PROCEDIMIENTO PARA LA APLICACIÓN DE CONCEPTOS DEL DESARROLLO DIRIGIDO POR MODELOS, MEDIANTE EL USO DEL FRAMEWORK SPRING.

Autor

FRANCISCO JAVIER GERALDINO FERNÁNDEZ

UNIVERSIDAD DE PAMPLONA

FACULTAD DE INGENIERÍAS Y ARQUITECTURA

DEPARTAMENTO DE ELÉCTRICA, ELECTRÓNICA, SISTEMAS Y TELECOMUNICACIONES

INGENIERÍA DE SISTEMAS

PAMPLONA, COLOMBIA

2017

PROCEDIMIENTO PARA LA APLICACIÓN DE CONCEPTOS DEL DESARROLLO DIRIGIDO POR MODELOS, MEDIANTE EL USO DEL FRAMEWORK SPRING.

Autor

FRANCISCO JAVIER GERALDINO FERNÁNDEZ

TRABAJO PRESENTADO PARA OPTAR POR EL TITULO

DE INGENIERO DE SISTEMAS

Director

LUIS ALBERTO ESTEBAN VILLAMIZAR

Licenciado en Matemáticas y Computación

Magister en Informática

UNIVERSIDAD DE PAMPLONA

FACULTAD DE INGENIERÍAS Y ARQUITECTURA

DEPARTAMENTO DE ELÉCTRICA, ELECTRÓNICA, SISTEMAS Y TELECOMUNICACIONES

INGENIERÍA DE SISTEMAS

PAMPLONA, COLOMBIA

2017

Procedimiento para la aplicación de conceptos del desarrollo dirigido por modelos, mediante el uso del framework spring.

Resumen Del Proyecto

Los modelos forman parte del gran conjunto de herramientas de ingeniería y su utilización nace de la necesidad de comprender y describir sistemas complejos. El desarrollo dirigido por modelos, como todos los paradigmas de la ingeniería busca dar solución a problemas o que el desarrollo de una actividad considerada como compleja se lleve a cabo de forma "fácil" haciendo uso eficiente de los recursos. El desarrollo dirigido por modelos (MDD) aparece como solución de la necesidad de seguir realizando actividades altamente adaptables, ofreciendo grandes capacidades de adaptabilidad, rehúso y fácil comprensión, esta última proporcionada por los modelos que describen la actividad.

El uso de modelos como base fundamental en el desarrollo dirigido por modelos y hace necesaria la existencia de etapas en el proceso y de cambios en los modelos, con el fin de que esos modelos se conviertan en un producto final. El MDD plantea una serie de conversiones para que un modelo pueda llegar a un producto final.

Las etapas de este proceso empieza con la necesidad de describir el sistema a construir generando los modelos independientes de plataforma, estos modelos describen un sistema de forma general, proporcionando información de cómo es el funcionamiento inicial. Estos modelos pasan por un proceso de conversión de modelo a modelo dando como resultado los modelos sujetos a plataforma o dependientes de plataforma, estos modelos como su nombre lo indican son los modelos que dan los detalles de estructura, elementos e interacciones entre los elementos. Estos últimos modelos son los que se convierten en código realizando una conversión de modelo a texto, el texto o código generado es el que finalmente se convertirá en el sistema.

Inicialmente se hace necesaria la fundamentación teórica correspondiente al paradigma de desarrollo dirigido por modelos, dentro de la fundamentación teórica se encuentra el ciclo de vida del DDM en conjunto con las definiciones básicas de modelo y los tipos de modelos presentes en este paradigma. Basado en la fundamentación teórica y teniendo presente las etapas del desarrollo dirigido por modelos se crea un procedimiento de desarrollo dirigido por modelos en Spring framework que posteriormente se implementó en la creación de tres funcionalidades básicas de un sistema web.

Tabla de Contenidos

Lista de Ilustraciones

1 Introducción

El desarrollo dirigido por modelos es un paradigma que nace de la necesidad de aprovechamiento de recursos y tiempos, proponiendo una metodología distinta, esta metodología ofrece al desarrollador beneficios como la reutilización, fácil comprensión y modificación, entre otras. Los beneficios son solo para quien aplica la metodología, el usuario común o ajeno a la metodología tiene la facilidad de comprender e interpretar la información que ofrece la metodología puesto que la existencia de modelos que describen el sistema facilita la interpretación del mismo.

Este documento contiene la fundamentación teórica relacionada con el paradigma de desarrollo dirigido por modelos, partiendo desde su concepto, la forma en cómo surge y las dificultades que se plantearon superar con esta metodología, las condiciones que se planea para su implementación y beneficios que se obtienen haciendo uso de la metodología de desarrollo dirigido por modelos.

El uso de metodologías o herramientas que permitan el desarrollo ágil de aplicaciones se ha vuelto casi que una necesidad debido a las altas exigencias de tiempo y calidad de desarrollo de aplicaciones, sumado la necesidad del acceso “instantáneo” y global de la información por medio de aplicaciones web; hace imprescindible encontrar herramientas que en conjunto con metodologías permitan que el desarrollo de dichas aplicaciones no sea una tarea extensa y en donde se deba partir desde cero. Existen herramientas como los frameworks que permiten desarrollar aplicaciones brindando un núcleo personalizable y ajustable a las necesidades, estos frameworks en conjunto con una metodología como el desarrollo dirigido por modelos permitirá la creación de una aplicación a partir de un modelo descriptivo de la aplicación.

1.1 Justificación

La tendencia al uso de aplicaciones con acceso desde cualquier parte con el uso de internet ha llevado a que el desarrollo de aplicaciones web se convierta en tendencia, brindando accesibilidad y disponibilidad. El uso de herramientas para la creación de aplicaciones web es diversa y enfocada a diferentes ámbitos del desarrollo como el frontend y backend. El frontend tiene herramientas que manejan JQuery, JavaScript, Html, Css entre otros, de estas herramientas se pueden destacar algunas como Bower y Modernizr.

Para el backend aparecen herramientas que ayudan a la construcción de la estructura de software e incluso algunos ofrecen unas bases para el desarrollo, portando funcionalidades predefinidas, estas herramientas son conocidas como frameworks. Destacar y hacer uso del Spring Framework es importante, ya que esta herramienta permite el desarrollo de aplicaciones haciendo uso de metodologías de desarrollo como MDD (Desarrollo dirigido por modelos o model driven development) con un módulo integrado en su núcleo llamado spring roo.

La utilización de Spring Framework y su módulo Spring Roo permitirá que el desarrollo de funcionalidades de una aplicación permitiendo la verificación de la efectividad de la metodología de desarrollo dirigido por modelos basados en la amplia documentación ofrecida por la comunidad.

1.2 Objetivos

1.2.1 Objetivo general

Diseñar un procedimiento para la aplicación de desarrollo dirigido por modelos en java haciendo uso del entorno de desarrollo Spring Framework.

1.2.2 Objetivos Específicos

- Realizar un estudio bibliográfico sobre el desarrollo dirigido por modelos y el framework Spring.
- Definir los pasos necesarios para la aplicación del desarrollo dirigido por modelos e n la construcción de aplicaciones web con el framework spring.
- Validar el procedimiento propuesto mediante el desarrollo de por lo menos tres func ionalidades de una aplicación web tales como login, registro de usuario y crud (crear, leer, modificar y borrar) contenido.

1.3 Acotaciones

Para el desarrollo del proyecto se hará uso de herramientas que ayuden al desarrollo como Netbeans o Eclipse como IDE de desarrollo, en cualquiera de los dos casos haciendo uso del plugin Spring Tool Suite que contiene la estructura de Spring Framework, el uso de Java Development Kit en su versión 6, Navegador web y una computadoras.

El desarrollo de Funcionalidades de una aplicación se llevará a cabo con el fin de realizar una demostración de cómo serían los resultados haciendo uso del procedimiento a desarrollar.

1.4 Metodología de trabajo

El desarrollo dirigido por modelos MDD ofrece soluciones a problemas comunes en el desarrollo de software como la reutilización y fácil comprensión del modelo por parte de usuarios finales y programadores. Entender la forma en cómo se lleva a cabo el desarrollo de software dirigido por modelos para abstraer ese conocimiento y aplicarlo en entornos comunes en el desarrollo de software como son los framework generando un procedimiento que variaría según la plataforma utilizada. Basándose en la premisa de generación de un procedimiento, se establecen los elementos, herramientas y entornos a integrar en el proceso de generación de dicho procedimiento.

Estudio de Spring: Spring es un framework que al igual que otros framework ofrece gran cantidad de posibilidades, particularmente spring, ofrece herramientas integradas que para el caso resultan ser de gran utilidad para el desarrollo dirigido por modelos.

Para el estudio de spring framework se partió desde los conceptos básicos, estos conceptos básicos siempre formaran parte de la base de conocimiento, permitiendo la implementación de estos en procedimientos más complejos. Dentro de los temas fundamentales de spring se encuentra la definición de métodos de resolución de peticiones o controladores, quienes se encargan de atender el tipo de petición se está realizando e indicar la respuesta que se le dará a esa petición. La inyección de dependencia, anotaciones, sobrecarga de métodos, Beans y otros, destacan en el conjunto de conceptos base de spring.

Estudio de Spring roo: Spring roo es una herramienta importante en el desarrollo dirigido por modelos, brindando esa reusabilidad y capacidad de adaptación a los cambios. Comprender el funcionamiento de esta herramienta y la integración con el propio framework y otras aplicaciones integradas con spring facilita el desarrollo de actividades en donde el desarrollo rápido y "simple" de aplicaciones es fundamental.

Estudio de los principios de Desarrollo dirigido por modelos (MDD): Comprender la estructura y objetivos del desarrollo dirigido por modelos empieza con el estudio del concepto básico de lo que es un modelo dirigido por modelos, teniendo en cuenta que el desarrollo dirigido por modelos es tan amplio que es aplicable a diversas áreas y la conceptualización de este concepto al desarrollo dirigido por modelos (MDD) es justamente nombrado como desarrollo dirigido por modelos, tomando los conceptos básicos de MDD ya aplicándolos al desarrollo de software. Vale la pena aclarar que la variación entre desarrollo dirigido por modelos y desarrollo de software dirigido por modelos radica en la utilización de diferente tipos de modelo, esta variación se presenta según el contexto de aplicación.

Definición de los principales pasos para la aplicación de MDD con Spring: El desarrollo dirigido por modelos define una estructura básica que describe de forma general los elementos y cambios de los elementos en el transcurso de la ejecución de MDD. De acuerdo al proceso de MDD se identificaron un conjunto de pasos que empiezan con el análisis de lo que se desea construir, un levantamiento de requerimientos del sistema a construir, teniendo en cuenta los requerimientos del sistema el siguiente paso es generar un modelo que plasme lo establecido por los requerimientos, este modelo es opcional y es utilizado para mostrar

como es el funcionamiento del sistema. Teniendo en cuenta el modelo general de descripción o directamente los requerimientos establecidos al principio, se crean los modelos independientes de plataforma que son modelos fáciles de interpretar puesto no están limitados a una tipología, estos modelos pueden ir desde un dibujo, storyboards hasta un diagrama UML. Los modelos independientes de plataforma son ideales para describir un sistema pero no muestran el comportamiento de cada uno de los elementos identificados según la plataforma en la que se está trabajando, traducir estos modelos es uno de los pasos necesarios, convirtiendo modelos independientes a plataforma a modelos más cercano a la plataforma elegida permite identificar componentes que están ligados directamente a una plataforma tales como un controlador o un Bean. Teniendo identificados los elementos que constituyen el sistema es hora de convertir los modelos sujetos a plataforma a código que es donde se generarán los elementos identificados, esa generación de elementos y código se llevará a cabo con spring roo y unas instrucciones para modificar o personalizar el sistema.

Planificación de la aplicación de procedimiento: El procedimiento se implementó siguiendo las recomendaciones definidas en el mismo, pero se llevó a cabo una etapa de revisión de implementación de modelos ya sea sujetos a plataforma o independientes de plataforma, con el fin de identificar si los modelos planteados estaban definidos de tal manera que llegaran a describir el funcionamiento real el sistema y finalmente teniendo todos los modelos se procede a generar los códigos necesarios (base de datos/SQL, MVC).

Definir los alcances del prototipo de aplicación web: Los prototipos de esta aplicación están definidos para tres funcionalidades que se construyeron aplicando el procedimiento planteado y que se ajustaron modificando secciones de código.

Definir los modelos independientes de plataforma: Los modelos independientes de plataforma que se crearon para la descripción de esta aplicación varían desde un caso de uso hecho en UML hasta un storyboard que muestra de manera general como es el funcionamiento del sistema, describiendo implícitamente algunas interacciones.

Describir los modelos independientes de plataforma: La descripción de los PMI se realizó

Convertir los modelos independientes de plataforma a modelos dependientes de plataforma: Los modelos dependientes de plataforma o sujetos de plataforma surgen después de la identificación de elementos e interacciones presentes en los PIM, en conjunto con esto se tuvo en cuenta la estructura de directorios y archivos que por defecto posee spring. Con estas consideraciones anteriores se definieron como modelos sujetos a plataforma el modelo entidad relación, diagramas de secuencia y estructura de archivos de spring.

Generar código a partir de modelos dependientes de plataforma: Teniendo definido los modelos sujetos a plataforma solo queda pasar esos modelos a código, se tuvo en cuenta que el desarrollo de este conjunto de funcionalidades está basado en spring y que se haría uso de spring roo. Los modelos entidad relación se convirtieron en código SQL, esta conversión se lleva a cabo tomando en consideración las indicaciones establecidas para generar SQL. Los diagramas de secuencia describiendo la interacción entre distintas capas la transformación de este modelo a código la lleva a cabo spring roo integrando el núcleo de spring con Jpa e Hibernate, convirtiéndose en un proceso transparente para el usuario y que finalmente genera el modelo mvc con sus correspondientes vistas.

Refinación manual de código: La refinación manual del código se llevó a cabo cuando se insertaron segmentos de código ya sea para modificar una funcionalidad, agregar llaves foráneas entre entidades o ingresar una rutina personalizada en el código fuente del proyecto. Estas modificaciones se realizaron en la personalización del Login, en la designación de los accesos a funcionalidades-

Pruebas de funcionalidades: El conjunto de pruebas que se llevó a cabo en las funcionalidades creadas se realizaron con el fin de probar el funcionamiento de cada una de estas funcionalidades, comprobando el correcto funcionamiento de cada una de las funcionalidades.

2 Marco teórico y estado del arte

El desarrollo dirigido por modelos o MDD, es un nuevo paradigma basado netamente en modelos aplicable en muchas áreas, la aplicación en las tecnologías de la información (TI) busca ofrecer mejoras en la eficiencia y calidad de soluciones de desarrollo de software. Teniendo en cuenta que el artefacto principal del MDD son los modelos, puesto estos son utilizados para la interpretación del sistema e incluso como punto de partida para la generación de código u otros modelos.

2.1 Marco conceptual

MDD (*model-driven development*) es un paradigma de desarrollo de software que utiliza modelos para diseñar los sistemas a distintos niveles de abstracción, y secuencias de transformaciones de modelos para generar unos modelos a partir de otros hasta generar el código final de las aplicaciones en las plataformas destino (Durán Muñoz, Troya Castilla, & Vallecillo Moreno).

MDA (*model-driven architecture*) es la propuesta concreta de la OMG para implementar MDD, usando las notaciones, mecanismos y herramientas estándares definidos por esa organización (Durán Muñoz, Troya Castilla, & Vallecillo Moreno).

MDE (*model-driven engineering*) es un paradigma dentro de la ingeniería del software que aboga por el uso de los modelos y las transformaciones entre ellas como piezas clave para dirigir todas las actividades relacionadas con la ingeniería del software (Durán Muñoz, Troya Castilla, & Vallecillo Moreno).

MBE (*model-based engineering*) es un término general que describe los enfoques dentro de la ingeniería del software que usan modelos en algunos de sus procesos o actividades (Durán Muñoz, Troya Castilla, & Vallecillo Moreno).

2.2 Marco teórico

El uso de modelos para el diseño de sistemas complejos es de rigor en las disciplinas de ingeniería tradicionales. No se puede imaginar la construcción de un edificio tan complejo como un puente o un automóvil sin primero construir una variedad de modelos de sistemas especializados. Los modelos ayudan a entender un problema complejo y sus posibles soluciones a través de la abstracción. Por lo tanto, parece evidente que los sistemas de software, que son a menudo uno de los sistemas de ingeniería más complejos, pueden beneficiarse enormemente del uso de modelos y técnicas de modelado (Bran , 2008).

2.2.1 Modelo

Un modelo se puede definir como una representación con detalles característicos de una actividad, fenómeno, objeto o persona, según (Durán Muñoz, Troya Castilla, & Vallecillo Moreno) modelo se puede definir de las siguientes formas:

- Un modelo es una **descripción** de un sistema, o parte de este, escrito en un lenguaje bien definido.

- Un modelo es una **representación** de una parte de la funcionalidad, estructura y/o comportamiento de un sistema

- Un modelo es una **descripción** o **especificación** de un sistema y su entorno definida para cierto propósito

Aunque no hay consenso sobre cuáles son las características que deben tener los modelos para ser considerados útiles y efectivos, una de las descripciones más acertadas es la de Bran Selic, uno de los fundadores del MDD y pionero en el uso de sus técnicas. Según él, los modelos deberían ser:

- **Adecuados**: Construidos con un propósito concreto, desde un punto de vista determinado y dirigidos a un conjunto de usuarios bien definido.
- **Abstractos**: Enfatizan los aspectos importantes para su propósito a la vez que ocultan los aspectos irrelevantes.
- **Comprensibles**: Expresados en un lenguaje fácilmente entendible para sus usuarios.
- **Precisos**: Representan fielmente al objeto o sistema modelado.
- **Predictivos**: Pueden ser usados para responder preguntas sobre el modelo e inferir conclusiones correctas.
- **Rentables**: Han de ser más fáciles y baratos de construir y estudiar que el propio sistema.

Los modelos ofrecen ciertas utilidades importantes en el desarrollo de software ya que esas características ayudan a comprender el sistema, es una excelente herramienta de comunicación por su fácil interpretación y finalmente pero no menos importante sirve de guía en el proceso de implementación, ocupando el papel de "planos" de construcción.

Los modelos por si solos no pueden describir de forma idónea un sistema, siempre está acompañado por unos submodelos descriptores, y la existencia de tipología de modelos ayuda a que se concentre en el tipo de información que quiere dar a entender, estas tipologías y modelos que describe otros modelos son mencionados por (Durán Muñoz, Troya Castilla, & Vallecillo Moreno) y cuyas definiciones se usaran a continuación:

2.2.2 Metamodelos

Es importante señalar el hecho de que el lenguaje que se usa para describir el modelo debe estar bien definido y ofrecer un nivel de abstracción adecuada para expresar el modelo y para razonar sobre él. En este sentido, la idea compartida por todos los paradigmas englobados dentro del MDD es la conveniencia de utilizar para el modelado lenguajes de mayor nivel de abstracción que los lenguajes de programación, esto es, lenguajes que manejen conceptos más cercanos al dominio del problema, denominados lenguajes específicos de dominio (o DSL, *domain-specific language*). Estos lenguajes requieren a su vez una descripción precisa; aunque esta puede darse de diferentes maneras, en el mundo de MDD lo normal y más apropiado es definirlos como modelos también. Como se verá con más detalle en el apartado 3, el metamodelado es una estrategia muy utilizada en la actualidad para la definición de lenguajes de modelado.

2.2.3 Tipos de modelos en el MDD

Un **modelo independiente de la plataforma** o PIM es un modelo del sistema que concreta sus requisitos funcionales en términos de conceptos del dominio y que es independiente de cualquier plataforma. Generalmente, la representación de un modelo PIM está basada en un lenguaje específico para el dominio modelado, donde los conceptos representados exhiben un cierto grado de independencia respecto a diversas plataformas; ya sean estas plataformas tecnológicas concretas (como CORBA, .NET o J2EE) o plataformas más abstractas (como por ejemplo, requisitos de seguridad o de fiabilidad). Es así como MDA puede asegurar que el modelo independiente de la plataforma (PIM) sobrevivirá a los cambios que se produzcan en futuras plataformas de tecnologías y arquitecturas software (Durán Muñoz, Troya Castilla, & Vallecillo Moreno).

Un **modelo específico para una plataforma** (PSM) es un modelo resultado de refinar un modelo PIM para adaptarlo a los servicios y mecanismos ofrecidos por una plataforma concreta (Durán Muñoz, Troya Castilla, & Vallecillo Moreno).

En MDA, una **plataforma** es un conjunto de subsistemas y tecnologías que describen la funcionalidad de una aplicación a través de interfaces y patrones específicos, facilitando que cualquier sistema que vaya a ser implementado sobre dicha plataforma pueda hacer uso de estos recursos sin tener en consideración aquellos detalles que son relativos a la funcionalidad ofrecida por la plataforma concreta.

2.2.4 Orígenes de MDD

Si bien MDD define un nuevo paradigma para el desarrollo de software, sus principios fundamentales no constituyen realmente nuevas ideas sino que son reformulaciones y asociaciones de ideas anteriores. MDD es la evolución natural de la ingeniería de software basada en modelos enriquecida mediante el agregado de transformaciones automáticas entre modelos. Por su parte, la técnica de transformaciones sucesivas tampoco es algo novedoso. Se Podría remitir al proceso de abstracción y refinamiento presentado por Edsger W. Dijkstra en su libro "A Discipline of Programming" donde se define que refinamiento es el proceso de desarrollar un diseño o implementación más detallado a partir de una especificación abstracta a través de una secuencia de pasos matemáticamente justificados que mantienen la corrección con respecto a la especificación original. Es decir, de acuerdo con esta definición, un refinamiento es una transformación semánticamente correcta que captura la relación esencial entre la especificación (es decir, el modelo abstracto) y la implementación (es decir, el código). Consecuentemente se puede señalar que el proceso MDD mantiene una fuerte coincidencia, en sus orígenes e ideas centrales, con el seminal concepto de abstracción y refinamientos sucesivos, el cual ha sido estudiado extensamente en varias notaciones formales tales como Z y B, y diversos cálculos de refinamientos. En general estas técnicas de refinamiento se limitan a transformar un modelo formal en otro modelo formal escrito en el mismo lenguaje (es decir, se modifica el nivel de abstracción del modelo, pero no su lenguaje), mientras que MDD es más amplio pues ofrece la posibilidad de transformar modelos escritos en distintos lenguajes (por ejemplo, se puede transformar un modelo escrito en UML en otro modelo escrito en notación Entidad-Relación) (Pascuas Rengifo, Mendoza Suarez, Eliana Dirley Córdoba Correa, & Córdoba Correa, 2015).

2.2.5 El ciclo de vida de desarrollo dirigido por modelos

Según (Pascuas Rengifo, Mendoza Suarez, Eliana Dirley Córdoba Correa, & Córdoba Correa, 2015) el ciclo de vida muestra los tipos de artefactos que se crean durante el proceso de desarrollo, estos son modelos formales, es decir, modelos que pueden ser comprendidos por computadores. Los modelos son una representación conceptual o física a escala de un

proceso o sistema, con el fin de analizar su naturaleza, desarrollar o comprobar hipótesis y permitir una mejor comprensión del fenómeno real al cual el modelo representa, perfeccionando los diseños, antes de iniciar la construcción de las obras u objetos reales.

El MDD identifica distintos tipos de modelos: modelos con alto nivel de abstracción, son modelos que elevan el nivel de desarrollo del software y reducen las diferencias entre los dominios de la tecnología; es decir modelos independientes de cualquier metodología computacional, llamados CIM (Computational Independent Model), los modelos independientes de cualquier tecnología de implementación llamados PIM (Platform Independent Model), modelos que especifican el sistema en términos de construcciones de implementación disponibles en alguna tecnología específica, conocidos como PSM (Platform Specific Model), y los Modelos que representan el código fuente.

Teniendo en cuenta los elementos que interactúan, se puede definir un ciclo de vida del desarrollo dirigido por modelos, este ciclo de vida está dividido en una serie de etapas, cada una de estas dejando un producto que será “procesado” para obtener un producto acorde con la capa en la que se encuentra.

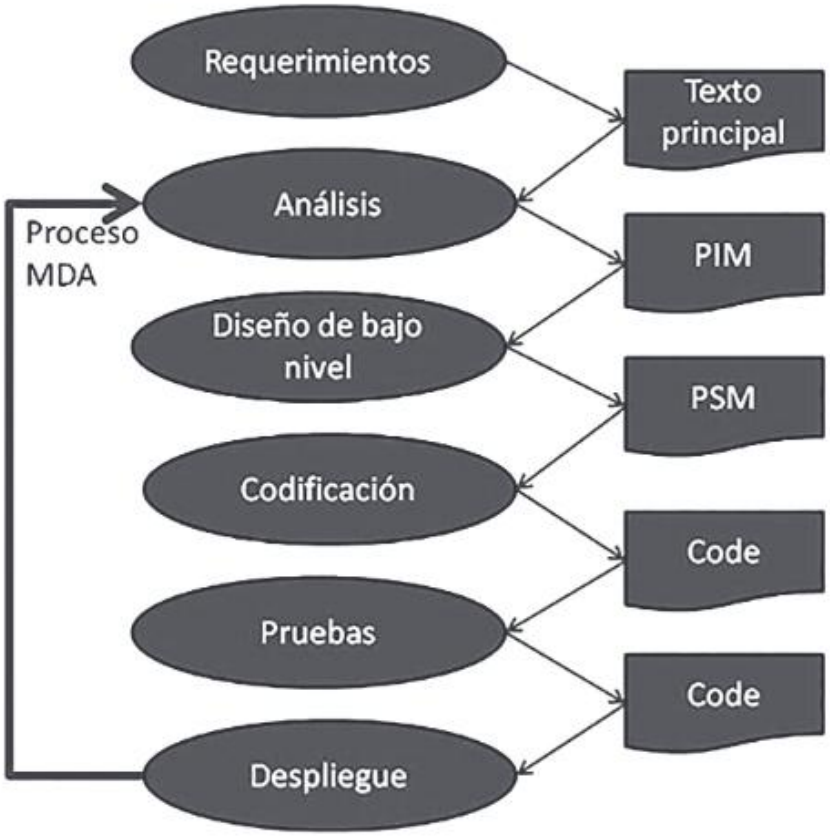

Figura 1Ciclo de vida del Desarrollo dirigido por modelos.

2.2.6 Desarrollo dirigido por modelos.

Paradigma emergente que resuelve numerosos problemas asociados con la composición e integración de sistemas a gran escala basado en el uso de modelos, soportado por potentes herramientas que tienen como objetivo reducir el tiempo de desarrollo y mejorar la calidad de los productos, separando el diseño de la arquitectura. Con el objetivo principal de permitir aumentar la productividad y reducir los costos del desarrollo (Bran , 2008).

Beneficios de MDD.

El desarrollo de software dirigido por modelos permite mejorar las prácticas corrientes de desarrollo de software. Las ventajas de MDD son las siguientes (Pons, Giandini, & Perez, 2010):

Incremento en la productividad. MDD reduce los costos de desarrollo de software mediante la generación automática del código y otros artefactos a partir de los modelos, lo cual incrementa la productividad de los desarrolladores. Note que debería sumar el costo de desarrollar (o comprar) transformaciones, pero es esperable que este costo se amortice mediante el re-uso de dichas transformaciones (Pons, Giandini, & Perez, 2010).

Adaptación a los cambios tecnológicos: el progreso de la tecnología hace que los componentes de software se vuelvan obsoletos rápidamente. MDD ayuda a solucionar este problema a través de una arquitectura fácil de mantener donde los cambios se implementan rápida y consistentemente, habilitando una migración eficiente de los componentes hacia las nuevas tecnologías. Los modelos de alto nivel están libres de detalles de la implementación, lo cual facilita la adaptación a los cambios que pueda sufrir la plataforma tecnológica subyacente o la arquitectura de implementación. Dichos cambios se realizan modificando la transformación del PIM (Modelos Independientes de Plataforma o Platform independent models) al PSM (Modelos Sujetos a Plataforma o Models subject to platform) .

La nueva transformación es reaplicada sobre los modelos originales para producir artefactos de implementación actualizados. Esta flexibilidad permite probar diferentes ideas antes de tomar una decisión final. Y además permite que una mala decisión pueda fácilmente ser enmendada (Pons, Giandini, & Perez, 2010).

Adaptación a los cambios en los requisitos: poder adaptarse a los cambios es un requerimiento clave para los negocios, y los sistemas informáticos deben ser capaces de soportarlos. Cuando se hace uso de un proceso MDD (Desarrollo dirigido por modelos o Model Driven development), agregar o modificar una funcionalidad de negocios es una tarea bastante sencilla, ya que el trabajo de automatización ya está hecho. Cuando se agrega una nueva función, sólo se necesita desarrollar el modelo específico para esa nueva función. El resto de la información necesaria para generar los artefactos de implementación ya ha sido capturada en las transformaciones y puede ser re-usada (Pons, Giandini, & Perez, 2010).

Consistencia: la aplicación manual de las prácticas de codificación y diseño es una tarea propensa a errores. A través de la automatización MDD favorece la generación consistente de los artefactos (Pons, Giandini, & Perez, 2010).

Re-uso: en MDD se invierte en el desarrollo de modelos y transformaciones. Esta inversión se va amortizando a medida que los modelos y las transformaciones son re-usados. Por otra parte el re-uso de artefactos ya probados incrementa la confianza en el desarrollo de nuevas funcionalidades y reduce los riesgos ya que los temas técnicos han sido previamente resueltos (Pons, Giandini, & Perez, 2010).

Mejoras en la comunicación con los usuarios: los modelos omiten detalles de implementación que no son relevantes para entender el comportamiento lógico del sistema. Por ello, los modelos están más cerca del dominio del problema, reduciendo la brecha semántica entre los conceptos que son entendidos por los usuarios y el lenguaje en el cual se expresa la solución. Esta mejora en la comunicación influye favorablemente en la producción de software mejor alineado con los objetivos de sus usuarios (Pons, Giandini, & Perez, 2010).

Mejoras en la comunicación entre los desarrolladores: los modelos facilitan el entendimiento del sistema por parte de los distintos desarrolladores. Esto da origen a discusiones más productivas y permite mejorar los diseños. Además, el hecho de que los modelos son parte del sistema y no sólo documentación, hace que los modelos siempre permanezcan actualizados y confiables (Pons, Giandini, & Perez, 2010).

2.2.7 Metodologías Relacionadas con Desarrollo dirigido por modelos (Model Driven Development)

Existen metodologías basadas en el desarrollo dirigido por modelos, a continuación se mencionara algunas:

2.2.7.1 Arquitectura Dirigida por Modelos (MDA)

Es una de las iniciativas más conocida y extendida dentro del ámbito de MDD. MDA es un concepto promovido por el OMG *(*acrónimo inglés de *Object Management Group)* a partir de 2000, con el objetivo de afrontar los desafíos de integración de las aplicaciones y los continuos cambios tecnológicos. MDA es una arquitectura que proporciona un conjunto de

guías para estructurar especificaciones expresadas como modelos, siguiendo el proceso MDD (Durán Muñoz, Troya Castilla, & Vallecillo Moreno).

En MDA, la funcionalidad del sistema es definida en primer lugar como un modelo independiente de la plataforma (Platform-Independent Model o PIM) a través de un lenguaje específico para el dominio del que se trate. En este punto aparece además un tipo de modelo existente en MDA, no mencionado por MDD: el modelo de definición de la plataforma (Platform Definition Model o PDM). Entonces, dado un PDM correspondiente a CORBA, .NET, Web, etc., el modelo PIM puede traducirse a uno o más modelos específicos de la plataforma (Platform-Specific Models o PSMs) para la implementación correspondiente, usando diferentes lenguajes específicos del dominio, o lenguajes de propósito general como Java, C#, Python, etc. El proceso MDA completo se encuentra detallado en un documento que actualiza y mantiene el OMG denominado la Guía MDA [MDAG] (Pons, Giandini, & Perez, 2010).

2.2.7.2 Ingeniería dirigida por modelos (MDA)

La ingeniería dirigida por modelos (MDA) surge como la respuesta de la ingeniería de software a la industrialización del desarrollo de software. MDA es la propuesta del Object Management Group (OMG), que centra sus esfuerzos en reconocer que la interoperabilidad es fundamental y que el desarrollo de modelos permite la creación de otros más que luego, al ser unidos, proveerían la solución a todo un sistema e independizarían el desarrollo de las tecnologías empleadas de modelos permite la creación de otros más que luego, al ser unidos, proveerían la solución a todo un sistema e independizarían el desarrollo de las tecnologías empleadas (Pascuas Rengifo, Mendoza Suarez, Eliana Dirley Córdoba Correa, & Córdoba Correa, 2015).

Se ha visto cómo el nivel de abstracción de los diferentes lenguajes de programación se ha incrementado durante décadas. Esto se puede considerar una evolución en los lenguajes de programación; así, antes se hablaba de lenguajes en binario; luego, del lenguaje ensamblador; más adelante, de los lenguajes procedimentales; posteriormente, de la orientación a objetos,

apoyada en diversos artefactos, como lenguaje unificado de modelado (UML); y ahora se habla de construcción de esos modelos como UML, a lo cual se le ha llamado precisamente ingeniería dirigida por modelos (MDA). Y aunque el nivel de abstracción aumente, este se acerca más al dominio específico sobre el cual se trabajará. Así, cualquier persona podría llegar a realizar un modelo empleando los conocimientos de su contexto, y la complejidad se traslada a ver cómo ese modelo se convierte en una solución desplegable y funcional sobre cualquier tecnología específica (Montenegro Marín, Ingeniería dirigida por modelos (MDA) y casos prácticos, 2017).

2.2.7.3 Ingeniería Dirigida Por Modelos (Mde)

La ingeniería dirigida por modelos surge como la respuesta de la ingeniería de software a la industrialización del desarrollo de software, MDA es la propuesta de la OMG, que centra sus esfuerzos en reconocer, que la interoperabilidad es fundamental y el desarrollo de modelos permite la generación de otros modelos que luego al ser juntados proveerán la solución a todo un sistema e independiza el desarrollo de las tecnologías empleadas.

Una buena interpretación de lo que es un modelo, un metamodelo son esas herramientas que permite la creación de un modelo, que es una descripción de uno o varios elementos del dominio o mundo real y finalmente el metametamodelo describe a esos metamodelos planteados, generando un grado de abstracción supremamente alto en el cuál coinciden todos los modelos (Montenegro Marín, Gaona García, Cueva Lovelle, & Sanjuan Martínez, 2011).

2.2.7.4 Modelado Específico del Dominio (DSM y DSLs)

Por su parte, la iniciativa DSM (Domain-Specific Modeling) es principalmente conocida como la idea de crear modelos para un dominio, utilizando un lenguaje focalizado y especializado para dicho dominio. Estos lenguajes se denominan DSLs (por su nombre en inglés: Domain Specific Language) y permiten especificar la solución usando directamente conceptos del dominio del problema. Los productos finales son luego generados desde estas especificaciones de alto nivel. Esta automatización es posible si ambos, el lenguaje y los

generadores, se ajustan a los requisitos de un único dominio. Definimos como dominio a un área de interés para un esfuerzo de desarrollo en particular. En la práctica, cada solución DSM se enfoca en dominios pequeños porque el foco reductor habilita mejores posibilidades para su automatización y estos dominios pequeños son más fáciles de definir. Usualmente, las soluciones en DSM son usadas en relación a un producto particular, una línea de producción, un ambiente específico, o una plataforma. El desafío de los desarrolladores y empresas se centra en la definición de los lenguajes de modelado, la creación de los generadores de código y la implementación de *frameworks* específicos del dominio, elementos claves de una solución DSM. Estos elementos no se encuentran demasiado distantes de los elementos de modelado de MDD. Actualmente puede observarse que las discrepancias entre DSM y MDD han comenzado a disminuir. Se puede comparar el uso de modelos así como la construcción de la infraestructura respectiva en DSM y en MDD. En general, DSM usa los conceptos dominio, modelo, metamodelo, metametamodelo como MDD, sin mayores cambios y propone la automatización en el ciclo de vida del software. Los DSLs son usados para construir modelos. Estos lenguajes son frecuentemente -pero no necesariamente gráficos. Los DSLs no utilizan ningún estándar del OMG para su infraestructura, es decir no están basados en UML, los metamodelos no son instancias de MOF, a diferencia usan por ejemplo MDF, el framework de Metadatos para este propósito. Finalmente, existe una familia importante de herramientas para crear soluciones en DSM que ayudan en la labor de automatización. En tal sentido, surge el término "*Software Factories*", que ha sido acuñado por Microsoft. Su descripción en forma detallada puede encontrarse en el libro de Greenfields del mismo nombre. Una *Software Factory* es una línea de producción de software que configura herramientas de desarrollo extensibles tales como Visual Studio Team System con contenido empaquetado como DSLs, patrones y *frameworks*, basados en recetas para construir tipos específicos de aplicaciones. Visual Studio provee herramientas para definir los metamodelos así como su sintaxis concreta y editores. En el capítulo 4 se verán con más detalle ésta y otras propuestas de herramientas que asisten en la tarea de construcción de modelos y lenguajes de modelado (Pons, Giandini, & Perez, 2010).

2.2.8 MVC

En líneas generales, MVC es una propuesta de diseño de software utilizada para implementar sistemas donde se requiere el uso de interfaces de usuario. Surge de la necesidad de crear software más robusto con un ciclo de vida más adecuado, donde se potencie la facilidad de mantenimiento, reutilización del código y la separación de conceptos.

Su fundamento es la separación del código en tres capas diferentes, acotadas por su responsabilidad, en lo que se llaman Modelos, Vistas y Controladores, o lo que es lo mismo, Model, Views & Controllers, si lo prefieres en inglés. En este artículo se estudiará con detalle estos conceptos, así como las ventajas de ponerlos en marcha cuando desarrollamos.

MVC es un "invento" que ya tiene varias décadas y fue presentado incluso antes de la aparición de la Web. No obstante, en los últimos años ha ganado mucha fuerza y seguidores gracias a la aparición de numerosos frameworks de desarrollo web que utilizan el patrón MVC como modelo para la arquitectura de las aplicaciones web (Alvarez, 2014).

Modelos: Es la capa donde se trabaja con los datos, por tanto contendrá mecanismos para acceder a la información y también para actualizar su estado. Los datos se tendrán habitualmente en una base de datos, por lo que en los modelos se tendrán todas las funciones que accederán a las tablas y harán los correspondientes selects, updates, inserts, etc.

No obstante, cabe mencionar que cuando se trabaja con MVC lo habitual también es utilizar otras librerías como PDO o algún ORM como Doctrine, que permiten trabajar con abstracción de bases de datos y persistencia en objetos. Por ello, en vez de usar directamente sentencias SQL, que suelen depender del motor de base de datos con el que se esté trabajando, se utiliza un dialecto de acceso a datos basado en clases y objetos (Alvarez, 2014).

Vistas: Las vistas, como su nombre da a entender, contienen el código de nuestra aplicación que va a producir la visualización de las interfaces de usuario, o sea, el código que nos permitirá renderizar los estados de nuestra aplicación en HTML. En las vistas nada más se tendrá los códigos HTML y PHP que nos permite mostrar la salida.

En la vista generalmente trabajamos con los datos, sin embargo, no se realiza un acceso directo a éstos. Las vistas requerirán los datos a los modelos y ellas se generarán la salida, tal como nuestra aplicación requiera (Alvarez, 2014).

Controladores: Contiene el código necesario para responder a las acciones que se solicitan en la aplicación, como visualizar un elemento, realizar una compra, una búsqueda de información, etc.

En realidad es una capa que sirve de enlace entre las vistas y los modelos, respondiendo a los mecanismos que puedan requerirse para implementar las necesidades de nuestra aplicación. Sin embargo, su responsabilidad no es manipular directamente datos, ni mostrar ningún tipo de salida, sino servir de enlace entre los modelos y las vistas para implementar las diversas necesidades del desarrollo (Alvarez, 2014).

2.2.8.1 Arquitectura de aplicaciones MVC

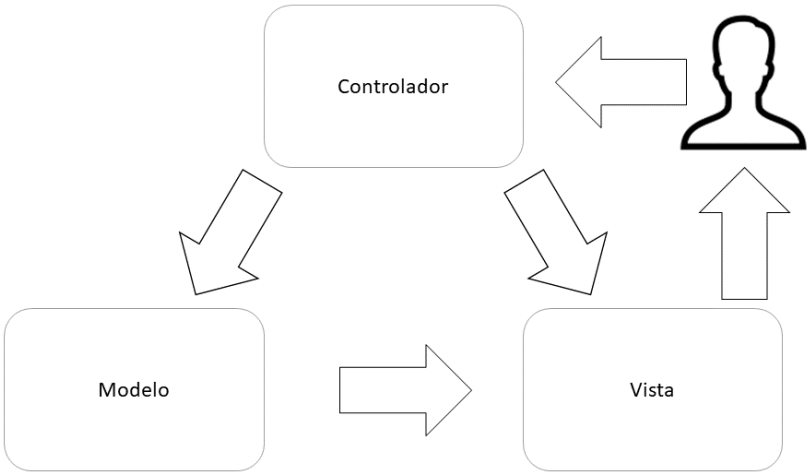

Figura 2 Arquitectura Modelo-Vista-Controlador (Alvarez, 2014).

En esta imagen se ha representado con flechas los modos de colaboración entre los distintos elementos que formarían una aplicación MVC, junto con el usuario. Como se puede ver, los controladores, con su lógica de negocio, hacen de puente entre los modelos y las vistas. Pero además en algunos casos los modelos pueden enviar datos a las vistas. Veamos paso a paso cómo sería el flujo de trabajo característico en un esquema MVC (Alvarez, 2014).

- El usuario **realiza una solicitud** a nuestro sitio web. Generalmente estará desencadenada por acceder a una página de nuestro sitio. Esa solicitud le llega al controlador.

- El **controlador comunica tanto con modelos como con vistas.** A los modelos les solicita datos o les manda realizar actualizaciones de los datos. A las vistas les solicita la salida correspondiente, una vez se hayan realizado las operaciones pertinentes según la lógica del negocio.
- Para producir la salida, en ocasiones las **vistas pueden solicitar más información a los modelos.** En ocasiones, el controlador será el responsable de solicitar todos los datos a los modelos y de enviarlos a las vistas, haciendo de puente entre unos y otros. Sería corriente tanto una cosa como la otra, todo depende de nuestra implementación; por eso esa flecha se ha coloreado de otro color.
- **Las vistas envían al usuario la salida.** Aunque en ocasiones esa salida puede ir de vuelta al controlador y sería éste el que hace el envío al cliente, por eso he puesto la flecha en otro color.

2.2.8.2 Framework MVC para el desarrollo web

Existen diversos de tipos frameworks MVC e igualmente la diversificación de estos según el lenguaje en el que estén basados y las funciones a cumplir. A continuación se mostraran algunos de los más notables frameworks MVC:

2.2.8.2.1 Laravel

Es uno de los frameworks de código abierto más fáciles de asimilar para PHP. Es simple, muy potente y tiene una interfaz elegante y divertida de usar. Fue creado en 2011 y tiene una gran influencia de frameworks como Ruby on Rails, Sinatra y ASP.NET MVC.

El objetivo de Laravel es el de ser un framework que permita el uso de una sintaxis refinada y expresiva para crear código de forma sencilla, evitando el "código espagueti" y permitiendo multitud de funcionalidades. Aprovecha todo lo bueno de otros frameworks y utiliza las características de las últimas versiones de PHP (Baquero García, 2015).

Características Generales

- Sistema de ruteo, también RESTful.
- Blade, Motor de plantillas.
- Peticiones Fluent.
- Eloquent ORM.
- Basado en Composer.
- Soporte para el caché.
- Soporte para MVC.
- Usa componentes de Symfony.
- Adopta las especificaciones PSR-2 y PSR-4.

La mayor parte de su estructura está formada por dependencias, especialmente de Symfony, lo que implica que el desarrollo de Laravel dependa también del desarrollo de sus dependencias (Baquero García, 2015).

Las Vistas: Laravel incluye de paquete un sistema de procesamiento de plantillas llamado Blade. Este sistema de plantillas, Blade favorece un código mucho más limpio en las Vistas, además de incluir un sistema de Caché que lo hace mucho más rápido (Desarrollando Webs Dinámicas, 2013).

2.2.8.2.2 AngularJS

AngularJS es un framework JavaScript de desarrollo de aplicaciones web en el lado cliente, viene de la mano de los chicos de Google y se podría decir que utiliza el patrón MVC (Model-View-Controller), aunque ellos mismos lo definen más bien como un MVW (Model-View-Whatever (whatever works for you)). Los creadores de este framework están convencidos de que HTML no está aún preparado para servir vistas dinámicas de un modo eficiente, así que han decidido extender la sintaxis de HTML para darle más funcionalidad (Lázaro, 2013).

Scopes: Los scopes son los distintos contextos de ejecución sobre los que trabajan las expresiones de AngularJS, por ejemplo, cuando se referencia un atributo del modelo mediante la directive ng-model, no estamos sino apuntando a un atributo que contiene el scope sobre el que se está trabajando. En los scopes se guarda la información de los modelos que se representan en la vista y también atributos que se utilizan para manejar la lógica de la misma (Lázaro, 2013).

Controllers: Los controllers son los encargados de inicializar y modificar la información que contienen los scopes en función de las necesidades de la aplicación (Lázaro, 2013).

Qué ofrece AngularJS

Client-side template: El sistema de plantillas en AngularJS es diferente del utilizado en otros frameworks. Por lo general es el servidor el encargado de mezclar la plantilla con los datos y devolver el resultado al navegador. En AngularJS el servidor proporciona los contenidos estáticos (plantillas) y la información que se va a representar (modelo) y es el cliente el encargado de mezclar la información del modelo con la plantilla para generar la vista (Lázaro, 2013).

Data binding: Con AngularJS se puede sincronizar el modelo y la vista automáticamente utilizando ciertas directives (ng-model en el ejemplo) del framework. Esta sincronización es bidireccional, es decir, la información se sincroniza tanto si cambia el valor en la vista como si lo hace el valor en el modelo (Lázaro, 2013).

Directives: Las directives son el plato fuerte de AngularJS. Mediante el uso de las mismas podemos extender la sintaxis de HTML y darle el comportamiento que se desee. Se puede crear directives a nivel de elemento, de atributo, de clase y de comentario. Un ejemplo sería el siguiente, mediante nuestra directive focusable (una directive a nivel de atributo) podemos modificar el comportamiento de los elementos input. En este caso cada vez que el input obtiene o pierde el foco cambia su color de fondo (Lázaro, 2013).

Filters: Los **filters** nos permiten modificar el modo en el que se va a presentar la información al usuario. La utilización de los mismos es similar a los Pipeline de Unix:

{{ expresión | filtro }}

Donde *expresión* puede ser cualquier tipo de expresión de AngularJS, como una variable del *$scope*, y *filtro* el nombre del filtro que le queremos aplicar a la expresión (Lázaro, 2013).

2.2.8.2.3 Struts

Struts es un framework para construir aplicaciones web Java basadas en la filosofía MVC. Veremos en este módulo en qué consiste la arquitectura MVC y cómo la implementa Struts.

MVC y Struts: Veamos cómo implementa Struts los componentes del patrón Modelo-Vista-**Controlador:** El controlador es un servlet, de una clase proporcionada por Struts. Será necesario configurar la aplicación web (a través del fichero web.xml) para que todas las peticiones del usuario se redirijan a este servlet. El controlador despacha las peticiones del usuario a la clase adecuada para ejecutar la acción. En struts, las clases que ejecuten las acciones deben heredar de la clase Action (Struts, 2014).

La vista: se implementará normalmente mediante páginas JSP. Struts ofrece dos herramientas para ayudar en la presentación de datos: **los ActionForms** son clases que capturan los datos introducidos en formularios y permiten su validación. **Las librerías de etiquetas** permiten mostrar errores y facilitar el trabajo con formularios. La implementación del modelo corre enteramente a cargo del desarrollador, ya que es propio de la capa de negocio y no está dentro del ámbito de Struts (Struts, 2014).

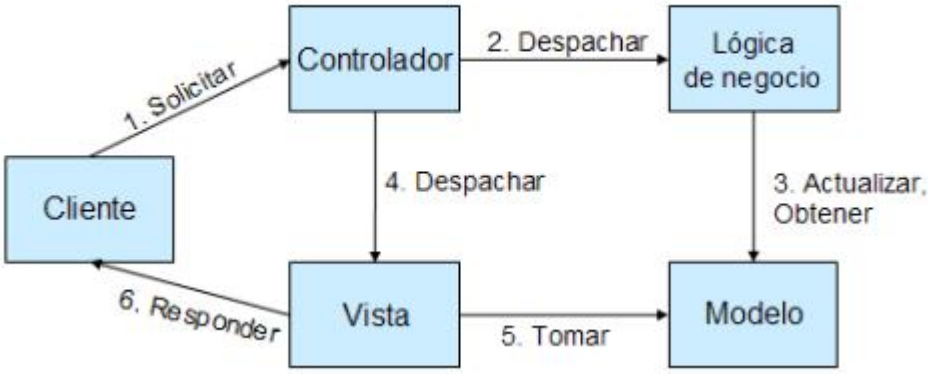

Figura 3 Modelo de ciclo de vida de Struts

2.2.8.3 Spring Framework

Spring es un framework alternativo al stack de tecnologías estándar en aplicaciones JavaEE. Nació en una época en la que las tecnologías estándar JavaEE y la visión "oficial" de lo que debía ser una aplicación Java Enterprise tenían todavía muchas aristas por pulir. Los servidores de aplicaciones eran monstruosos devoradores de recursos y los EJB eran pesados, inflexibles y era demasiado complejo trabajar con ellos. En ese contexto, Spring popularizó ideas como la inyección de dependencias o el uso de objetos convencionales (POJOs) como objetos de negocio, que suponían un soplo de aire fresco. Estas ideas permitían un desarrollo más sencillo y rápido y unas aplicaciones más ligeras. Eso posibilitó que de ser un framework inicialmente diseñado para la capa de negocio pasara a ser un completo stack de tecnologías para todas las capas de la aplicación (Johnson, y otros, 2004).

Desde un punto de vista genérico, Spring se puede ver como un soporte que nos proporciona tres elementos básicos:

Servicios Enterprise: Se puede hacer de manera sencilla que un objeto sea transaccional, o que su acceso esté restringido a ciertos roles, o que sea accesible de manera remota y

transparente para el desarrollador, o acceder a otros muchos servicios más, sin tener que escribir el código de manera manual. En la mayoría de los casos solo es necesario anotar el objeto (Johnson, y otros, 2004).

Estereotipos configurables: Para los objetos de nuestra aplicación: se puede anotar nuestras clases indicando por ejemplo que pertenecen a la capa de negocio o de acceso a datos. Se dice que son configurables porque se podría definir nuestros propios estereotipos "a medida": por ejemplo se podría definir un nuevo estereotipo que indicara un objeto de negocio que además sería cacheable automáticamente y con acceso restringido a usuarios con determinado rol (Johnson, y otros, 2004).

Inyección de dependencias: Ya se ha visto este concepto cuando se hablaba de CDI de JavaEE. La inyección de dependencias permite solucionar de forma sencilla y elegante cómo proporcionar a un objeto cliente acceso a un objeto que da un servicio que este necesita. Por ejemplo, que un objeto de la capa de presentación se pueda comunicar con uno de negocio. En Spring las dependencias se pueden definir con anotaciones o con XML (Johnson, y otros, 2004).

2.2.8.4 Spring Roo

Spring Roo es una herramienta de desarrollo rápido de aplicaciones empresariales en el lenguaje de programación Java. Le permite construir alta calidad, aplicaciones empresariales de bloqueo de alto rendimiento en cuestión de minutos. Lo mejor de todo es que Roo trabaja junto con sus conocimientos, habilidades y experiencia de Java existentes. Probablemente no necesite aprender nada nuevo para usar Roo, ya que no hay un nuevo idioma o plataforma de tiempo de ejecución necesaria. Usted simplemente programa en su forma normal de Java y Roo simplemente funciona, sentado en el fondo cuidando las cosas que no quiere preocuparse (Johnson, y otros, 2004).

2.2.8.4.1 Aclaraciones

Roo no está un tiempo de ejecución: Roo no está involucrado con su proyecto cuando se ejecuta en producción. No encontraras cualquier JAR en su classpath de tiempo de ejecución o anotaciones Roo compiladas en sus clases. Esto es realmente una cosa maravillosa. Significa que no tiene que preocuparse por el bloqueo. Probablemente también significa que no necesitará obtener la aprobación para uso de Roo, también significa que no hay una forma técnica posible para que Roo disminuya su velocidad proyectarse en el tiempo de ejecución, desperdiciar memoria o hinchar sus artefactos de implementación con JAR. Estamos realmente orgulloso del hecho de que Roo impone sin concesiones de ingeniería, ya que era uno de nuestros diseños centrales objetivos (Johnson, y otros, 2004).

Roo no es un complemento IDE: No se requiere un "complemento Roo Eclipse" o un "complemento Roo IntelliJ". Roo funciona perfectamente bien en su propia ventana de comandos del sistema operativo. (Johnson, y otros, 2004).

2.3 Estado del Arte

Teniendo en cuenta que se desea solucionar el problema anteriormente descrito, se ha hecho necesaria la consulta de material bibliográfico en busca de una introducción a la temática, dentro de esos recursos bibliográficos tal como Desarrollo de software dirigido por modelos cuyo contenido fue desarrollado por los señores Francisco Durán Muñoz, Javier Troya Castilla y Antonio Vallecillo Moreno de la universidad de Cataluña, quienes haciendo uso del concepto básico de desarrollo dirigido por modelos y describen otro concepto que está más enfocado al desarrollo de software y es conocido como el desarrollo de software dirigido por modelos, que en pocas palabras es la abstracción de los conceptos básicos de desarrollo dirigido por modelos y son aplicados al desarrollo de software.

El estudio de los beneficios ofrecidos por el desarrollo dirigido por modelos en la aplicación de repositorios institucionales, que se llevó a cabo con la finalidad de diseñar un aplicativo que estandarice la estructura de los documentos generados y guardados en dichos repositorios, estudio realizado por Texier José, De Giusti Marisa, Oviedo Néstor, Villarreal, Gonzalo y Lira Ariel en la universidad e Táchira en Venezuela.

Lidia Fuentes y Pablo Sánchez realizan una propuesta de metodología de desarrollo orientado a objetos basado en el desarrollo dirigido por modelos, dándole una gran capacidad de adaptación y asimilación de los procesos relacionados directamente con el desarrollo , todo esto plasmado en su publicación llamada “Desarrollo de software con aspectos dirigido por modelos”.

En Colombia en junio de 2015 Yois Smith Pascuas Rengifo, Jefersson Andres Mendoza Suarez, Eliana Dirley Córdoba Correa de la facultad de Ingeniería de Sistemas de la Universidad de la Amazonia, describen el desarrollo dirigido por modelos aplicado en un contexto educativo, describiendo como sería el desarrollo de unos laboratorios virtuales y el uso de herramientas según la necesidad.

En marzo del presente año (2017) describe una metodología relacionada directamente con el MDD, detallando herramientas de descripción de modelos y trasformaciones de modelos y

la aplicación de cada uno de estos conceptos en la práctica, plasmados en el texto “Ingeniería dirigida por modelos (MDA) y casos prácticos” creado por Carlos Enrique Montenegro Marín para obtener el título de doctorado en la universidad distrital Francisco José de Caldas.

En nuestra institución un trabajo llamado conjunto de reglas como estrategia para la aplicación de MDD en el desarrollo de un aplicativo web para el proceso de recaudación de impuestos de la secretaria de tránsito y transporte del municipio de pamplona que plasma estrategias para que el desarrollo dirigido por modelos se pueda aplicar en un proceso especifico, identificando cada uno de los tipos de modelos a implementar según la necesidad y equivalencia.

3 Procedimiento de desarrollo dirigido por modelos con spring framework

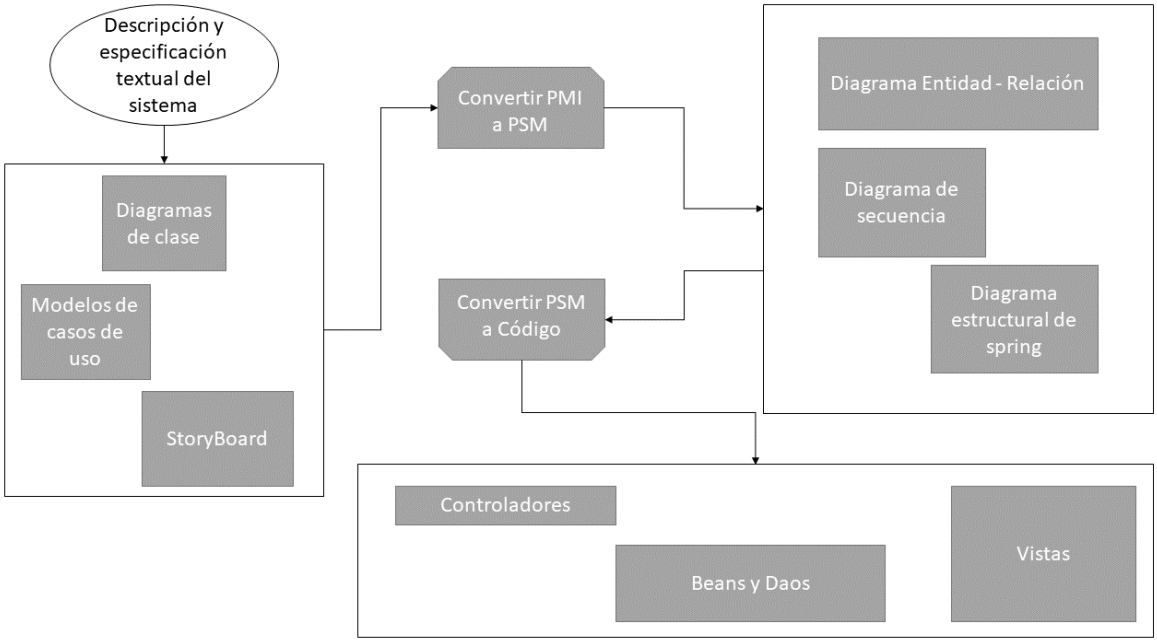

Figura 4 etapas de procedimiento de aplicación de desarrollo dirigido por modelos en Spring.

Para aplicar desarrollo dirigido por modelos se debe partir de un modelo que describa de forma eficiente las características del sistema a construir, partiendo de modelos que sean fáciles de leer e interpretar por el usuario, por lo general estos modelos no son dependientes de una plataforma específica. El lenguaje unificado de modelado (UML) proporciona una serie de modelos que en conjunto muestran como es el funcionamiento de un sistema, siendo UML un lenguaje que no está sujeto o es dependiente de una plataforma , los modelos construidos con este lenguaje serán modelos independientes de plataforma (PIM).

Una vez diseñados y definidos los modelos independientes de plataforma se procede a realizar una traducción e interpretación de estos modelos en una plataforma específica, esta

interpretación se hará de acuerdo a los parámetros definidos para el funcionamiento del framework Spring, construido en java.

3.1 Procedimiento

A continuación se establecerán los pasos para la implementación del desarrollo dirigido por modelos en una aplicación construida en spring framework.

3.1.1 Descripción y especificación del sistema

Describir de forma clara los alcances y limitaciones del sistema, para identificar las funcionalidades que componen el sistema.

3.1.2 Crear modelos independientes de plataforma

Teniendo en cuenta las funcionalidades identificadas en la descripción del sistema, para la creación de modelos independientes de plataforma se utilizaran los diagramas de clases, diagramas de caos de uso y los storyboards. Los diagramas de casos de uso para mostrar las interacciones de usuarios con el sistema, storyboards describen las respuestas a las acciones ejecutadas por el usuario en el sistema y los diagramas de clases los objetos que componen el sistemas, los atributos de los objetos y las tareas que se realizan sobre los objetos.

3.1.3 Convertir modelos independientes de plataforma en modelos dependientes

Los diagramas de clases pasan a ser modelos de entidad relación, esta conversión se da siguiendo las siguientes acotaciones:

- Cada clase del diagrama de clases corresponderá a una entidad o tabla del modelo.
- Cada atributo de una clase representa un campo o columna de una entidad del modelo.
- Los atributos de las clases son de un tipo específico de datos, un tipo de datos varying de un atributo correspondería a un campo de tipo varchar, un atributo de tipo integer de una clase corresponde a un campo de tipo integer o int, los atributos de tipos específicos como booleano serán campos del mismo tipo en el modelo entidad

relación y finalmente los atributos con tipo de datos date serán equivalentes a campos de tipo date o timestamp en una entidad.

- Por efectos de orden y normalización cada campo perteneciente a una entidad tomará el nombre que posee el atributo equivalente, por consecuencia el nombre que el campo tomara será el nombre del atributo acompañado por un sufijo, dicho sufijo será una abreviación del nombre de la entidad a la que pertenece o una abreviatura diciente y que sea fácil de relacionar con la entidad. El sufijo y el nombre de atributo serán unido por una (_) guion al piso. Ejemplo:

  Nombre de clase: persona

  Nombre atributo: identificación

  Nombre resultante del campo: pers_identificacion

- Dado el caso en el que el nombre de una clase o el nombre de un atributo está constituido por dos palabras, entonces el nombre de la entidad o nombre del campo tomará abreviaturas dicientes de las dos palabras y se unirán pasando la primera letra de la segunda palabra a mayúsculas. Si las palabras que conforman el nombre son cortas se pueden unir cambiando la primera letra a mayúsculas. Ejemplo:

  Nombre de clase: persona asistente

  Nombre atributo: primer apellido

  Nombre resultante entidad: personaAsis

  Nombre resultante campo: perAsis_primerApe

- Cada entidad tendrá un id o identificador secuencial, de tipo int.

- Las relaciones de dependencia de clases será interpretada como una relación entre tablas o llave foránea, haciendo alusión a una herencia de datos relacionados entre las tablas y relación de uno a uno.
- Una relación de composición denota una relación foránea donde se hace relación a contenidos o procesos creados por el usuario y con una relación de uno a muchos.
- Las primary key o llaves primarias serán los identificadores de cada entidad.

Los storyboards se convierten en diagramas de secuencia, para hacer está conversión se establecen unas capas de interacción presentes en los sistemas basados en web que son invisibles para el usuario y son las siguientes.

- Presentación
- Servicio
- Acceso a datos
- Fuente de datos

Teniendo en cuenta las capas se crea diagramas de interacción entre cada una de estas, cada capa proporciona a la siguiente datos. La capa de presentación ofrece datos ingresados por el usuario, la capa de servicio aporta peticiones de acuerdo a los datos del usuario, la capa de acceso a datos objetos con datos y la fuente proporciona los datos.

Los diagramas de casos de uso se utilizan para mostrar las interacciones del usuario con el sistema y poder ser descrito en el diagrama de secuencia.

3.1.4 Crear modelos sujetos a plataforma

Siguiendo las recomendaciones para convertir modelos independientes de plataforma en modelos sujetos de plataforma se generan los diagramas entidad relación y los diagramas de secuencia. El diagrama entidad relación se utilizara para la generación del código SQL correspondiente a la base de datos y los diagramas de secuencia definen las interacciones entre las capas definidas dentro del sistema.

3.1.5 Convertir modelos sujetos a plataforma en código

Para convertir el modelo entidad relación se emplean los métodos tradicionales en la generación de Scripts SQL como el siguiente:

Convertir cada entidad en una tabla. Los atributos de cada entidad representan los campos de las tablas.

- Cada una de las entidades se convertirá en una tabla.
- Cada atributo de la entidad corresponde a un campo en la tabla con su correspondiente tipo de dato.
- establecer el identificador de la tabla, el identificador es un campo tipo entero secuencial o autoincrementable.
- Las relaciones identificadas como relaciones de muchos a muchos se generan como una tabla que contiene los identificadores de las tablas relacionadas.
- Cada campo se le agrega un sufijo relacionado con el nombre de la tabla concatenados con un (_), Ejemplo:

  Nombre de tabla: persona
  Nombre de campo: cedula
  Campo con sufijo: pers_cedula

- Si el nombre del atributo está compuesto por dos palabras utilizar las primeras 3 letras de cada palabra con la primera letra de la segunda palabra en mayúscula. Ejemplo:

  Nombre atributo: teléfono oficina
  Resultado: pers_telOfi

Para generar los códigos que comprenden y construyen el funcionamiento de la aplicación se hace uso de la herramienta Spring Roo con las siguientes instrucciones:

- Conectar con la base de datos y definir el acceso a datos:

Jpa setup --provider HIBERNATE --database servidor_baseDatos --databaseName baseDatosNom

- Establecer usuario y contraseña del servidor de base de datos en el archivo: meta-inf/spring/database.properties

- Crear entidades, existe una entidad por cada tabla en la base de datos
  entity jpa --class ~.rutaAdicionalPaquete.nombreEntidad –accion

- Crear campos en las entidades
  field tipoDato --fieldName nombreCampo --restrccion --sizeMax Tamaño

- Crear relaciones entre entidades
  field set --fieldName campoEntidad --type ~.nombreEntidad --cardinality (MANY_TO_MANY, MANY_TO_ONE, ONE_TO_MANY)
- Generar la capa web
  web mvc setup
- Generar toda la estructura de acceso a datos
  web mvc all --package ~.web
- Integrar capa de seguridad al proyecto
  security setup

3.1.6 Generar código

Se genera el código SQL de la base de datos siguiendo las indicaciones en el punto anterior, también se genera el código de la aplicación construida haciendo uso de las instrucciones establecidas con la herramienta Spring Roo.

3.2 Aplicación del procedimiento

Para efectos de demostración, la aplicación del procedimiento se ejecutará en una de las funcionalidades del sistema que se planteara, puesto que la aplicación del procedimiento se llevaría a cabo de manera muy similar en otras funcionalidades del sistema.

3.2.1 Descripción y especificación del sistema

Una cadena de hoteles quiere ofrecer vía web el servicio de reservas, en la construcción del sistema de reservas debe tener en consideración lo siguiente:

De los hoteles se necesita la información básica, como el nombre, dirección, teléfono y ciudad de ubicación. Los hoteles tiene diferentes categorías de habitaciones (suites, dobles, individuales, etc.). Así pues, de cada habitación se desea guardar el código y el tipo de habitación. Los particulares pueden realizar reservas de las habitaciones de los hoteles. En la reserva de los particulares figurarán el nombre, la dirección y el teléfono. A cada habitación está asociado un mini bar con productos que el cliente puede consumir generando cargo a su cuenta. Para la reserva se debe tener en cuenta la fecha de inicio y finalización de la reserva, la habitación que se reserva y el monto con el que se hizo la reserva.

3.2.2 Crear modelos independientes de plataforma.

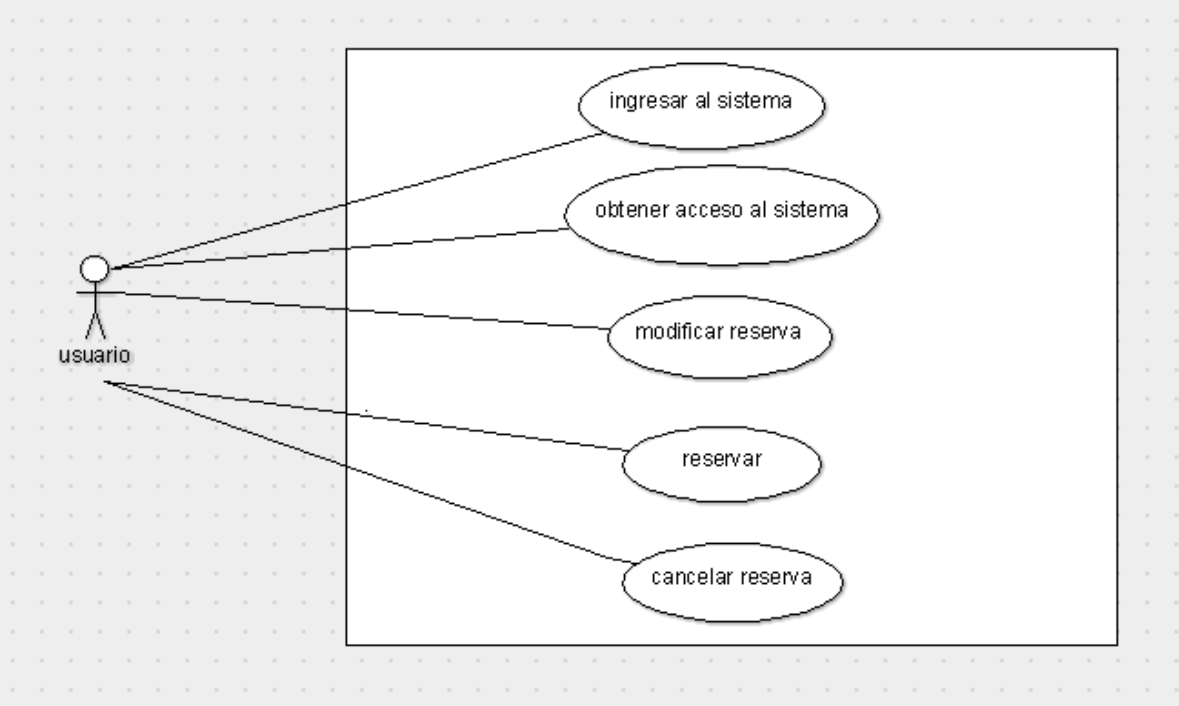

Figura 5Diagrama de casos de uso del sistema.

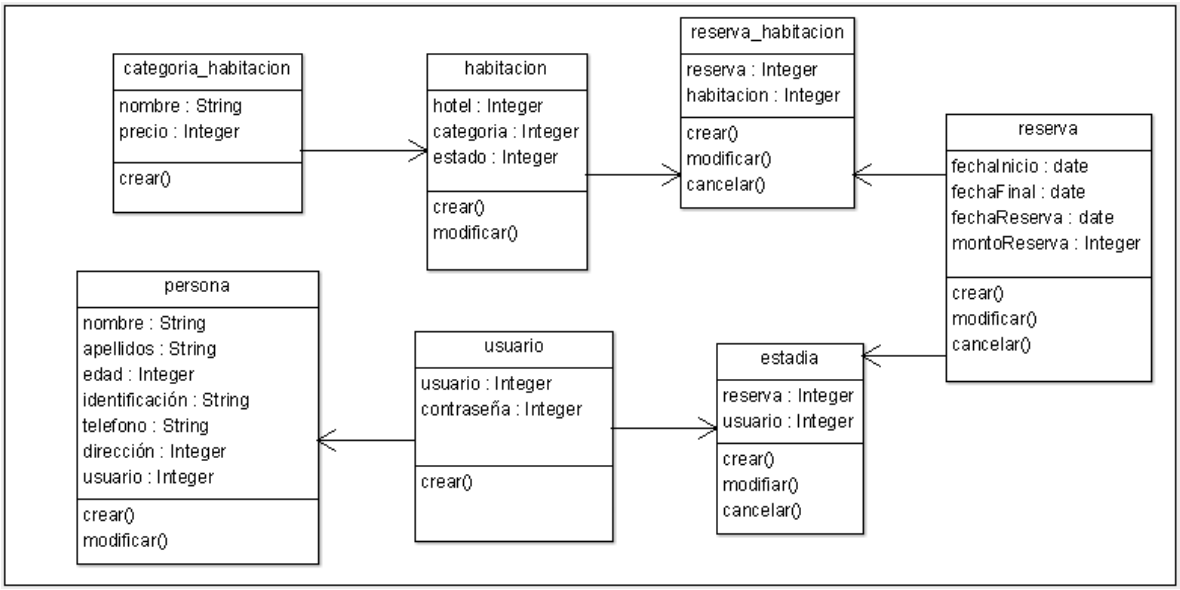

Figura 6 Diagrama de clases de la funcionalidad reserva.

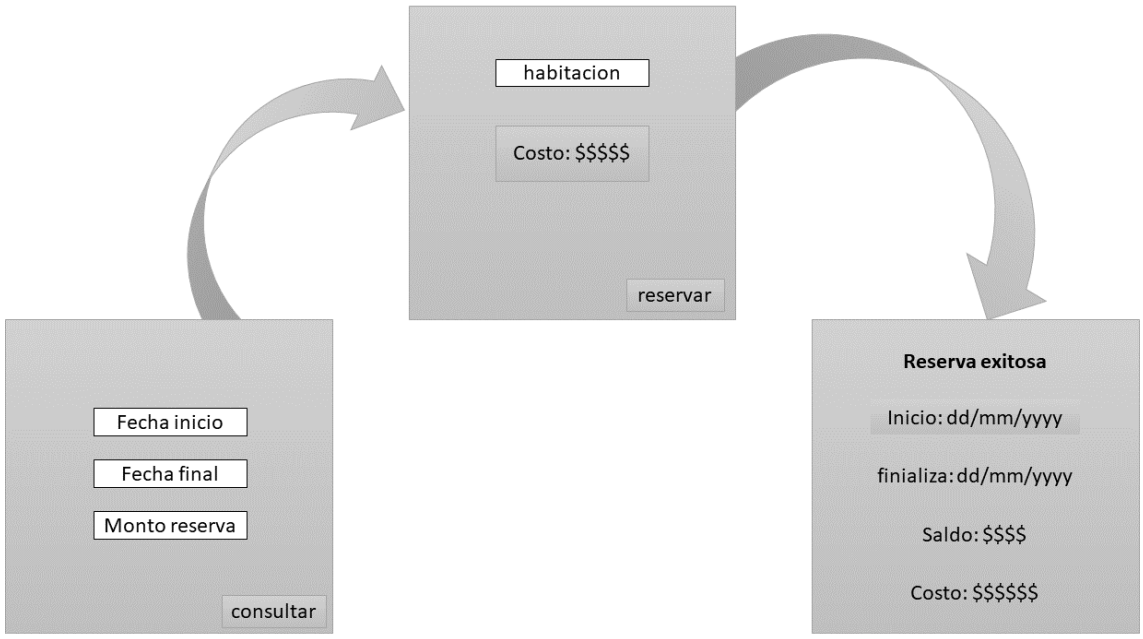

Figura 7 Storyboard funcionalidad reserva.

3.2.3 Convertir modelos independientes de plataforma en modelos dependientes

En esta etapa del procedimiento se toman los modelos independientes de plataforma y se empiezan a convertir en modelos sujetos a plataforma atendiendo a las acotaciones establecidas en el procedimiento.

Ejemplo:

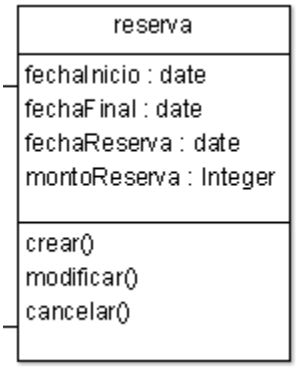

Figura 8 clase reserva.

Esta clase se convertirá en unos modelos sujetos a plataforma muy parecido al siguiente:

Figura 9Entidad Reserva.

3.2.4 Crear modelos sujetos a plataforma

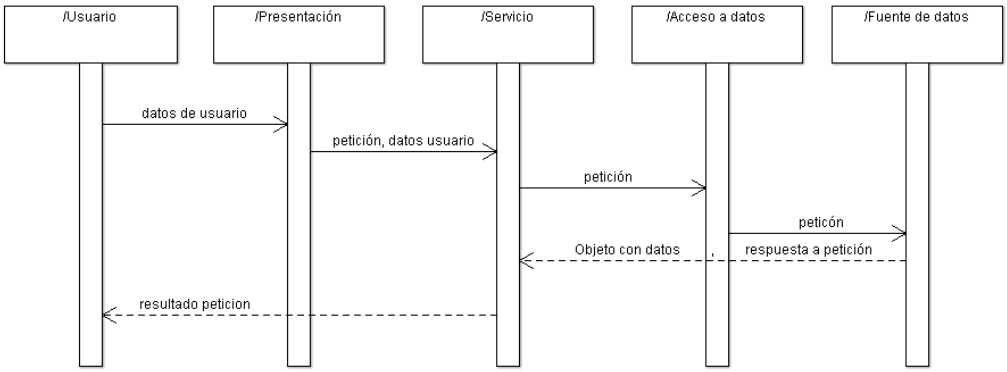

Figura 10 Diagrama de secuencia de la funcionalidad Reserva.

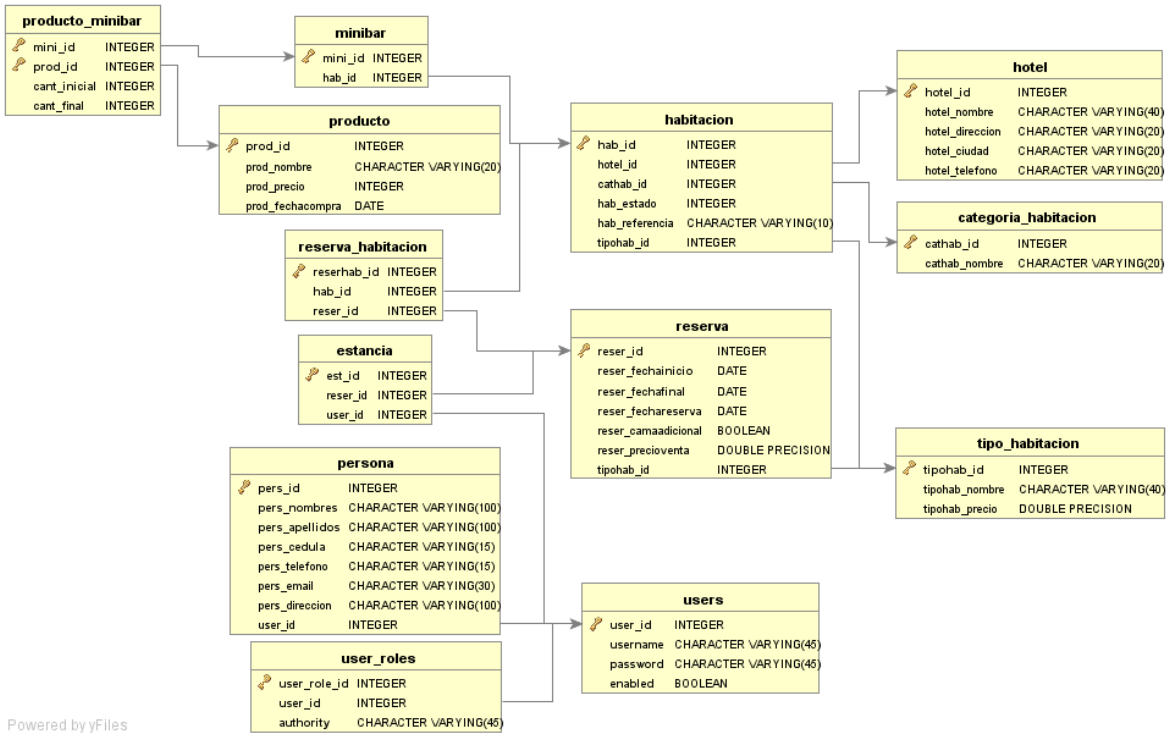

Figura 11 Diagrama Entidad relación del sistema.

3.2.5 Generación de código a partir de los modelos sujetos a plataforma

Una vez se tienen todos los modelos dependientes de plataforma o sujeto a plataforma es momento de convertir esos modelos a código, código que le dará forma a las funcionalidades del sistema. Vale la pena destacar que debido a que se está haciendo uso de la herramienta Roo, que ayuda a generar aplicaciones de forma rápida con un código "genérico", la generación de código se llevara a cabo de forma transparente al usuario haciendo uso de unas instrucciones específicas de Roo ya descritas anteriormente.

Conexión a la base de datos

jpa setup --provider HIBERNATE --database POSTGRES --databaseName spring01

Creación de entidades

entity jpa --class ~.entidades.categoria_habitacion

entity jpa --class ~.entidades.estancia

entity jpa --class ~.entidades.habitacion

entity jpa --class ~.entidades.hotel

Creación de campos en entidades

focus --class ~.entidades.habitacion

field set --fieldName hotel_id --type hotel --cardinality ONE_TO_MANY --mappedBy

field set --fieldName catHab_id --type categoria_habitacion --cardinality ONE_TO_MANY --mappedBy

3.2.6 Generar código

Generación de código SQL a partir del diagrama entidad relación, llevando a cabo las directrices de cambio entre modelos:

- Tabla categoría_habitación.

```
create table categoria_habitacion(
catHab_id serial,
catHab_nombre varchar(20),
primary key(catHab_id)
);
```

- Tabla Hotel

```
create table hotel(
hotel_id serial,
hotel_nombre varchar(40),
hotel_direccion varchar(20),
hotel_ciudad varchar(20),
hotel_telefono varchar(20),
primary key(hotel_id),
```

```
unique(hotel_nombre)
);
```

- Tabla habitación

```
create table habitacion(
hab_id serial,
hotel_id int,
catHab_id int,
hab_estado int,
hab_referencia varchar(10),
tipoHab_id int,
primary key(hab_id),
foreign key(catHab_id) references categoria_habitacion(catHab_id),
foreign key(tipoHab_id) references tipo_habitacion(tipoHab_id),
foreign key(hotel_id) references hotel(hotel_id)
);
```

- Tabal Reserva

```
create table reserva(
reser_id serial,
reser_fechaInicio date not null,
reser_fechaFinal date not null,
reser_fechaReserva date not null,
reser_camaAdicional boolean,
reser_precioVenta float,
tipoHab_id int,
foreign key(tipoHab_id) references tipo_habitacion(tipoHab_id),
primary key(reser_id)
);
```

- Tabla Reserva_habitación

```
create table reserva_habitacion(
reserHab_id serial,
hab_id int not null,
reser_id int not null,
foreign key(hab_id) references habitacion(hab_id),
foreign key(reser_id) references reserva(reser_id),
primary key(reserHab_id)
);
```

- Tabla users

```
create table users (
user_id serial not null,
username VARCHAR(45) NOT NULL,
password VARCHAR(45) NOT NULL,
enabled boolean NOT NULL,
PRIMARY KEY (user_id)
);
```

Generación de código del sistema, generado por instrucciones propias de spring Roo

```java
package hotel.clases.entidades;
import org.springframework.roo.addon.javabean.RooJavaBean;
import org.springframework.roo.addon.jpa.activerecord.RooJpaActiveRecord;
import org.springframework.roo.addon.tostring.RooToString;
import java.util.HashSet;
import java.util.Set;
import javax.persistence.CascadeType;
import javax.persistence.OneToMany;
import javax.validation.constraints.Size;
@RooJavaBean
@RooToString
@RooJpaActiveRecord
public class Habitacion {
    /**
     */
    @OneToMany(cascade = CascadeType.ALL)
    private Set<Hotel> hotel_id = new HashSet<Hotel>();
    /**
     */
    @OneToMany(cascade = CascadeType.ALL)
    private Set<Categoria_habitacion> catHab_id = new HashSet<Categoria_habitacion>();
    /**
     */
    private Double hab_estado;
    /**
     */
    @Size(max = 10)
    private String hab_referencia;
    /**
     */
    @OneToMany(cascade = CascadeType.ALL)
    private Set<Tipo_habitacion> tipohab_id = new HashSet<Tipo_habitacion>();
}
```

Figura 12 Entidad habitación de Spring

```
<div xmlns:spring="http://www.springframework.org/tags" xmlns:util="urn:jsptagdir:/WEB-INF/tags/util" xmlns:jsp="http://java.sun.
  <jsp:directive.page contentType="text/html;charset=UTF-8" />
  <jsp:output omit-xml-declaration="yes" />
  <spring:message var="app_name" code="application_name" htmlEscape="false" />
  <spring:message var="title" code="welcome_titlepane" arguments="${app_name}" htmlEscape="false" />
  <util:panel id="title" title="${title}">
    <h3>
      <spring:message code="welcome_h3" arguments="${app_name}" />
    </h3>
    <p>
      <spring:message code="welcome_text" />
    </p>
  </util:panel>
</div>
```

Figura 13 Estructura de la codificación de la vista principal o index.

```
~.web roo> focus --class ~.entidades.users
~.entidades.Users roo> field set --fieldName user_id --type persona --cardinality ONE_TO_MANY --mappedBy
Updated SRC_MAIN_JAVA\hotel\clases\entidades\Users.java
Created SRC_MAIN_WEBAPP\WEB-INF\views\userses
Created SRC_MAIN_WEBAPP\WEB-INF\views\userses\views.xml
Updated SRC_MAIN_WEBAPP\WEB-INF\views\userses\views.xml
Updated SRC_MAIN_WEBAPP\WEB-INF\i18n\application.properties
Updated SRC_MAIN_JAVA\hotel\clases\web\UsersController_Roo_Controller.aj
Updated SRC_MAIN_JAVA\hotel\clases\entidades\Users_Roo_Jpa_ActiveRecord.aj
Created SRC_MAIN_JAVA\hotel\clases\entidades\Users_Roo_JavaBean.aj
Created SRC_MAIN_WEBAPP\WEB-INF\views\userses\list.jspx
Created SRC_MAIN_WEBAPP\WEB-INF\views\userses\show.jspx
Created SRC_MAIN_WEBAPP\WEB-INF\views\userses\create.jspx
Updated SRC_MAIN_WEBAPP\WEB-INF\views\menu.jspx
Created SRC_MAIN_WEBAPP\WEB-INF\views\userses\update.jspx
~.entidades.Users roo> Updated SRC_MAIN_JAVA\hotel\clases\web\PersonaController_Roo_Controller.aj
Updated SRC_MAIN_WEBAPP\WEB-INF\views\personae\create.jspx
Updated SRC_MAIN_WEBAPP\WEB-INF\views\personae\update.jspx
Updated SRC_MAIN_JAVA\hotel\clases\entidades\Persona_Roo_Jpa_ActiveRecord.aj
Updated SRC_MAIN_JAVA\hotel\clases\entidades\Persona_Roo_JavaBean.aj
~.entidades.Users roo> focus --class ~.entidades.persona
~.entidades.Persona roo> field set --fieldName user_id --type users --cardinality MANY_TO_ONE --mappedBy
Cardinality must be ONE_TO_MANY or MANY_TO_MANY for the field set command
~.entidades.Persona roo> field set --fieldName user_id --type users --cardinality ONE_TO_MANY --mappedBy
Updated SRC_MAIN_JAVA\hotel\clases\entidades\Persona.java
Updated SRC_MAIN_JAVA\hotel\clases\web\PersonaController_Roo_Controller.aj
Updated SRC_MAIN_JAVA\hotel\clases\entidades\Persona_Roo_Jpa_ActiveRecord.aj
Updated SRC_MAIN_JAVA\hotel\clases\entidades\Persona_Roo_JavaBean.aj
Updated SRC_MAIN_WEBAPP\WEB-INF\views\personae\create.jspx
Updated SRC_MAIN_WEBAPP\WEB-INF\views\personae\update.jspx
~.entidades.Persona roo>
```

Figura 14 ejecución de instrucciones de Spring roo.

En la aplicación del procedimiento se deben tener ciertas consideraciones debido a que la implementación de roo como herramienta de desarrollo y generación de código, implica la implementación de Apis y módulos propios de spring y spring Roo. El módulo Security de Roo permite implementar una capa de seguridad web como es el login, este login está configurado con anotaciones en un archivo XML llamado applicationContext-seurity.xml, en este archivo se encuentran definidos los usuarios, roles de usuario y los permisos de acceso de esos usuarios. La forma de leer o identificar los usuarios puede secar cambiada, haciendo que estos sean verificados desde los datos existentes en una base de datos, la tabla que contiene los datos de usuarios debe tener unos campos específicos (user_id, username, password y enabled) para poder realizar la lectura y validación de datos. Los roles de usuario también pueden ser obtenidos desde una tabla en una base de datos, la tabla roles de usuario también debe tener unos campos específicos (user_id y authority) y una llave foránea que la relaciona con la tabla usuario.

En cuanto el uso de Hibernate y Jpa, estos son de mucha ayuda, puesto que la aplicación de estos por parte de Roo genera aplicaciones funcionales, con sus correspondientes vistas dejando un aplicativo funcional, Hibernate crear métodos estándar que en ocasiones no son suficientes, haciendo necesaria la personalización del código y por ende el estudio de Hibernate. La personalización de código puede llevar a tener que escribir por completo los métodos de los controladores, perdiendo los códigos implementados por spring Roo.

4 Conclusiones, recomendaciones y trabajos futuros

4.1 Conclusiones

- Realizando recolección de información relacionada con el desarrollo de software dirigido por modelos encontrada en documentos cómo tesis de pregrado, artículos, libros y publicaciones, se generó un procedimiento para la aplicación de desarrollo dirigido por modelos ajustado para la plataforma Spring.
- En el proceso de creación del procedimiento, se estableció un conjunto de pasos, este conjunto de pasos permite llevar a cabo el desarrollo de dicho procedimiento definiendo las tareas a realizar en cada uno de estos pasos.
- Como resultado de la aplicación del procedimiento creado se obtiene un sistema que no cumple al 100% con las especificaciones establecidas y funcionalidades.
- Teniendo en cuenta los resultados se puede decir que la implementación de spring roo como herramienta de desarrollo no es suficiente para realizar desarrollo dirigido por modelos, aun cuando ofrecen facilidades debido a la integración con otras Apis dentro del framework.

4.2 Recomendaciones

- Teniendo en cuenta los resultados obtenidos haciendo la aplicación del procedimiento a un sistema considerablemente pequeño y haciendo uso de Spring roo como herramienta de desarrollo, se encuentra limitaciones en cuanto a la posibilidad de generar un aplicativo funcional que sea totalmente acorde a lo modelado, teniendo que recurrir a personalizaciones de código bastante extensas en relación al alcance de las funcionalidades. La implementación de Roo como herramienta de desarrollo brilla en funcionalidades en donde hace presencia mayormente la creación, lectura, edición y borrado de un contenido (CRUD). El

estudio de las Apis integradas dentro de Spring tales como Hibernate y Jpa ayudará en gran medida al desarrollo de aplicaciones debido al alto uso de estas dos Apis en el módulo Roo.

4.3 Trabajos futuros

En la aplicación del procedimiento que se creó se presentaron inconvenientes con una de las funcionalidades, específicamente en la funcionalidad de registro de usuario, debido a que la integración de "security" implica la personalización de código, creando la necesidad de generar anotaciones para configurar los permisos de los tipos de roles y la creación de usuarios. Las anotaciones se tomaron como tarea de desarrollo de Roo, por ende el estudio e implementación de anotaciones es de vital importancia, ayudando a personalizar el funcionamiento de una aplicación.

Spring roo trabaja con librerías incluidas en el paquete de spring, entre estas destacan el uso de JPA e Hibernate. Estas librerías ayudan a la construcción de toda la capa de modelo en un mvc en conjunto con spring roo que ejecuta procedimientos de configuración de estas librerías en el proyecto, haciendo necesario conocer la integración de estas librerías con spring para la personalización de funcionalidades.

5 Bibliografía

Montenegro Marín, C. E. (marzo de 2017). *Ingeniería dirigida por modelos (MDA) y casos prácticos.* Bogotá: Hographics Impresores.

Alvarez, M. (02 de enero de 2014). *desarrolloweb.com*. Obtenido de https://desarrolloweb.com/articulos/que-es-mvc.html

Baquero García, J. (11 de 12 de 2015). *arsys*. Obtenido de https://www.arsys.es/blog/programacion/que-es-laravel/

Bran , S. (2008). The Pragmatics of Model-driven Development. *IEEE Computer Society*.

Cabot, J. (7 de Abril de 2011). Obtenido de https://es.slideshare.net: https://es.slideshare.net/jcabot/mdd-desarrollo-de-software-dirigido-por-modelos-que-funciona-de-verdad

Cueva Lovelle, J. M., & García-Bustelo, C. P. (Noviembre de 2008). MDE: Ingeniería dirigida por modelos. Bogotá, Cundinamarca, Colombia.

Desarrollando Webs Dinámicas. (21 de 03 de 2013). Obtenido de http://desarrollandowebsdinamicas.blogspot.com.co/2013/03/que-es-laravel.html

Durán Muñoz, F., Troya Castilla, J., & Vallecillo Moreno, A. (s.f.). *Desarrollo de Software dirigido por modelos.* Catalunya: Universidad abierta de Catalunya.

Fuentes , L., & Sánchez, P. (s.f.). Desarrollo de software con aspectos dirigido por modelos.

Johnson, R., Hoeller, J., Arendsen, A., Sampaleanu, C., Harrop, R., Risberg, T., . . . Poutsma, A. (2004). Spring java/j2ee Application framework.

Lázaro, P. (20 de mayo de 2013). *Pablo Lázaro*. Obtenido de http://pablolazarodev.blogspot.com.co/2013/05/que-es-angularjs-una-breve-introduccion.html

Lira, A., Villarreal, G., Oviedo, N., De Giusti, M., & Texier, J. (s.f.). Los Beneficios del Desarrollo Dirigido por Modelos. Táchira, Venezuela.

Martín Vera , P. (14 de Septiembre de 2015). Desarrollo Dirigido por Modelos Basado en Componentes de Interfaz de Usuario. La Plata, Argentina.

Montenegro Marín, C. E., Gaona García, P. A., Cueva Lovelle, J. M., & Sanjuan Martínez, O. (25 de Julio de 2011). *Aplicación De Ingeniería Dirigida Por Modelos (Mda), Para La Construcción De Una Herramienta De Modelado De Dominio Específico (Dsm) Y La Creación De Módulos En Sistemas De Gestión De Aprendizaje (Lms) Independientes De La Plataforma.* Medellin.

Pascuas Rengifo, Y. S., Mendoza Suarez, J. A., Eliana Dirley Córdoba Correa, & Córdoba Correa, E. D. (2015). Desarrollo Dirigido por Modelos (MDD) en el Contexto Educativo. *Scientia et technica*.

Pons, C., Giandini, R., & Perez, G. (2010). *Desarollo de Software dirigido por odelos, Conceptos teoricos y su aplicacion practica.* La Plata - Argentina: Editorial de la universidad de la Plata.

Rodriguez López, C. A. (2013). *conjunto de reglas como estrategia para la aplicación de MDD en el desarrollo de un aplicativo web para el proceso de recaudación de impuestos de la secretaria de tránsito y transporte del municipio de pamplona.* Pamplona.

Schmidt, D. (2006). Model-driven engineering technologies offer a promising approach to address the inability of third-generation. *IEEE computer Society,Vanderbilt University*.

Stahl, T., & Völter, M. (2006). *Model-Driven Software Technology, Engineering, Management.* England: John Wiley & sons, Ltd.

Struts. (26 de 06 de 2014). Aalicante, España .