.

Para el caso especifico de este proyecto, los pasos que se siguieron para lograr la construcción de un “cartridge” que permita la creación de aplicaciones JEE usando como capa de presentación Spring MVC, son los siguientes:

1. Modificar el metamodelo PSM, para agregar las características requeridas para la construcción de la capa de presentación, dando como producto de esta modificación, el siguiente modelo (*figura 9*):

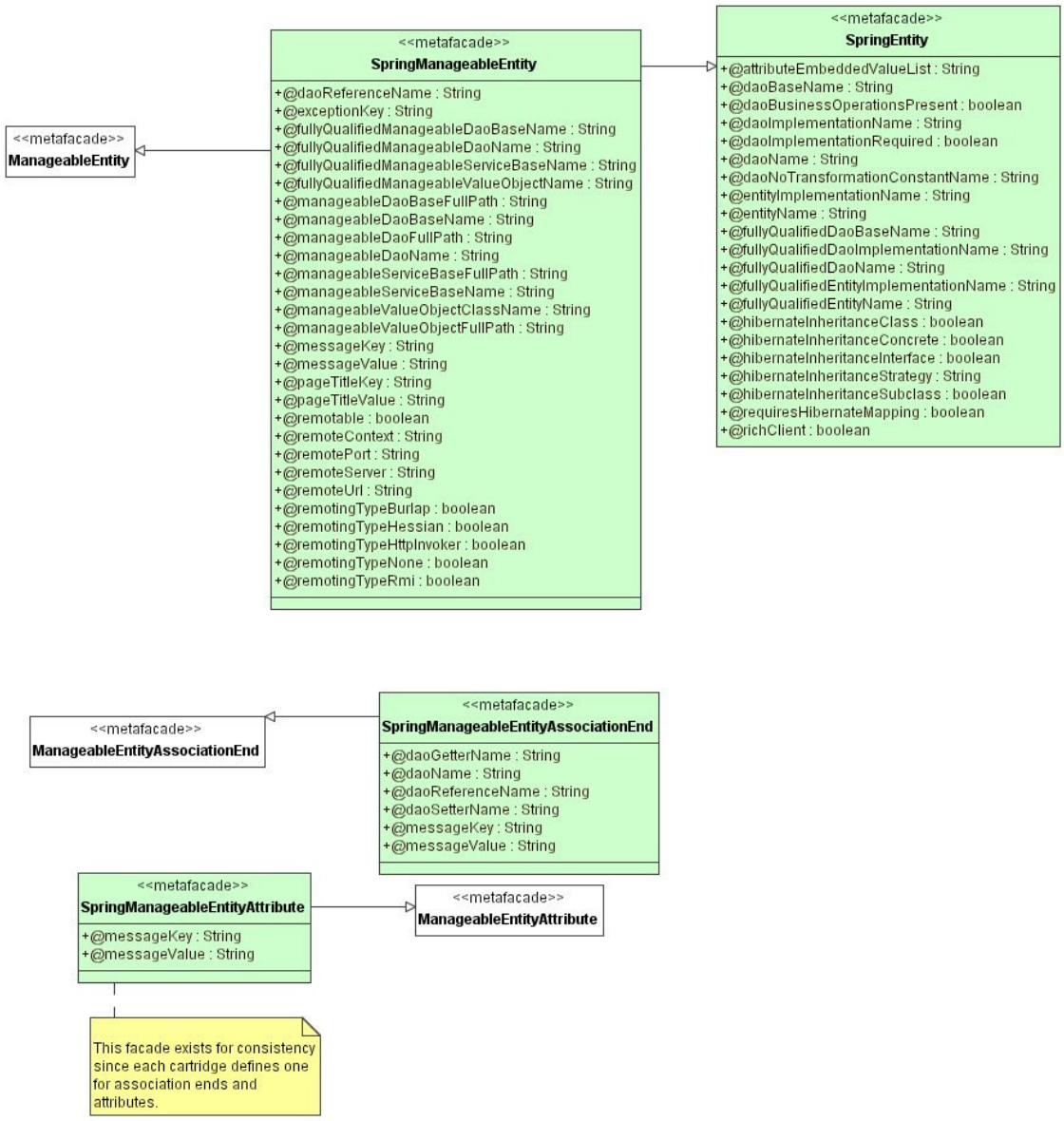

Figura 9: Meta-Modelo PSM de Spring para clases **<<Manageable>>**

2. Agregar las variables requeridas en el espacio de nombres del “cartridge”. Las variables adicionadas son las siguientes:

configuration:

La dirección donde se almacenan los archivos de configuración de la aplicación Web.

Valor por defecto: No tiene

webApplicationContext:

Nombre del archivo generado que contiene la definición del Application Context de spring para la capa de presentación.

Valor por defecto: webApplicationContext.xml

pages:

Dirección física del directorio raíz de la aplicación Web generada.

Valor por defecto: No tiene.

internalPages:

Dirección física del directorio donde se generaran los archivos JSP requeridos por la aplicación.

Valor por defecto: No tiene

internalDir:

Dirección relativa a la raíz de la aplicación Web, en donde serán almacenadas las páginas JSP generadas.

Valor por defecto: /WEB-INF/jsp

viewUtils:

Dirección física donde se almacenaran todas las clases java generadas para la aplicación.

Valor por defecto: No tiene.

messages:

Dirección física donde se almacenará el archivo de propiedades que contiene los mensajes utilizados por la aplicación.

Valor por defecto: No tiene.

dateFormat:

Formato de fechas utilizado por la aplicación para el manejo de campos que sean de tipo java.util.Date. Este formato, debe ser compatible con los formatos establecidos en la clase java.text.SimpleDateFormat.

Valor por defecto: yyyy/MM/dd

3. Escribir las plantillas necesarias para la generación de los distintos componentes requeridos para la aplicación Web.

   En el caso del "cartridge" de spring, se utiliza Velocity como motor de generación de componentes. Siguiendo con esta misma línea, se crearon las siguientes plantillas para los componentes específicos requeridos por la aplicación Web.

   web.xml.vsl

   Contiene la definición del archivo web.xml requerido en una aplicación Web Java. En esta plantilla se registra el servlet de spring así como los diferentes application contexts generados en la aplicación.

   index.jsp.vsl

Contiene la definición de la página index.jsp, utilizada como punto de entrada a la aplicación.

webApplicationContext.xml.vsl

Contiene la definición del archivo XML que compone el application Context necesario para el funcionamiento de Spring MVC.

portal.jsp.vs

Contiene la definición de la página JSP utilizada como plantilla de la aplicación Web. La página JSP generada define la forma en que será visualizada la aplicación Web.

ajaxPortal.jsp.vsl

Contiene la definición de la página JSP utilizada en las peticiones gestionadas a través de Ajax.

ajaxErrorResponse.jsp.vsl

Contiene la definición de la página JSP generada para mostrar los errores producidos durante el funcionamiento de la aplicación Web. Errores de validación o restricciones de integridad serán generados y visualizados a través de esta página.

tiles.xml.vsl

Contiene la definición del archivo tiles.xml. Este archivo permite la definición de las vistas utilizadas en la aplicación. Cada vista está asociada a una página JSP.

menu.jsp.vsl

Contiene la definición de la página que servirá como menú de la aplicación Web generada. Este menú contiene un link a una página por cada clase en el modelo UML que haya sido marcada con el estereotipo <<Manageable>>.

paginaBlanca.jsp.vsl

Contiene la definición de una página en blanco.

LoadViewController.java.vsl

Contiene la definición de una clase Java que permite mostrar cualquier vista definida en el archivo tiles.xml.

adminEntidad.jsp.vsl

Contiene la definición de una página JSP que contiene toda la funcionalidad Web requerida para la manipulación de una tabla de base de datos. Por cada clase en el modelo UML marcada con el estereotipo <<Manageable>> se crea una página JSP diferente.

listEntidad.jsp.vsl

Contiene la definición de una página JSP que servirá para mostrar todos los registros almacenados en base de datos para cada entidad. Por cada clase en el modelo UML marcada con el estereotipo <<Manageable>> se crea una página JSP diferente.

application-resources.properties.vsl

Contiene la definición del archivo de propiedades utilizado para almacenar los mensajes que serán mostrados en la aplicación Web.

ElementCommandController.java.vsl

Contiene la definición de una clase Java que implementa la jerarquía de clases propuesta por el modelo de Spring MVC, esta clase permite la interacción de peticiones http realizadas con Ajax.

CallbackCommandController.java.vsl

Contiene la definición de una clase Java que implementa la jerarquía de clases propuesta por el modelo de Spring MVC, esta clase permite la interacción de peticiones http realizadas con Ajax y la ejecución posterior de una función javascript definida en una página JSP.

CallbackObject.java.vsl

Contiene la definición de una clase Java requerida para el correcto funcionamiento de la clase generada en el archivo CallbackCommandController.java.vsl.

XMLEncoder.java.vsl

Contiene la definición de una clase Java que permite la codificación de Strings para que puedan ser transportados mediante XML.

AjaxResponseCreator.java.vsl

Contiene la definición de una clase Java que permite la generación de contenido XML requerido para el procesamiento de peticiones http realizadas con Ajax.

ClassPropertyEditor.java.vsl

Contiene la definición de una clase Java que permite la creación de una instancia de alguna clase a través de una propiedad de tipo String. Esta clase es una extensión de la clase java.beans.PropertyEditorSupport.

SaveController.java.vsl

Contiene la definición de una clase Java que se encargará de almacenar o actualizar un nuevo registro en base de datos. Por cada clase en el modelo UML marcada con el estereotipo <<Manageable>> se crea una clase diferente.

ListController.java.vsl

Contiene la definición de una clase Java que permite cargar un listado con todos los registros almacenados en base de datos para una entidad en

particular. Por cada clase en el modelo UML marcada con el estereotipo <<Manageable>> se crea una clase diferente.

LoadController.java.vsl

Contiene la definición de una clase Java que permite cargar una instancia en particular de una entidad desde la base de datos. Por cada clase en el modelo UML marcada con el estereotipo <<Manageable>> se crea una clase diferente.

DeleteController.java.vsl

Contiene la definición de una clase Java que permite tomar un conjunto de registros y eliminarlos de base de datos para una entidad en particular. Por cada clase en el modelo UML marcada con el estereotipo <<Manageable>> se crea una clase diferente.

ListSelectController.java.vsl

Contiene la definición de una clase Java que permite tomar todos los registros de una entidad en particular y desplegarlos en forma de lista de valores. Por cada clase en el modelo UML marcada con el estereotipo <<Manageable>> se crea una clase diferente.

4. Adicionar los recursos requeridos para la aplicación Web.

   En este caso, se deben adjuntar al “cartridge” todos aquellos archivos requeridos por la aplicación Web para su correcto funcionamiento. Estos archivos contienen la definición de funciones JavaScript, archivos con hojas de estilo (CSS), imágenes, etc.

5. Modificar el archivo descriptor del “cartridge”.

   Este archivo define cada una de las plantillas utilizadas por el “cartridge” para la generación de los diferentes componentes que formaran la aplicación generada.

   Existen dos tipos de plantillas, aquellas que producen como resultado un solo archivo y otras, donde se genera un archivo por cada clase marcada con el estereotipo <<Manageable>>.

   Un ejemplo de plantilla simple es la siguiente:

```
<template
        path="templates/spring/mvc/controller/ClassPropertyEditor.java.vsl"
        outputPattern="$generatedFile"
        outlet="viewUtils"
        overwrite="true"

        outputToSingleFile="true"/>
```

Con la definición de la propiedad “outputToSingleFile” con un valor “true”, se logra que solo se genere un archivo.

Un ejemplo de plantilla que produce como resultado varios archivos, es la siguiente:

```
<template
      path="templates/spring/mvc/controller/SaveController.java.vsl"
      outputPattern="$generatedFile"
      outlet="viewUtils"
      overwrite="true">
      <modelElements variable="entidad">
         <modelElement>
            <type
name="org.andromda.cartridges.spring.metafacades.SpringManageableEnt
ity"/>
         </modelElement>
      </modelElements>

   </template>
```

En este caso, no es requerida la propiedad “outputToSingleFile”, dado que su valor por defecto es “false”.

Por último, se deben definir los recursos adicionales mencionados anteriormente, que sirven de complemento para la aplicación Web generada, de la siguiente manera:

```
<resource
        path="resources/*.*"
        outputPattern="layout/{0}"
        outlet="pages"
        overwrite="true"/>

    <resource
        path="resources/calendar/*.*"
        outputPattern="layout/calendar/{0}"
        outlet="pages"

        overwrite="true"/>
```

MANUAL TÉCNICO

1. Prerrequisitos:

Para el correcto funcionamiento del software desarrollado en este proyecto, se requiere de la previa instalación de los siguientes elementos de software (Todo el software requerido para el funcionamiento de este proyecto está incluido dentro del CD que acompaña a este documento):

Java Development Kit

Este software puede ser descargado de la siguiente dirección:

http://java.sun.com/javase/downloads/?intcmp=1281 (Se recomienda JDK 6 Update 6)

El procedimiento para su instalación se encuentra en la siguiente dirección:

http://java.sun.com/javase/6/webnotes/install/index.html

Maven

Este software permite gestionar proyectos Java. Se utilizara para compilar sus clases, crear archivos JAR, crear archivos EAR, crear archivos WAR, etc. Adicionalmente permite realizar el despliegue de las aplicaciones generadas hacia un contenedor JEE o contenedor de servlets tales como Apache Tomcat o JBoos Aplication Server.

Este software puede ser descargado de la siguiente dirección:

http://maven.apache.org/download.html

Se recomienda trabajar con la versión 2.0.5.

Este software se descarga como un archivo comprimido (.zip).

Para instalarlo basta con descomprimir el contenido de dicho archivo en cualquier ubicación dentro del sistema archivos de la maquina.

2. Creación de variables de entorno

- JAVA_HOME: Se debe crear una variable de entorno con este nombre, cuyo valor debe ser la dirección física donde que instalo el JDK.
  Por ejemplo: C:\Archivos de programa\Java\jdk1.5.0_06
- M2_HOME: Esta variable debe contener como valor, la dirección física en donde se descomprimió el archivo que contiene el software maven.
  Por ejemplo: C:\javaUtils\maven-2.0.5
- M2: Esta variable contiene la dirección física de los archivos ejecutables del software maven.
  Por ejemplo: C:\javaUtils\maven-2.0.5\bin
- MAVEN_OPTS: Variable de configuración de opciones de Maven
  Valor sugerido: -Xms256m -Xmx512m
- PATH: Se debe modificar el valor de esta variable para incluir los comandos de ejecución de maven. Al momento de modificarla se debe incluir el siguiente valor: %PATH%;%M2%

Para verificar que todas las variables han sido creadas con éxito y que está listo el ambiente de ejecución del software, se debe abrir una consola DOS y ejecutar el siguiente comando:

- mvn --version

Como resultado debe aparecer el siguiente mensaje:

- C:\ >mvn --version

  Maven version: 2.0.5

3. Compilar el software.

El software se encuentra en el CD que acompaña este documento, en un archivo llamado ***andromda-src-3.2.zip***

Se debe descomprimir en alguna ubicación dentro del sistema de archivos.

Suponiendo que el archivo fue descomprimido en la ubicación:

C:\ andromda-src-3.2, se abre una consola DOS y se ejecutan los siguientes comandos:

- C:\>cd andromda-src-3.2
- C:\andromda-src-3.2>mvn install

Produciendo como resultado el siguiente mensaje:

[INFO] --

[INFO] BUILD SUCCESSFUL

[INFO] --

[INFO] Total time: 19 minutes 41 seconds

[INFO] Finished at: Mon May 26 21:00:59 GMT-05:00 2008

[INFO] Final Memory: 81M/254M

[INFO] --

Se debe tener en cuenta que la primera vez que se compila el proyecto, se toma un tiempo considerable en esta tarea, debido a que maven se encarga de descargar todas las dependencias requeridas desde ubicaciones en Internet.

Nota: Para ejecutar maven desde un computador que se encuentra en una red protegida por un firewall, se deben establecer los parámetros del Proxy de la siguiente manera:

- Abrir y editar el archivo de configuración de maven en la ubicación <M2_HOME>\conf\settings.xml

  En este archivo, buscar la sección de configuración de Proxies.

  Por defecto, esta configuración se encuentra de la siguiente manera:

```
<proxies>
  <!-- proxy
   | Specification for one proxy, to be used in connecting to the network.
   |
  <proxy>
    <id>optional</id>
    <active>true</active>
    <protocol>http</protocol>
    <username>proxyuser</username>
    <password>proxypass</password>
    <host>proxy.host.net</host>
    <port>80</port>
    <nonProxyHosts>local.net,some.host.com</nonProxyHosts>
```

</Proxy>

-->

</proxies>

- Adicionar la definición de un Proxy, de la siguiente manera:

<proxy>

<active>true</active>

<protocol>http</protocol>

<host>proxy.somewhere.com</host>

<port>8080</port>

<username>proxyuser</username>

<password>somepassword</password>

<nonProxyHosts>www.google.com|.somewhere.com</nonProxyHosts>*

</Proxy>

Los valores deben ser modificados dependiendo de los parámetros usados en la red donde se encuentra la maquina desde donde se esta ejecutando maven.

4. Ejecución del proyecto ejemplo:

4.1. Prerrequisitos:

4.1.1. Instalación del motor de base de datos:

En el CD que acompaña este trabajo se encuentra un archivo llamado mysql-5.0.51b-win32.zip, descomprimir y ejecutar el archivo setup.exe. Para mayor información sobre el procedimiento de

instalación de MySql, se puede tomar como referencia la documentación encontrada en la página http://www.mysql.com/

4.1.2. instalación del Servidor de aplicaciones.

En el CD que acompaña este trabajo se encuentra un archivo llamado jboss4.zip. Descomprimir este archivo en alguna ruta dentro del sistema de archivos de la maquina.

4.1.3. Definición de variable de entorno.

Se debe crear una variable de entorno con el nombre JBOSS_HOME, cuyo valor es la dirección física donde se descomprimió jboss. Por ejemplo:

C:\javaUtils\jboss4

4.1.4. Ejecución del proyecto ejemplo:

1. En el CD que acompaña este proyecto, se encuentra un archivo llamado proyectoEjemplo.zip. Descomprimir este archivo en alguna ruta del sistema de archivos.
2. Dentro del directorio <Ruta donde se descomprime>/proyectoEjemplo, ejecutar el comando mvn.

   C:\tesis\proyectoEjemplo>mvn

   Debe producir como salida el siguiente mensaje:

   [INFO] BUILD SUCCESSFUL

   [INFO] --

   [INFO] Total time: 1 minute 17 seconds

[INFO] Finished at: Mon May 26 21:27:08 GMT-05:00 2008

[INFO] Final Memory: 9M/254M

[INFO] --

3. Iniciar el servidor de aplicaciones.

   En una consola DOS ejecutar el siguiente comando:

   %JBOSS_HOME%\bin\run.bat

   Debe producir la siguiente salida:

   21:44:23,565 INFO [Server] JBoss (MX MicroKernel) [4.0.4.GA (build: CVSTag=JBoss_4_0_4_GA date=200605151000)] Started in 22s:692ms

4. Crear el esquema de base de datos.

   Crear en mysql un esquema llamado “ejemplo”.

   Crear un usuario llamado “ejemplo” con clave “ejemplo”.

   Adicionar todos los permisos al usuario “ejemplo” en el esquema “ejemplo”.

   En una consola DOS, estando posicionados dentro del directorio del proyecto ejemplo, ejecutar el siguiente comando:

   mvn -f core/pom.xml andromdapp:schema -Dtasks=create

5. Desplegar la aplicación generada.

   En la consola DOS ejecutar el siguiente comando, estando dentro del directorio del proyecto ejemplo:

   mvn -f web/pom.xml –Ddeploy

6. Probar el funcionamiento de la aplicación.

Ejecutar un browser de Internet y abrir la siguiente dirección:

http://localhost:8080/proyectoEjemplo-web-1.0/

Como resultado, se debe desplegar una página con el menú de opciones de la aplicación.

5. Manejo de posibles errores.

Durante la ejecución del procedimiento de instalación tanto del *cartridge* de Spring como del proyecto ejemplo, es posible que el procedimiento falle. Esto debido a mientras se compilan estos proyectos, son descargadas todas las dependencias que estos tienen desde Internet, en repositorios alojados por en los sitios Web de AndroMDA u otros que pueden ser configurados. En ocasiones la descarga de estas dependencias puede fallar. La solución que se puede usar para afrontar este problema es simplemente repetir el procedimiento de instalación descrito en este manual, que con seguridad, después de pocos intentos finalmente logrará descargar todas las dependencias requeridas.

CONCLUSIONES

- Por medio de herramientas que apoyan el desarrollo de software se logró extender la funcionalidad de AndroMDA en cuanto a la generación de aplicaciones CRUD que permitieran la construcción de aplicaciones Web basadas en Spring MVC.
- Se logró realizar la integración de componentes existentes de AndroMDA con la nueva funcionalidad desarrollada en este trabajo de grado.
- Se logró generar las capas de negocio y persistencia de aplicaciones Web basados en Spring Framework e Hibernate a través de la funcionalidad implementada como extensión del proyecto AndroMDA.
- Se contribuyó con la comunidad de investigación de AndroMDA en cuanto a una nueva funcionalidad del proyecto, documentando y poniendo a disposición de las personas interesadas el proyecto de grado para su usabilidad.
- Se contribuyó con el desarrollo de una nueva funcionalidad de AndroMDA utilizando tecnologías MDA que faciliten la labor de creación de aplicaciones Web utilizadas en las organizaciones actuales.
- Se aplicaron las técnicas para el desarrollo de software aprendidas durante el desarrollo de la carrera y la experiencia laboral.
- Se genero una documentación guía que permita la posterior mejora del proyecto hacia nuevas tecnologías.
- Se deja el proyecto abierto y documentado disponible para la comunidad de desarrollo de AndroMDA con el fin de que este proyecto se pueda

mejorar y tomar como punto de partida para la posterior creación de un "cartridge" completo de generación de capa de presentación usando Spring MVC, tomando en cuenta que este proyecto se limita solo a generación de los componentes Web para una aplicación CRUD

- Se aprendió sobre las diferentes alternativas que se pueden emplear para lograr extender la funcionalidad de AndroMDA.
- Es importante tener en cuenta las versiones de los diferentes componentes de software que se deben utilizar para poder hacer uso de AndroMDA.
- Este trabajo se puede complementar con la creación de un "cartridge" independiente que permita la generación no solo de CRUD's sino también de toda la capa de presentación que se puede obtener con AndroMDA a partir de diagramas de casos de uso.
- En el medio, no es común el uso de tecnologías MDA. Seria de gran importancia realizar una labor de evangelización de las ventajas que se pueden obtener, al utilizar este tipo de tecnologías.

GLOSARIO

A continuación se relacionan términos y siglas más utilizadas en este documento con su correspondiente definición:

AJAX (Asynchronous JavaScript And XML): "Es una técnica de desarrollo Web para crear aplicaciones interactivas o RIA (Rich Internet Applications). Éstas se ejecutan en el cliente, es decir, en el navegador de los usuarios y mantiene comunicación asíncrona con el servidor en segundo plano. De esta forma es posible realizar cambios sobre la misma página sin necesidad de recargarla. Esto significa aumentar la interactividad, velocidad y usabilidad en la misma."[12]

AOP (Aspect-oriented programming): "Es un paradigma de programación relativamente reciente cuya intención es permitir una adecuada modularización de las aplicaciones y posibilitar una mejor separación de conceptos. Gracias a AOP se pueden encapsular los diferentes conceptos que componen una aplicación en entidades bien definidas, eliminando las dependencias entre cada uno de los módulos. De esta forma se consigue razonar mejor sobre los conceptos, se elimina la dispersión del código y las implementaciones resultan más comprensibles, adaptables y reusables. Varias tecnologías con nombres diferentes se encaminan a la consecución de los mismos objetivos y así, el término AOP es usado para referirse a varias tecnologías relacionadas como

[12] http://java.sun.com/developer/technicalArticles/J2EE/AJAX/

los métodos adaptivos, los filtros de composición, la programación orientada a sujetos o la separación multidimensional de competencias."[13]

API (Application Programming Interface): "Interfaz de Programación de Aplicaciones - Es el conjunto de funciones y procedimientos (o métodos si se refiere a programación orientada a objetos) que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción. Una API representa una interfaz de comunicación entre componentes software. Se trata del conjunto de llamadas a ciertas bibliotecas que ofrecen acceso a ciertos servicios desde los procesos y representa un método para conseguir abstracción en la programación, generalmente (aunque no necesariamente) entre los niveles o capas inferiores y los superiores del software."[14]

Cartridge: Contiene un conjunto de reglas de transformación e incluso en algunos casos, de verificación de modelos, las cuales, cuando son aplicadas a un modelo de entrada, chequean el modelo para comprobar si es correcto con respecto a la transformación implementada por el cartucho MDA. Se instala como un plugin e implantan las reglas de transformación.

CIM (Computational-Independent Model): "Modelo independiente de la computación, se centra en los requerimientos y representa el nivel más alto del modelo de negocio."[15]

---

[13] http://www.developer.com/design/article.php/3308941
[14] http://www.sei.cmu.edu/str/descriptions/api.html
[15] http://www.epidataconsulting.com/tikiwiki/tiki-read_article.php?articleId=55

CRUD (Create, Read, Update and Delete): "Es el acrónimo de Crear, Obtener, Actualizar y Borrar. Es usado para referirse a las funciones básicas en bases de datos o la capa de persistencia en un sistema de software."[16]

DAO (Data Access Object): "Es un componente de software que suministra una interfaz común entre la aplicación y uno o más dispositivos de almacenamiento de datos, tales como una Base de datos o un archivo. El término se aplica frecuentemente al Patrón de diseño Object."[17]

DSDM (Dynamic Systems Development Model): "DSDM es un método que provee un framework para el desarrollo ágil de software, apoyado por su continua implicación del usuario en un desarrollo iterativo y creciente que sea sensible a los requerimientos cambiantes, para desarrollar un sistema que reúna las necesidades de la empresa en tiempo y presupuesto."[18]

EJB (Enterprise JavaBeans): "Los Enterprise JavaBeans (también conocidos por sus siglas EJB) son una de las API que forman parte del estándar de construcción de aplicaciones empresariales J2EE de Sun Microsystems (ahora JEE 5.0)."[19]

Framework: "Es una estructura de soporte definida,en la cual otro proyecto de software puede ser organizado y desarrollado. Típicamente, un framework

[16] www.novicksoftware.com/Articles/crud-operations-using-sql-server-stored-procedures-part-1.htm
[17] http://java.sun.com/blueprints/patterns/DAO.html
[18] http://www.lcc.uma.es/~av/MDD-MDA/
[19] http://java.sun.com/products/ejb/

puede incluir soporte de programas, bibliotecas y un lenguaje interpretado entre otros software para ayudar a desarrollar y unir los diferentes componentes de un proyecto. Un framework representa una arquitectura de software que modela las relaciones generales de las entidades del dominio. Provee una estructura y una metodología de trabajo la cual extiende o utiliza las aplicaciones del dominio. Un framework no es más que una base de programación que atiende a sus descendientes (manejado de una forma estructural y/o en cascada), posibilitando cualquier respuesta ante las necesidades de sus miembros, o secciones de una aplicación."[20]

Hibernate: Es un framework de persistencia de objetos, de código abierto, que encaja dentro del concepto de ORM (Object Relational Mapping). Es usado, para realizar un mapeo entro entidades de base de datos y objetos Java, encargándose este de convertir un objeto en un registro de base de datos y viceversa.

JSF (Java Server Faces): "Es un framework para aplicaciones Java basadas en Web que simplifica el desarrollo de interfaces de usuario en aplicaciones Java EE. JSF usa JavaServer Pages (JSP) como la tecnología que permite hacer el despliegue de las páginas."[21]

JSP (JavaServer Pages): "Es una tecnología Java que permite generar contenido dinámico para Web, en forma de documentos HTML, XML o de otro

[20] http://soaagenda.com/journal/articulos/que-son-los-frameworks/
[21] http://java.sun.com/javaee/javaserverfaces/

tipo. Esta tecnología es un desarrollo de la compañía Sun Microsystems. La Especificación JSP 1.2 fue la primera que se liberó y en la actualidad está disponible la Especificación JSP 2.1.

Las JSP's permiten la utilización de código Java mediante scripts. Además es posible utilizar algunas acciones JSP predefinidas mediante etiquetas. Estas etiquetas pueden ser enriquecidas mediante la utilización de Librerías de Etiquetas (TagLibs o Tag Libraries) externas e incluso personalizadas."[22]

Layers: "Un layer o capa es un soporte que contiene cierta información, ya sea gráfica, lógica o de cualquier otra índole y que es, en sí mismo, un objeto integral e independiente."[23]

MDE (Model-Driven Engineering): "Es el uso sistemático de modelos como principales artefactos de ingeniería en todo el ciclo de vida de ingeniería. MDE puede ser aplicado a software, sistema, y los datos de ingeniería. Los modelos son considerados como entidades de primera clase."[24]

MVC: "Modelo Vista Controlador (MVC) es un patrón de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario y la lógica de control en tres componentes distintos. El patrón MVC se ve frecuentemente en aplicaciones Web, donde la vista es la página HTML y el código que provee de datos dinámicos a la página, el modelo es el Sistema de

[22] http://java.sun.com/products/jsp/
[23] www.cenart.gob.mx/data.lab.02/programas/versionh/layers.htm
[24] www.cs.colostate.edu/~ghosh/modevva2008/

Gestión de Base de Datos y la Lógica de negocio y el controlador es el responsable de recibir los eventos de entrada desde la vista."[25]

OCL (Object Constraint Language): "Es un lenguaje para la descripción formal de expresiones en los modelos UML. Sus expresiones pueden representar *invariantes*, *precondiciones*, *postcondiciones*, *inicializaciones*, *guardias*, *reglas de derivación*, así como *consultas* a objetos para determinar sus condiciones de estado. Se trata de un lenguaje sin efectos de borde, de manera que la verificación de una condición, que se presupone una operación instantánea, nunca altera los objetos del modelo. Su papel principal es el de completar los diferentes artefactos de la notación UML con requerimientos formalmente expresados."[26]

PDM (Platform-Definition Model): "Modelo de definición de la plataforma, representa los diferentes tipos de elementos que intervienen en la plataforma de un sistema."[27]

PIM (Platform-Independent Model): "Modelo independiente de la plataforma, representa el modelo de procesos del negocio a ser implementado."[28]

Plugin: "Un plugin o componente enchufable es una aplicación informática que interactúa con otra aplicación para aportarle una función o utilidad específica,

[25] http://ootips.org/mvc-pattern.html
[26] http://www.omg.org/technology/documents/modeling_spec_catalog.htm
[27] http://arquivosevt.lncc.br/pdfs/2008-mda.aula4.pdf
[28] http://www.epidataconsulting.com/tikiwiki/tiki-read_article.php?articleId=55

generalmente muy específica, como por ejemplo servir como driver (controlador) en una aplicación, para hacer así funcionar un dispositivo en otro programa. Esta aplicación adicional es ejecutada por la aplicación principal. Los plugin´s típicos tienen la función de reproducir determinados formatos de gráficos, reproducir datos multimedia, codificar/decodificar e-mails, filtrar imágenes de programas gráficos, entre otros."[29]

POJO (Plain Old Java Object): "Es una sigla creada por Martin Fowler, Rebecca Parsons y Josh MacKenzie en septiembre de 2000 y utilizada por programadores de lenguaje de programación Java, para enfatizar el uso de clases simples y que no dependen de un framework en especial. Este acrónimo surge como una reacción en el mundo Java a los frameworks cada vez más complejos, y que requieren un complicado andamiaje que esconde el problema que realmente se está modelando. En particular surge en oposición al modelo planteado por los estándares EJB anteriores al 3.0, en los que los "Enterprise JavaBeans" debían implementar interfaces especiales.

POJO es una nueva palabra para designar algo viejo. No existe en Java una nueva tecnología con ese nombre, sino que el nombre existe en el marco de una revalorización de la programación "simplemente orientada a objetos". Esta revalorización tiene que ver también con el éxito de lenguajes orientados a objetos más puros y sencillos, que empezaron a tomar parte del mercado al que Java apunta."[30]

[29] http://www.learningmovabletype.com/a/000497what_is_plugin/
[30] http://c2.com/cgi/wiki?PlainOldJavaObject

PSM (Plantform-specific Model): "Modelo específico de la plataforma, representa la proyección de los PIM´s en una plataforma específica."[31]

QVT (Queries/Views/Transformations): "es un estándar para la transformación de modelos que permite a los modelos UML ser transformados en software."[32]

SOA (Service Oriented Architecture): "Arquitectura Orientada a Servicios, es un concepto de arquitectura de software que define la utilización de servicios para dar soporte a los requerimientos de software del usuario. SOA es una arquitectura de software que permite la creación y/o cambios de los procesos de negocio desde la perspectiva de TI de forma ágil, a través de la composición de nuevos procesos utilizando las funcionalidades de negocio que están contenidas en la infraestructura de aplicaciones actuales o futuras (expuestas bajo la forma de Web Services)."[33]

STRUTS: "Struts es una herramienta de soporte para el desarrollo de aplicaciones Web bajo el patrón MVC bajo la plataforma J2EE (Java 2, Enterprise Edition). Struts se desarrollaba como parte del proyecto Jakarta de la Apache Software Foundation, pero actualmente es un proyecto independiente conocido como Apache Struts.

[31] http://www.epidataconsulting.com/tikiwiki/tiki-read_article.php?articleId=55
[32] www.omg.org/mda/mda_files/GC34-2485-03_LoRes.pdf
[33] http://www.udistrital.edu.co/comunidad/grupos/arquisoft/index.php?id=78&type=1

Struts permite reducir el tiempo de desarrollo. Su carácter de "software libre" y su compatibilidad con todas las plataformas en que Java Entreprise esté disponible, lo convierte en una herramienta altamente disponible."[34]

UML (Unified Modeling Language): "Es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad; está respaldado por el OMG (Object Management Group). Es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema de software. UML ofrece un estándar para describir un "plano" del sistema (modelo), incluyendo aspectos conceptuales tales como procesos de negocios y funciones del sistema, y aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y componentes de software reutilizables. Es importante resaltar que UML es un "lenguaje" para especificar y no para describir métodos o procesos. Se utiliza para definir un sistema de software, para detallar los artefactos en el sistema y para documentar y construir."[35]

Web Service: "Es un conjunto de protocolos y estándares que sirven para intercambiar datos entre aplicaciones. Distintas aplicaciones de software desarrolladas en lenguajes de programación diferentes, y ejecutadas sobre cualquier plataforma, pueden utilizar los servicios Web para intercambiar datos en redes de ordenadores como Internet. La interoperabilidad se consigue mediante la adopción de estándares abiertos. Las organizaciones OASIS y W3C son los comités responsables de la arquitectura y reglamentación de los

[34] http://struts.apache.org/
[35] http://www-306.ibm.com/software/rational/uml/

servicios Web. Para mejorar la interoperabilidad entre distintas implementaciones de servicios Web se ha creado el organismo WS-I, encargado de desarrollar diversos perfiles para definir de manera más exhaustiva estos estándares."[36]

Workflow (Flujo de trabajo): "Es el estudio de los aspectos operacionales de una actividad de trabajo: cómo se estructuran las tareas, cómo se realizan, cuál es su orden correlativo, cómo se sincronizan, cómo fluye la información que soporta las tareas y cómo se le hace seguimiento al cumplimiento de las tareas. Generalmente los problemas de flujo de trabajo se modelan con redes de Petri. Si bien el concepto de flujo de trabajo no es específico a la tecnología de la información, una parte esencial del software para trabajo colaborativo (groupware) es justamente el flujo de trabajo.

Una aplicación de Flujos de Trabajo (WorkFlow) automatiza la secuencia de acciones, actividades o tareas utilizadas para la ejecución del proceso, incluyendo el seguimiento del estado de cada una de sus etapas y la aportación de las herramientas necesarias para gestionarlo."[37]

XDoclet: "Es un motor de Código abierto para el Lenguaje de programación Java, su función es la generación de código. Está asociado con la programación orientada a los atributos, es decir, se puede lograr más

[36] http://www.w3.org/TR/ws-arch/#whatis
[37] http://www.e-workflow.org/

funcionalidad agregándole metadata (atributos) al código. Esto se lleva a cabo con tag´s JavaDoc."[38]

XML (Extensible Markup Language): "Lenguaje de marcas extensible, es un meta-lenguaje extensible de etiquetas desarrollado por el World Wide Web Consortium (W3C). Es una simplificación y adaptación del SGML y permite definir la gramática de lenguajes específicos (de la misma manera que HTML es a su vez un lenguaje definido por SGML). Por lo tanto, XML no es realmente un lenguaje en particular, sino una manera de definir lenguajes para diferentes necesidades. Algunos de estos lenguajes que usan XML para su definición son XHTML, SVG, MathML."[39]

[38] http://xdoclet.sourceforge.net/xdoclet/index.html
[39] http://sherekan.com.ar/blog/2008/05/16/introduccion-a-xml/

BIBLIOGRAFIA

L. Cuaderno, E. Di Lorenzo, A. Gaig, D. García, R. Giandini, L. Nahuel, L. Ocaranza, M. Pinasco, C. Pons, F. Salvatierra. Herramientas de soporte al proceso de desarrollo dirigido por modelos y su implementación con DSL Tools. La Plata, Buenos Aires, Argentina, 2007. Universidad Tecnológica Nacional, Facultad Regional.

MDA Guide Version 1.0.1, en: www.omg.org/docs/omg/03-06-01.pdf, (visita: marzo 8 de 2008)

Model Driven Architecture, en: http://martinfowler.com/bliki/ModelDrivenArchitecture.html, (visita: marzo 8 de 2008)

An introduction to Model Driven Architecture, en: http://www-128.ibm.com/developerworks/rational/library/3100.html, (visita: marzo 8 de 2008).

QVT, en: http://www.omg.org/mda/mda_files/GC34-2485-03_LoRes.pdf, (visita: febrero 15 de 2008).

What is a Plugin?, en: http://www.learningmovabletype.com/a/000497what_is_plugin/, (visita: febrero 15 de 2008).

MDA: Reusabilidad Orientada al Negocio, en: http://www.epidataconsulting.com/tikiwiki/tiki-read_article.php?articleId=55, (visita: marzo 15 de 2008).

Getting Started with UML, en: http://www.uml.org/, (visita: marzo 15 de 2008).

Unified Modeling Language, en: http://www-306.ibm.com/software/rational/uml/, (visita: marzo 15 de 2008).

Lenguaje UML, en: http://www.hipertexto.info/documentos/uml.htm, (visita: marzo 15 de 2008).

¿Qué es un framework Web?, en: http://www.lsi.us.es/~javierj/investigacion_ficheros/Framework.pdf, (visita: marzo 8 de 2008)

Que son los FrameWorks, en: http://soaagenda.com/journal/articulos/que-son-los-frameworks/, (visita: marzo 8 de 2008)

DSDM: Desarrollo de Software Dirigido por Modelos. MDA y Aplicaciones, en: http://www.lcc.uma.es/~av/MDD-MDA/, (visita: febrero 23 de 2008).

Dynamic Systems Development Model (DSDM) Methodology, en: http://www.mariosalexandrou.com/methodologies/dynamic-systems-development-model.asp, (visita: febrero 23 de 2008).

Enterprise Architect - Herramienta de diseño UML, en: http://www.sparxsystems.com.ar/products/ea.html, (visita: febrero 8 de 2008).

OMG Model Driven Architecture, en: http://www.omg.org/mda/ , (visita: abril 16 de 2008).

AndroMDA, en: www.andromda.org/, (visita: abril 16 de 2008).

Cutting Edge MDSD/MDA Toolkit, en: http://www.andromda.org/, (visita: abril 16 de 2008)

Implementing CRUD operations using Stored Procedures: Part 1, en: http://www.novicksoftware.com/Articles/crud-operations-using-sql-server-stored-procedures-part-1.htm, (visita: abril 16 de 2008)

LAYERS, sobre el diseño y nuevos medios, en: http://www.cenart.gob.mx/data.lab.02/programas/versionh/layers.htm, (visita: mayo 21 de 2008)

Welcome! What is XDoclet?, en: http://xdoclet.sourceforge.net/xdoclet/index.html, (visita: mayo 21 de 2008)

Introducción a XML, en: http://sherekan.com.ar/blog/2008/05/16/introduccion-a-xml/, (visita: mayo 25 de 2008)

Model-View-Controller, en: http://ootips.org/mvc-pattern.html, (visita: mayo 25 de 2008)

Getting started with AndroMDA for Java, en: http://galaxy.andromda.org/docs/getting-started/java/index.html, (visita: mayo 3 de 2008)

What is Workflow?, en: http://www.e-workflow.org/, (visita: mayo 3 de 2008)

Application Programming Interface, en: http://www.sei.cmu.edu/str/descriptions/api.html, (visita: mayo 3 de 2008)

Modelos que intervienen en MDA, en: http://arquivosevt.lncc.br/pdfs/2008-mda.aula4.pdf, (visita: mayo 25 de 2008)

Object Constraint Language (OCL), en: http://www.omg.org/technology/documents/modeling_spec_catalog.htm, (visita: mayo 25 de 2008)

Arquitectura Orientada a Servicios (SOA:Services Oriented Architecture), en: http://www.udistrital.edu.co/comunidad/grupos/arquisoft/index.php?id=78&type=1, (visita: mayo 25 de 2008)

Apache Struts, en: http://struts.apache.org/, (visita: mayo 25 de 2008)

JavaServer Faces Technology, en: http://java.sun.com/javaee/javaserverfaces/, (visita: mayo 25 de 2008)

Enterprise JavaBeans Technology, en: http://java.sun.com/products/ejb/, (visita: mayo 25 de 2008)

Data Access Object, en: http://java.sun.com/blueprints/patterns/DAO.html, (visita: mayo 25 de 2008)

Web Services Architecture, en: http://www.w3.org/TR/ws-arch/#whatis, (visita: mayo 25 de 2008)

Spring Framework, en: http://www.springframework.org, (visita: septiembre 23 de 2007)

Relational Persistence for Java and .NET, en: http://www.hibernate.org, (visita: septiembre 23 de 2007)

Struts, en: http://struts.apache.org, (visita: septiembre 23 de 2007)

Plain Old Java Object, en: http://c2.com/cgi/wiki?PlainOldJavaObject, (visita: mayo 25 de 2008)

JavaServer Pages Technology, en: http://java.sun.com/products/jsp/, (visita: mayo 25 de 2008)

Asynchronous JavaScript Technology and XML (Ajax) With the Java Platform, en: http://java.sun.com/developer/technicalArticles/J2EE/AJAX/, (visita: mayo 25 de 2008)

Aspect Oriented Programming, en: http://www.developer.com/design/article.php/3308941, visita: mayo 25 de 2008)