PAI161-1

ZOE-GEN: Un transformador para facilitar la generación de aplicaciones basado en modelos.

John Carlos Olarte Abello

PONTIFICIA UNIVERSIDAD JAVERIANA
FACULTAD DE INGENIERIA
MAESTRÍA EN INGENIERÍA DE SISTEMAS Y COMPUTACIÓN
BOGOTÁ, D.C.
2016
PAI161-1

ZOE-GEN: Un transformador para facilitar la generación de aplicaciones basado en modelos.

Autor:
John Carlos Olarte Abello

MEMORIA DEL TRABAJO DE GRADO REALIZADO PARA CUMPLIR UNO DE LOS REQUISITOS PARA OPTAR AL TITULO DE
Magíster en Ingeniería de Sistemas y Computación

Directora
Jaime Andrés Pavlich Mariscal PhD
Comité de Evaluación del Trabajo de Grado
Maria Consuelo Franky de Toro
Maria Catalia Acero
Página web del Trabajo de Grado
http://pegasus.javeriana.edu.co/~PA161-1-ZOE-GEN

PONTIFICIA UNIVERSIDAD JAVERIANA
FACULTAD DE INGENIERIA
MAESTRÍA EN INGENIERIA DE SISTEMAS Y COMPUTACIÓN
BOGOTÁ, D.C.
Junio, 2016

PONTIFICIA UNIVERSIDAD JAVERIANA

FACULTAD DE INGENIERIA

MAESTRÍA EN INGENIERÍA DE SISTEMAS Y COMPUTACIÓN

Rector Magnífico

Jorge Humberto Pelaez, S.J.

Decano Facultad de Ingeniería

Ingeniero Jorge Luis Sanchez

Director Maestría en Ingeniería de Sistemas y Computación

Ingeniera Ángela Carrillo Ramos

Director Departamento de Ingeniería de Sistemas

Ingeniero Efraín Ortiz Pabón

Tabla de Contenido

Índice de tablas.

Índice de figuras.

Artículo 23 de la Resolución No. 1 de junio de 1946

"La Universidad no se hace responsable de los conceptos emitidos por sus alumnos en sus proyectos de grado. Sólo velará porque no se publique nada contrario al dogma y la moral católica y porque no contengan ataques o polémicas puramente personales. Antes bien, que se vean en ellos el anhelo de buscar la verdad y la Justicia"

AGRADECIMIENTOS

A mi familia y amigos por su permanente apoyo durante el desarrollo de este trabajo. Un agradecimiento especial a los profesores María Consuelo Franky y Jaime Andrés Pavlich Mariscal, por el permanente acompañamiento durante el curso de mis estudios.

ABSTRACT

The traditional software development process involves manual management of the entire lifecycle of an application. With the model-based development seeks to contribute to the improvement of delays, quality and productivity in development projects. This project sought to solve the problem of the manual web application development, to somehow mitigate the problems of quality, productivity and offsets estimates. It managed to build a code generator platform independent models with JavaEE code pages, drivers, services and entities.

The transformer was built with tools for application modeling and approaches to domain-specific languages. The generated application was validated with an adaptive application, which the transformer built in this work, turns out to be a complement to the generation of adaptive applications that manage their life cycle with different frameworks.

RESUMEN

El proceso tradicional del desarrollo de software implica la gestión manual de todo el ciclo de vida de una aplicación. Con el desarrollo basado en modelos se pretende aportar al mejoramiento de retrasos, calidad y productividad en proyectos de desarrollo. Este proyecto buscó solucionar el problema del desarrollo manual de aplicaciones web, para mitigar de alguna manera los problemas de calidad, productividad y desfases de las estimaciones. Se logró construir un generador de código de modelos independientes de plataforma a código en JavaEE con páginas, controladores, servicios y entidades.

El transformador se construyó con herramientas para el modelado de aplicaciones y con aproximaciones a lenguajes específicos de dominio. La aplicación generada se validó con una aplicación adaptativa, con lo cual el transformador construido en este trabajo, resulta ser un complemento en la generación de aplicaciones adaptativas, que gestionen su ciclo de vida con distintos frameworks.

Introducción

La forma de construir software ha venido adhiriendo nuevas metodologías, estándares y modelos a través de las últimas décadas. Esto con el objetivo de optimizar el proceso de desarrollo de software, aumentar la calidad y mantenibilidad de los productos construidos. De igual manera para disminuir costos, aumentar la productividad y mitigar de alguna manera los desfases de los proyectos de desarrollo de software. Múltiples metodologías y modelos se han implementado, tales como CMMI, RUP, metodologías ágiles, herramientas y estándares para diseño y modelado.

Desde la primera década del siglo XXI, se ha venido refinando el concepto de desarrollo dirigido por modelos, como respuesta a los recurrentes inconvenientes de los proyectos de software: desfases, falta de calidad y productividad. Además de la necesidad de portabilidad y demandas de nuevas tecnologías para evitar la obsolescencia de una aplicación entre otras. Múltiples herramientas han tomado estos principios para optimizar el desarrollo de software. Uno de ellos es Model-Driven-Enginering (MDE) que es el término que se utiliza para los procesos de desarrollo que son centrados en el modelo en lugar de centrados en el código. En MDE, los modelos son los primeros artefactos y pueden existir en múltiples niveles de abstracciones y someterse a transformaciones en otros modelos y / o código; dentro de los que se encuentra un modelo que se ajusta a un meta-modelo que define construcciones y normas para implementar estos mismos. En este aspecto un requisito esencial de un lenguaje de meta-modelado es su capacidad para capturar de forma concisa todos los aspectos de un lenguaje de modelado, incluyendo su sintaxis y la semántica (Aagedal, Mohagheghi, 2007).

Con base en estos principios, necesidades y conceptos se propone en este trabajo ZOE-GEN: Un transformador para facilitar la generación de aplicaciones basado en modelos. Con este trabajo se pretende concentrar los esfuerzos de los ingenieros y desarrolladores en construir modelos de una aplicación y no tanto en los detalles de la tecnología específica en la que se implementa el código. De igual manera se pretende ofrecer una herramienta de fácil manejo y configuración basada en sólidos conceptos y un óptimo diseño, que permita agregar nuevos elementos para el modelado de aplicaciones.

Para lograr lo anterior, se ha llevado a cabo un esfuerzo que es descrito en este documento de la siguiente manera: en la sección uno se encuentra una descripción general del proyecto y de la problemática que se quiere atacar. En la sección dos se encuentran los objetivos del proyecto y la descripción de las fases metodológicas definidas, desarrollo y validación. En la sección tres se encuentra el marco teórico donde se aborda más profundamente los conceptos de desarrollo dirigido por modelos, estándares, lenguajes específicos de modelado, el lenguaje ISML y sus componentes. De igual manera en esta sección se encuentra una descripción y comparación de los trabajos

relacionados con el transformador Zoe-Gen. En la sección cuatro se encuentran los requerimientos del transformador, consideraciones de licenciamiento, documentación y estándares aplicables. En la sección cinco se encuentra el diseño del transformador, con una breve descripción basada en el diseño 4+1. Posteriormente se encuentra la descripción de cómo se llevó a cabo la evaluación de los resultados. Finalmente se encuentran las conclusiones y consideraciones para el trabajo futuro, junto con las referencias del documento.

1. Descripción general

Oportunidad y problemática

Al momento de abordar la construcción de algún producto o solución informática, la forma tradicional de resolver dicho problema, es codificar de manera manual un sistema, que implemente los requerimientos y el diseño especificados por los ingenieros de software (Fleurey y Solberg, 2009). Este proceso conlleva una serie de riesgos, además supone una inversión de tiempo y recursos bastante alta. Inclusive durante el mismo, se pueden presentar errores que implican la realización de reprocesos para corregir fallas en el software, que se ven reflejadas en tiempos y costos.

De igual manera, diferentes problemas se presentan durante el ciclo de vida de desarrollo de software, como son los requerimientos inconclusos, según (Steen, Pires, y Iacob, 2010). Además de estimaciones desfasadas, errores de comunicación entre diferentes roles, exceso de defectos inyectados en las fases de desarrollo y requerimientos, que son difíciles de corregir. Los inconvenientes anteriores, deben ser atacados por herramientas y nuevas tecnologías, que soportan muchos de estos aspectos simultáneamente y permitan superar brechas, que hasta el momento, parecen no cerrarse ni dejar de afectar los procesos de desarrollo de software.

Los artefactos de diseño creados en las fases iniciales de un desarrollo, pierden rápidamente su valor cuando comienza la codificación, según (Keple, Warner y Bast, 2003), Esto porque en muchas ocasiones los documentos de diseño no reflejan realmente la implementación, debido a que no reflejan los cambios hechos en el código. Sumado a lo anterior se encuentra la necesidad de capacitar a las nuevas personas que hagan mantenimiento a una aplicación y la correspondiente curva de aprendizaje.

Frente a lo concerniente a la portabilidad, (Keple, Warner y Bast, 2003) anotan que la industria está demandando soluciones en nuevas tecnologías que se cobran vigencia rápidamente (Java, .Net, XML, HTML, JSP, Ruby). Por lo cual las compañías deben migrar en el menor tiempo posible de una tecnología a otra. El desarrollo tradicional hace este proceso muy costoso y demorado. En este sentido el problema de portabilidad está ligado a la dificultad de la industria para cambiar de tecnología. Adicionalmente se pueden presentar problemas en la generación de la documentación de una aplicación, lo que dificulta el mantenimiento. Lo anterior se explica porque la tarea del desarrollador de escribir documentación mientras programa, baja la productividad y no se completa de la manera más adecuada.

Como solución a estos problemas, la ingeniería dirigida por modelos (MDE por sus siglas en inglés) se presenta como una posible alternativa de solución. La ingeniería dirigida por modelos (Kent, 2002) es un enfoque para el desarrollo de software, el cual utiliza modelos para especificar los

diferentes aspectos de un sistema, desde los requisitos, pasando por el diseño, para llegar finalmente a la implementación. La inclusión de tecnologías como MDE, plantea nuevos paradigmas al complementar de una manera novedosa el proceso de desarrollo. Lo anterior, porque trata de introducir una nueva manera de apreciar el ciclo de desarrollo, al incorporar nuevos roles en las compañías de tecnología, replanteando el papel del desarrollador, y generando la necesidad de construir modelos elevando los niveles de abstracción de alto nivel (Hurtado, Bastarrica, Quispe y Ochoa, 2011).

Para poder manipular estos modelos se usan transformadores, entendiendo por transformador un artefacto que, al igual que los modelos, se construyen durante un proyecto (Kent, 2002), cuyo objetivo es realizar la conversión o mapping de un modelo a otro modelo, o de un modelo a código. Mediante el uso de transformadores es posible acelerar, hasta cierto punto, el ciclo de desarrollo de una aplicación, generando código de manera automática o semiautomática a partir de un modelo.

Un ejemplo de la solución expuesta anteriormente, es el proyecto Lion2 (Franky, Pavlich-Mariscal, 2015), en el cual se logró implementar un transformador que automatiza el proceso de desarrollo de software. Los modelos de entrada para el transformador en mención están especificados con el lenguaje ISML (Information Systems Modeling Language), el cual tiene la facilidad de expresar modelos de aplicaciones web, en un lenguaje textual. Los modelos son transformados en código fuente, basado en diferentes tecnologías específicas (Franky, Pavlich-Mariscal, 2015: 2). Este transformador es de código cerrado, propiedad de la compañía Heihnsohn Bussiness Tecnology. Se enfoca en la tecnología JEE6 y está integrado al framework de la compañía.

2. Descripción del proyecto

Con base en los resultados de dicho proyecto y las lecciones aprendidas, el presente trabajo construirá el transformador ZOE-GEN de modelos, especificados en ISML, hacia Java EE. Este transformador será licenciado de forma libre. Tendrá elementos adicionales, recogiendo las actividades de trabajo futuro del proyecto Lion2, tales como: una parametrización más detallada del proceso de generación de código y nuevos componentes visuales. En la Figura 1, se presenta la arquitectura de referencia según el estándar MDA, mencionado anteriormente. En este contexto, ISML permite crear modelos independientes de plataforma (PIM), mientras que transformará dichos modelos en código específico para la plataforma de destino.

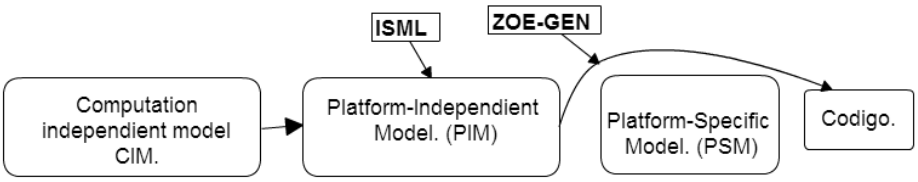

Figura 1 Arquitectura MDA

Una aplicación concreta a corto plazo de ZOE-GEN es el uso del transformador por parte del framework MiDAS (Model-Driven Approach for Adaptive Systems), el cual, ofrece un enfoque basado en modelos para el desarrollo adaptativo. En concreto, MiDAS ofrece: (i) un nuevo lenguaje para especificar de manera formal y abstracta los requerimientos, (ii) otro lenguaje para especificar el diseño de un sistema adaptativo y (iii) un mecanismo para generar el código de un sistema adaptativo de manera automática, a partir de las especificaciones de requerimientos y diseño(Bocanegra, Pavlich-Mariscal y Carrillo-Ramos, 2015).

La especificación de requerimientos de ZOE-GEN, tendrá en cuenta las necesidades de MiDAS. De esta manera se crearán componentes reutilizables a nivel de modelo y código, que serán útiles en el contexto de este framework. Se espera que el transformador objetivo de este trabajo, así como otros similares que se creen en el futuro, sirvan de intermediarios para hacer transformaciones a diversas plataformas sin alterar el CIM, que es donde se encuentran los modelos de Midas

2.1. Objetivo general

Construir un transformador para generar una aplicación en Java EE a partir de un modelo independiente de plataforma, especificado en ISML.

2.2 Objetivos específicos

1. Especificar los requerimientos del transformador.

2. Diseñar el transformador.

3. Desarrollar el transformador, basado en la especificación del diseño obtenida en el objetivo anterior.

4. Validar el transformador a través de una prueba de concepto.

2.3 Fases de desarrollo

2.3.1 Fases Metodológicas

Para el desarrollo y pruebas de los transformadores se utilizará SCRUM siguiendo a Rising y Janoff (2000) como metodología de desarrollo ágil adaptada al contexto. Las fases definidas se pueden ver en la figura 2. A estas fases se agregara una de validación del transformador.

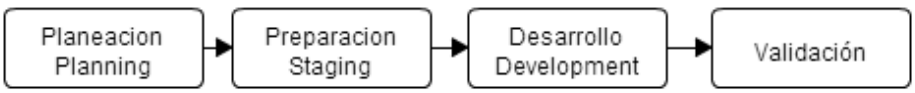

Figura 2Fases del Trabajo de Grado. Adaptado de Larman, C. (2004).

En la fase de Planeación se realizarán actividades correspondientes a la construcción de la propuesta y las validaciones hechas en la exposición del poster del trabajo. En la fase de preparación se definirán los requerimientos y se planearan los sprints. Posteriormente en la fase de desarrollo se construirá el transformador en base al desarrollo en plantillas y en la validación se ejecutarán las pruebas de aceptación al transformador.

En cuanto a los roles, el director del trabajo de grado actuará como Scrum Master y Dueño del Producto, liderando los encuentros, controlando el avance de los sprints y validando el contenido del backlog.

Adicionalmente se utilizarán las herramientas de apoyo al método Kanban, que se usa en unidades de desarrollo de software y equipos de proyecto para visualizar el flujo de trabajo (como se cita en Ahmad, Markkula, y Oivo, 2013).

2.3.2 Implementación del Sistema

En esta actividad se construirán los componentes del transformador para generar los distintos elementos de la tecnología JEE y se especificarán modelos para realizar las pruebas de esta transformación.

Antes de cada sprint se actualizará y se priorizan las tareas respectivamente. En cada sprint se desarrollarán las plantillas de generación de código siguiendo el proceso de la figura 3. Estas actividades se realizarán por parte del estudiante, con la dirección del Scrum Master. Se comenzará con la construcción de una aplicación de referencia en código, junto con un modelo que la represente en ISML. Luego se construirán y modificarán las plantillas para generar una aplicación equivalente a partir del modelo en ISML. Se generará el código, para proseguir con la corrección y depuración de los defectos encontrados y la propagación de los cambios de vuelta a las plantillas (Franky, Pavlich-Mariscal, 2012).

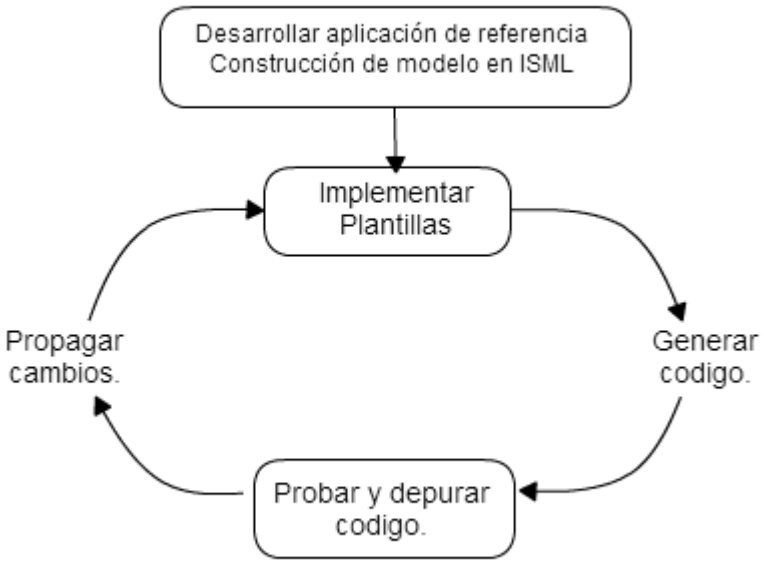

Figura 3. Desarrollo de software basado en plantillas

Basado en la experiencia adquirida en la construcción de transformadores en el proyecto Lion 2 (Franky, Pavlich-Mariscal, 2015), se seguirán utilizando herramientas como Xtext y Xtend para su implementación (Bettini, 2013).

2.3.3 Validación del Sistema

Tras la ejecución exitosa de las fases anteriores, se pretende validar el transformador a través de una serie de pruebas de aceptación. Estas implican la ejecución de la implementación del software, con datos de prueba. Se examinan las salidas del software y su entorno operacional para comprobar que funciona tal y como se requiere (Sommerville, 2005). Cada requerimiento seleccionado tendrá asociado un caso de prueba debidamente documentado. Estas pruebas se llevarán a cabo con una aplicación tomada como caso de estudio, Prenat, sistema adaptativo de control prenatal (Bocanegra, Hernandez, Olarte, 2014). Esta aplicación contiene unos requerimientos adaptativos basados en unas funcionalidades específicas, las cuales se pretenden probar a partir de una aplicación generada con el transformador ZOE-GEN.

3. Marco teórico / estado del arte

Antes de describir algunos de los elementos del desarrollo dirigido por modelos, es importante tener en cuenta claros algunos conceptos. En primer lugar se encuentra el concepto de modelo de un sistema que se refiere a una descripción o una especificación de ese sistema y su entorno para desempeñar un determinado objetivo (Mellor, Stephen J et al., 2004). Los modelos se presentan normalmente como una combinación de texto y dibujos. El texto se puede presentar en lenguaje de modelado, o en lenguaje natural. Cuando se hace referencia a MDA se dice que es dirigido por modelos porque usa los modelos para dirigir el ámbito del desarrollo, el diseño, la construcción, el despliegue, la operación, el mantenimiento y la modificación de los sistemas (Frankel David S, 2003).

También se encuentra el concepto de una plataforma, que es un conjunto de subsistemas y tecnologías que aportan un conjunto coherente de funcionalidades a través de interfaces y determinados patrones de uso, que cualquier aplicación que se construya para esa plataforma puede usar sin preocuparse por los detalles de la implementación o cómo se lleva a cabo la misma dentro de la plataforma (Lucredio, Santana de Almeida y Fortes, 2012). A partir de lo anterior es importante tener en cuenta la independencia de la plataforma es una cualidad que tienen que presentar los modelos. Lo que significa que un modelo es independiente de las facilidades o

características que implementan las plataformas, de cualquier tipo (Kleppe Anneke et al., 2003; Arlow Jim, Neustadt Ila, 2004).

3.1 Ingeniería dirigida por modelos

El proceso de desarrollo de software ha evolucionado y se ha formalizado en las últimas décadas, adquiriendo madurez a través de metodologías, modelos e interacciones con otras disciplinas. Lo cual se refleja en que muchas empresas buscan la implementación de modelos de madurez como CMMI (Beecham, 2003). No obstante persisten inconvenientes de calidad, retrasos en los proyectos o aumento de costos, entre otros; lo cual demanda la aplicación general de metodologías y buenas prácticas orientadas a los procesos de desarrollo de software.

En este contexto la "Ingeniería basada en modelos (MDE) está orientada a utilizar modelos para todas las etapas del proceso de ingeniería de software (requisitos, diseño, etc.). En particular, la intención del MDE es utilizar herramientas de transformación para generar automáticamente el código fuente a partir de modelos. Como consecuencia, un conjunto de modelos de una aplicación puede ser utilizada para generar software para diferentes plataformas y marcos" (Franky, Pavlich-Mariscal, 2015). MDE se perfila como un acercamiento para abordar los problemas típicos del desarrollo de software.

La inclusión de tecnologías como MDE, plantea nuevos paradigmas al complementar de una manera novedosa el proceso de desarrollo. Lo anterior porque trata de introducir una nueva manera de apreciar el ciclo de desarrollo al incorporar nuevos roles en las compañías de tecnología, replanteando el papel del desarrollador y generando la necesidad de construir modelos elevando los niveles de abstracción de alto nivel en el proceso de construcción de estos mismo (Hurtado, Bastarrica, Quispe, Ochoa, 2011).

En este orden de ideas "el aumento de la productividad y la disminución del tiempo de desarrollo, la mejora de la calidad, automatización, estandarización y formalismo, mantenimiento, mejora de la comunicación y el intercambio de información" (Mohagheghi ,Dehlen,2008) está intrínsecamente relacionado con la ingeniería dirigida por modelos por lo que para aliviar esta situación "algunos han tratado de aplicar los procesos de software pre-existentes a MDE, como usando una versión modificada del Rational Unified Process (RUP) , y la combinación de metodologías ágiles y MDE. Otros han intentado procesos definitorios para MDE. En primer lugar, THALES ha definido un proceso MDE mediante la ampliación del estándar 1471 del IEEE (Mohagheghi, Dehlen, 2008)."

Algunas muestras del uso de MDE vienen dadas por la búsqueda de "la trazabilidad a lo largo de los artefactos de desarrollo de software, la evaluación temprana, la promoción de la reutilización, portabilidad de soluciones a nuevas plataformas, y la capacidad para estimar los costos basado en los modelos" (Mohagheghi,Dehlen, 2008).

El grado de incertidumbre en cuanto a estimaciones de un proyecto se reduciría únicamente al momento en el que se generan los modelos principales ya que los atributos de calidad de un modelo en MDE contienen transformabilidad, en donde un modelo debe tener la capacidad de ser transformado a otros modelos con mayor detalle y para piezas ejecutables de código, y Modificabilidad, para que los cambios realizados en los requisitos se representen correctamente en los modelos y se reflejen en el código (Aagedal, Mohagheghi,2007).

MDE también incluye el desarrollo iterativo dentro del ciclo de vida del software, contempla la creación de modelos con requisitos explícitos de las partes interesadas, lo que se constituye en una primera garantía de calidad (QA) puesto que pasa con la comprobación de tipos, validación semántica y la transformación de éstos en los modelos intermedios, y una segunda etapa de control de calidad con validación estática de modelos (Aagedal, Mohagheghi,2007). En el mismo sentido, para los desarrolladores los lenguajes de modelado proveen un mecanismo para elevar el nivel de abstracción. Utilizando lenguajes de modelado, los modelos representan objetos en el dominio del problema, no en el contexto del código. Los lenguajes de modelado siguen las abstracciones del dominio y la semántica (Kelly Tolvanen, 2008).

Se evidencian aspectos a ser tenidos en cuenta para estudios detallados de todo el potencial que MDE ofrece como complemento al proceso de desarrollo de software para resolver algunas de las necesidades de calidad, estimación y productividad que todo proceso requiere. En referencia a esto, se pueden observar ventajas de la utilización de MDE en el desarrollo de software en base a unos casos de estudio en donde se evidencia la aplicación de estos conceptos. Como aporte para esto se tienen dos perspectivas importantes como los son Productividad y Calidad.

Dado que hay mucha evidencia en el aumento de la productividad desde la visión de la generación de código a partir de modelos, es importante considerar que dicha productividad no necesariamente se ve reflejada desde el momento de la implementación del modelo, porque esta puede ser una tarea compleja, pero que al estar finalizada pretende ser reutilizada (Lucredio, Santana de Almeida y Fortes, 2012). Es decir, que la implementación de los mismos modelos puede ser aprovechada en diferentes proyectos, diferentes plataformas y/o diferentes etapas del mismo proyecto.

Con los componentes del ciclo de vida del software, como son Análisis de Requerimientos, Diseño de Sistema, Diseño de Arquitectura, Diseño de Módulos, Codificación, Instalación y Mantenimiento (Hashimi H, Hafez A, Beraka M, 2012), se requiere de roles que en “la ingeniería de software comprendan muchas actividades diferentes, incluso si se atribuyen principalmente a los "desarrolladores". Aun cuando no se han actualizado completamente, roles tales como ingeniero de requerimientos, analista de sistemas, arquitecto, etc., a menudo existen tácitamente” (Hutchinson, Whittle, Rouncefield, Kristoffersen, 2011). Estos inconvenientes están enmarcados dentro de un

problema de productividad, puesto que todo el proceso descrito anteriormente se realiza a bajo nivel.

3.1.1 Perspectiva de calidad

En MDE la reutilización de software es alcanzada a través de generación de código fuente, dentro de lenguajes de dominio específico y transformaciones de software (Lucredio, Santana de Almeida y Fortes, 2012). Se parte de la idea de modelos que pueden ser llevados a líneas de código fuente que no requieren de la manipulación de programadores. Vista esta característica MDE puede entonces traer grandes beneficios a nivel de estimación de costos y calidad de productos.

Si se observa que el desarrollo de software basado en reutilización de componentes tiene alto grado de popularidad. Algunos arquitectos y directores de proyectos generalmente quieren hacer uso de componentes ya desarrollados en lugar de promover el desarrollo desde cero, apreciando así la reutilización de módulos de proyectos previos, si existen (Lucredio, Santana de Almeida y Cortes, 2012).

Dentro de los beneficios esperados en la reutilización del software, las organizaciones apuntan a potenciales como reducción de costos y esfuerzo de desarrollo, menores tiempos de desarrollo para salir más rápidamente al mercado y así mismo disminuir el costo del mantenimiento. Con reutilización sistemática de software, se involucran en una técnica empleada para direccionar la necesidad por mejorar el desarrollo de software eficiente y de calidad. Esto envuelve el uso de artefactos que pertenecen a sistemas existentes para construir nuevos en orden de mejorar la calidad y la mantenibilidad y reducir el costo en el tiempo de desarrollo (Rothenberger, Dooley, Kulkarni, Nada, 2003).

Como sabemos hay técnicas de estimación y cuantificación del producto que se basan en líneas de código fuente, las cuales pueden ser subdivididas en nuevas, reusadas, editadas y eliminadas. La consideración de elementos que ya existen o que son reutilizados permite estimaciones en los proyectos más acertadas. Es por esto que en métodos de estimación como PROBE se considera el código que es reutilizado dentro de la estimación total del proyecto. PROBE usa LOC y KLOC para cuantificar el tamaño y esfuerzo de los productos; de igual forma si líneas de código no se consideran apropiadas se sugiere la utilización de objetos, tablas de bases de datos, puntos de función, pantallas u otros niveles de granularidad que se necesiten (Nasir, 2006).

Dentro de la búsqueda de calidad de los productos en la industria del software y el mejoramiento continuo de los procesos de desarrollo de software, vemos como esto es visto dentro del marco de Personal Software Process (PSP). El cual dentro de sus características propone registrar las actividades de un ingeniero para desarrollar un producto y de los elementos que definen el producto desarrollado.

En la etapa de planeación de PSP se propone estimar el tamaño del proyecto, para lo cual se debe contar con los elementos necesarios para emplear PROBE y por lo tanto estar direccionados en las siguientes etapas: primero determinar los objetos requeridos para desarrollar un producto a partir de un diseño global, segundo determinar el tipo y número probable de métodos para el desarrollo del producto, tercero referirse a la información histórica de los productos similarmente desarrollados y utilizar una regresión lineal para encontrar un probable tamaño del producto total, que de acuerdo con la información histórica la regresión será mayormente acertada (Nasir,2006).

La información histórica es asumida a partir de los scripts de desempeño de PSP. Dichos scripts o plantillas, recopilan los elementos del producto como lo son (Nasir, 2006):

- Base: Cuando se mejora un producto existente, las LOC base se definen a partir del tamaño del producto original antes de alguna modificación.

- Adición: El código que se adiciona en un nuevo programa o se adiciona a la base del programa en mejora.

- Modificación: Las modificaciones sobre las líneas de código base del programa a ser modificado.

- Borrado: Líneas de código borradas de la base del producto.

Para los cuales se puede inferir en que la base será aquella calculada por LOC generadas a partir de la transformación de los modelos, algo que es controlado y no es susceptible de incertidumbre o variabilidad, como sucede con cualquier código desarrollado desde cero.

Visto entonces desde la perspectiva de MDE, en donde los modelos finalmente deben ser traducidos en código fuente, promoviendo la reutilización del código (Hurtado, 2011); se cuenta con fuentes estables y confiables de código que no requieren de la manipulación humana para su generación. Esto mejora calidad, porque gracias a la influencia de las propiedades de las tecnologías del modelado se reduce significativamente la accidentalidad asociada con la manipulación manual del software complejo (France, R.; Rumpe, B, 2007). Además, el uso de lenguajes de modelado ofrece algunos beneficios, tendiendo a la elevación de los niveles de abstracción: favoreciendo la productividad, facilitando el ocultamiento de la complejidad y favoreciendo la calidad del sistema (Kelly Tolvanen, 2008).

En un exitoso caso de estudio de MDE (Hutchinson, J., Rouncefield, M., & Whittle, J, 2011), se observa como la reutilización del código es una de las principales características de éxito. En el estudio realizado en una compañía de impresoras, donde se requería software embebido para hardware de impresión, se alcanzaron los siguientes beneficios a partir del entendimiento y descubrimiento de MDE. Lograron desarrollar una estrategia de reutilización de código a partir de modelos, para desarrollo de software embebido de alta calidad. Aunque en el caso no se dan cifras

exactas del impacto de la utilización de MDE, se menciona la reducción de un 50 % de los recursos necesarios para el desarrollo de software.

Sin embargo, no se debe dejar atrás que la generación automática de código a partir de modelos puede ser un inconveniente cuando el tema no es claro para las personas que hacen parte de proyectos del tipo de implementación de MDE. En un caso de estudio no exitoso, se conoce una fábrica multinacional de sistemas electrónicos para las telecomunicaciones (Hutchinson, J., Rouncefield, M., & Whittle, J, 2011), en donde se reconoce que en un despliegue sin el apropiado conocimiento y dominio de MDE, se generaron modelos que proporcionaban cantidades incontrolables de código fuente. Respecto a este caso el código no era manejable por ingenieros de software, quienes por el tipo de negocio tenían limitantes memoria que les impedía usar el código generado por los modelos.

Si se realiza una comparación entre el caso de éxito y el caso sin éxito, se puede ver que la intención de la aplicación de MDE es la reutilización de los modelos para generar código fuente de calidad. En el caso de éxito se observa que la virtud se encuentra en la buena definición de los modelos, los cuales si se encuentran correctamente definidos y se usan apropiadamente son un factor de éxito en cuanto a calidad del producto.

Sin embargo, este aspecto se puede mitigar en alguna medida a partir de propiedades que se pretenden evaluar, en principio, con métricas sugeridas como lo son dinamicidad (la complejidad de una clase interna, comportamiento basado en las llamadas de mensajes, las transiciones de estado DIT (la profundidad del árbol de herencia), la cohesión (qué parte de una clase se necesitan para realizar una sola tarea), UCN (el número de clases por casos de uso), y NUC (el número de casos de uso por clase)(Mohagheghi ,Dehlen,2008); métricas que si bien no dan el impacto final sobre el proyecto permiten establecer un estándar de calidad de la aplicación de la técnica y realizar estimaciones basadas en complejidad.

Por otro lado para profundizar más en el desarrollo basado en modelos, la iniciativa Model-Driven Architecture (MDA), desarrollo basado en modelos de OMG (MDD) o Ingeniería Dirigida por Modelos (MDE) ha sido aclamada como la solución a manejar problemas claves a los que se enfrenta la industria de desarrollo de software (Mohagheghi ,Dehlen,2008). No obstante, aunque MDE afirma muchos beneficios potenciales, sobre todo las ganancias en productividad, portabilidad, facilidad de mantenimiento y la interoperabilidad, este se ha desarrollado en gran medida con el apoyo de datos empíricos (Hutchinson, Whittle, Rouncefield, Kristoffersen, 2011). Vale la pena cuantificar acertadamente si se dan impactos positivos o negativos con el uso de esta tecnología dentro de los proyectos de software, los cuales se ven diariamente afectados por el tema de incertidumbre, dada la falta de información precisa y cuantificable para las evaluaciones de impacto y toma de decisiones. La forma que plantea MDA para hacerlo, es especificar la

funcionalidad de un sistema con un modelo independiente de alguna plataforma y luego traducir este modelo en un modelo de una plataforma específica y código fuente totalmente ejecutable (Calic, Dascalu, Egbert, 2008).

3.1.2 Estándar MDA

El estándar para MDA (Model Driven Architecture), definido por el OMG (Object Management Group, 2014). MDA especifica tres puntos de vista distintos de un sistema, y sugiere el uso de modelos en las distintas etapas del ciclo de desarrollo como son: CIM, que es un modelo que se enfoca en requerimientos, y está especificado usando los conceptos del dominio como vocabulario; PIM, que corresponde a un modelo independiente de plataforma y de detalles de tecnología; y PSM, que hace referencia a un modelo con detalles específicos de una plataforma. En el contexto del estándar MDA se encuentra: MDA-Cartridges, un Cartucho MDA o MDA-Cartridge contiene las reglas necesarias para realizar una transformación de modelos. Pueden ser instalados como plugin, descargarse de Internet, y editarse o extenderse si es necesario.

Para complementar lo mencionado en secciones anteriores, Model Driven Architecture (MDA), es un framework para el desarrollo de software, definido por el Object Management Group (OMG) (Keple, Warmer, Bast, 2007). MDA propone tres tipos de modelos que se constituyen en su parte principal como lo son:

Platform Independent Model (PIM): es el primer modelo que define MDA, tiene un alto nivel de abstracción independiente de la tecnología de implementación. Describe un sistema de software que soporta algún negocio, por lo cual está modelado desde este punto de vista.

Platform Specific Model (PSM): En el ciclo de vida de MDA, presentado en la figura 1, el PIM es transformado en un modelo específico de plataforma (PSM). Un PSM mapea directamente la implementación en términos de una tecnología específica. Un modelo de tipo PIM, puede ser transformado en uno o más PSM. Para cada plataforma especifica, un modelo PSM es generado, por lo cual es común tener varios modelos PSM para un PIM (Keple, Warmer, Bast, 2007).

Código: Es el paso final del desarrollo después de la transformación de cada PSM en código. Debido a que un modelo PSM, es muy cercano a una tecnología especifica, la transformación a código suele ser sencilla.

MDA además de definir los modelos y el código, también define como están relacionados. Un PIM después de creado, es transformado en uno o varios PSMs, el cual posteriormente es transformado en código. El proceso más complejo en el desarrollo con MDA es la forma como un modelo PIM es

transformado en uno o más PSMs como se detalla en la figura 4. Según Keple, Warmer y Bast, el proceso de desarrollo con MDA puede resultar muy similar a la manera de desarrollo tradicional. Esto explicado en la forma como se elevan los niveles de abstracción y se realizan los modelos de acuerdo a las necesidades de un negocio específico. No obstante recalcan que existe una crucial diferencia, ya que en el desarrollo tradicional las transformaciones entre modelos o de modelos a código, se suelen realizar de forma manual. Existen muchas herramientas que generan código a manera de plantillas, no obstante la mayoría de la codificación debe realizarse de manera manual.

Figura 4. Adaptado de (Kleppe, Warmer, Bast, 2003).

En contraste estos mismos autores exponen cómo las transformaciones de modelos, orquestadas por MDA, siempre son ejecutadas por herramientas de manera automática. Esta es la principal diferencia con el desarrollo tradicional. En este orden de ideas las transformaciones de modelos de tipo PSM a código suelen ser sencillas, según se expuso anteriormente y existen muchas herramientas que realizan este tipo de transformación, por lo cual no es novedoso. Como por ejemplo las transformaciones generadas de modelos UML a código en alguna tecnología específica. MDA aporta la transformación automática entre modelos calificados de alto nivel como son los de tipo PIM a uno o varios PSM. Todo esto conlleva las ventajas de productividad, portabilidad e interoperabilidad que se describieron anteriormente.

Al momento de escribir código, la aproximación de MDA es muy novedosa. Como resultado de esta, las herramientas actuales no son lo suficientemente sofisticadas para proveer las utilidades necesarias para llevar a cabo una transformación de modelos de tipo PIM a PSM y de PSM a código en un cien por ciento. En estos casos, el desarrollador necesita completar la transformación de manera manual. No obstante, las herramientas actuales son capaces de ofrecer una funcionalidad básica, por lo cual el desarrollador puede obtener una retroalimentación automática y un prototipo de manera eficiente (Keple, Warmer, Bast, 2007).

En este orden de ideas, el ciclo de vida de MDA descrito, involucra varios modelos en su desarrollo. Por lo que se puede profundizar más en el concepto de transformación. Se han mencionado herramientas de transformación que pueden interactuar como modelos de tipo PIM y convertirlos en modelos PSM. La misma herramienta u otras pueden tomar el modelo PSM y transformarlo en código.

Las herramientas de transformación definen las reglas de los elementos que interactúan dentro de la misma. Inclusive en algunos casos, describen como los modelos deben ser transformados. Siguiendo a Keple, Warmer y Bast, quienes anotan que se debe notar la distinción entre la transformación en sí misma, como el proceso de generar un nuevo modelo a partir de otro modelo y lo que llaman *definición de la transformación,* que hace referencia a las especificaciones de cómo se debe realizar la transformación. Las herramientas de transformación usan la misma definición de la transformación para cada conversión de un modelo de entrada.

En general se puede decir que una transformación es la generación automática de un modelo a partir de otro modelo fuente, de acuerdo a la definición de la transformación. Una definición de una transformación consiste en una colección de reglas de transformación con especificaciones bien definidas acerca de cómo un modelo puede ser usado para crear otro. Una regla de transformación se refiere a la descripción de cómo uno o más constructos en un modelo fuente pueden ser transformados en uno o más constructos en un lenguaje de destino (Keple, Warmer, Bast, 2007).

3.1.3 Meta modelos

Un meta modelo define los elementos de un modelo y cómo estos pueden ser usados al momento de construir un modelo. Un meta modelo es en sí mismo un modelo, por lo tanto debe ser escrito en un lenguaje bien definido. En este orden de ideas, un metalenguaje juega un rol diferente en el framework de MDA, porque se especializa en un lenguaje que describe otros lenguajes de modelado. Por lo tanto un modelo está escrito en un lenguaje y este a su vez está escrito en un metalenguaje, por lo cual no hay distinción entre un lenguaje y el meta modelo que define el lenguaje (Keple, Warmer, Bast, 2007).

Los meta modelos son importantes en el framework de MDA debido a que las reglas de transformación describen como un modelo puede ser transformado en un lenguaje de destino. Según Keple, Warmer y Bast esas reglas usan los meta modelos del modelo independiente de plataforma y las fuentes de lenguajes de destino, para definir las transformaciones.

3.1.4 Capas de modelado de OMG

El estándar de la OMG para MDA, define una serie de capas de modelado que se usan como estándares para este tipo de arquitecturas. Estas se clasifican en distintos niveles: M0, M1, M2 y M3. A continuación se realizará una aproximación a cada uno de estos.

Capa M0: En esta capa se encuentran las instancias reales de un modelo. A este nivel se modelan por ejemplo, elementos del negocio e items que son tangibles como entidades. Estas son representaciones del mundo real.

Capa M1: Corresponde a la definición de modelos en los que se categorizan los elementos descritos en la capa M0. Para el modelamiento de software se pueden representar a nivel de modelos en UML, en donde se describen instancias más específicas de la capa M0 representadas como el modelo de una aplicación de software.

Capa M2: En este punto se pueden crear elementos más específicos aumentando el nivel de abstracción. Entre estos ítems se pueden encontrar, clases y componentes entre otros. De manera análoga a lo expresado en la capa M1, cada elemento de M2 corresponde a una instancia de M1 respectivamente. En este nivel se alojan los meta modelos, como modelos de un modelo.

Capa M3: Los elementos de la capa M3 a su vez son instancias de los ítems de la capa M2. La notación que es usado para representar un meta meta modelo es la misma notación usada para especificar un meta modelo y un modelo (Keple, Warmer, Bast, 2007). Según OMG el estándar aplicable para la capa M3 es MOF. El cual se revisará en la sección 5.3.

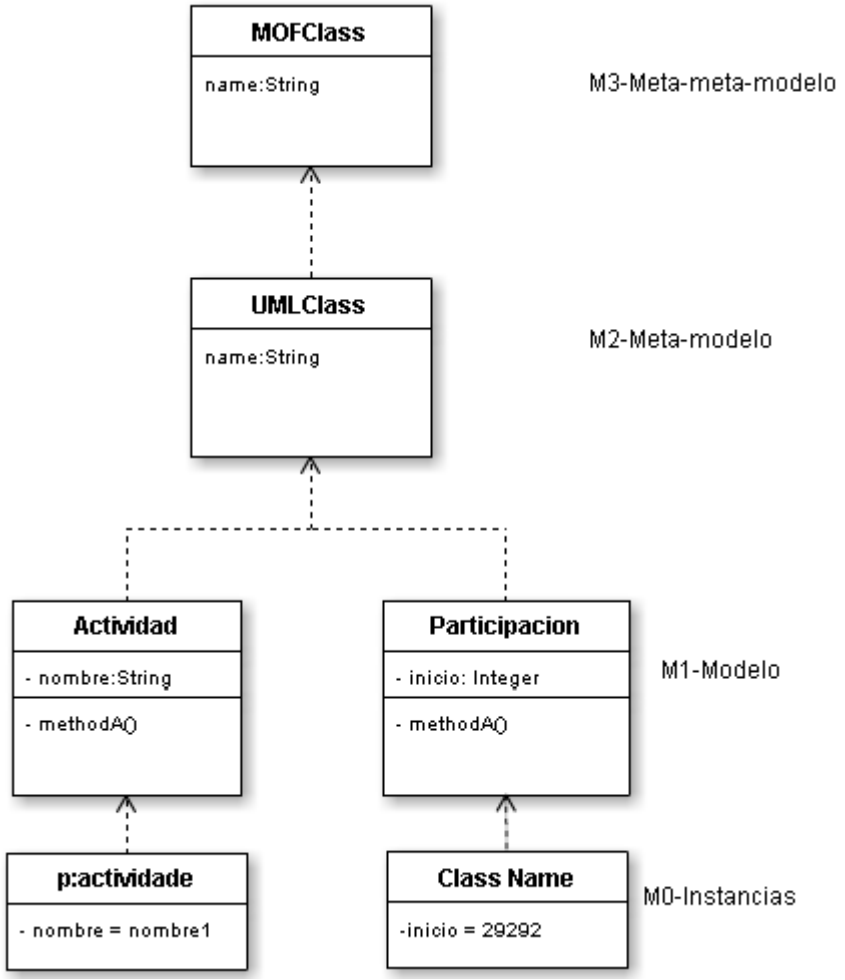

Figura 5. adaptada de (Kleppe, Warmer, Bast, 2003)

En la figura 4, se pueden observar las distintas capas de modelado de OMG. A partir de estas es posible entender los distintos niveles de abstracción que se presentan dentro de MDA. Con ellos se permite especificar un modelo de negocio a alto nivel y continuar su definición con herramientas de modelado hasta tener representaciones más específicas. Se entiende que cada elemento de negocio está representado en uno o varios componentes de bajo nivel, permitiendo la trazabilidad de cada ítem.

3.1.5 Lenguajes de dominio específico

Los lenguajes de dominio específico elevan los niveles de abstracción y proveen la automatización necesaria. Un elemento clave en el elevamiento de estos niveles de abstracción es la posibilidad de tener lenguajes de modelado que mapean más de cerca el problema del dominio. Los lenguajes de dominio específico esconden la complejidad mientras los desarrolladores continúan construyendo los diseños de un dominio en particular. Para proveer la automatización, los modelos necesitan ser mapeados a implementaciones de código. En estos casos es necesario construir generadores y frameworks de soporte (Kelly Tolvanen, 2008).

El concepto de DSM (Lenguajes de dominio específico), cobra especial relevancia en el contexto de la construcción del transformador ZOE-GEN, puesto que se utiliza un DSM para mapear los modelos de un dominio a código en una tecnología específica. Por lo demás, DSM se enfoca en la automatización del desarrollo de software y del desarrollo manual en una definida área de interés. Como su nombre lo indica, es de dominio específico, más que de dominio general. En este orden de ideas, DSM puede soportar las tareas de desarrollo desde el lenguaje de modelado acerca del problema de dominio y los generadores pueden crear la solución del problema desde el lado de la implementación (Kelly Tolvanen, 2008).

EL lenguaje usado para este trabajo y que se enmarca en el contexto descrito anteriormente, es ISML. Utilizando las especificaciones de este lenguaje se implementaron los componentes necesarios para el transformador ZOE-GEN. Por medio de la especificación de modelos en ISML, se logra realizar el mapeo entre los modelos y el código.

3.1.6 Lenguaje ISML

Dentro del proyecto Lion2 (Franky, Pavlich-Mariscal, 2015) se definió el lenguaje de dominio específico denominado **ISML** (Information Systems Modeling Language) el cual permite modelar aplicaciones web en un lenguaje textual sencillo. Este modelado se realiza dentro de un ambiente de

desarrollo (basado en el IDE Eclipse) que permite escribir modelos en el lenguaje ISML y transformarlos a código.

Los principales elementos de un modelo en ISML describen la estructura y comportamiento de un sistema de información web, basado en el paradigma Modelo-Vista-Controlador (MVC). Los elementos de un modelo esenciales de un modelo en ISML son:

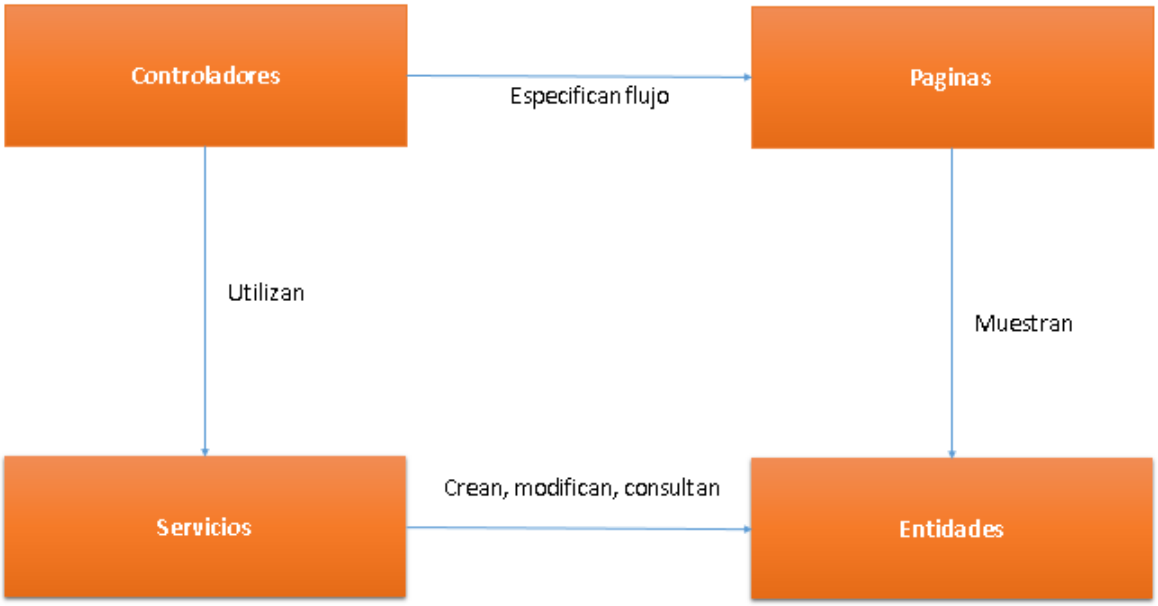

Figura 6. Adaptado de MDE: Manual para el mantenimiento del generador de código JEE6 para una aplicación web modelada en ISML.

1. Controladores. Se encargan de recibir las peticiones del usuario, invocar servicios y mostrar las páginas correspondientes.
2. Páginas. Describen la información visual a presentar al usuario.
3. Servicios. Encapsulan algoritmos y procesos específicos del sistema de información, relacionados normalmente con la lógica del negocio. Los servicios son los encargados de manipular y recuperar entidades.
4. Entidades. Corresponden a las entidades de negocio persistentes del sistema.

3.1.7 Elementos del lenguaje

Para ilustrar de una manera más óptima los componentes del lenguaje representados en la figura 5, se visualizara mediante un breve ejemplo que modela una aplicación en el lenguaje ISML con los componentes básicos de un desarrollo web. Los modelos ISML se presentan con estructuras muy parecidas a las de proyectos regulares de Java. Los archivos que están contenidos dentro de cada sección del modelo deben tener la extensión **.ISML.** Cada uno de estos archivos contiene alguno de

los elementos referenciados en la figura 5, como: entidades, controladores, páginas, servicios. Para poder utilizar los componentes en las distintas partes de los proyectos, deben importarse como funciona una aplicación regular en un lenguaje como Java. Para visualizar de manera más clara los elementos del lenguaje, se recurrirá a algunos fragmentos del modelo construido para validar el transformador Zoe-Gen, haciendo referencia a la aplicación Prenat que se describirá más detalladamente en la sección de validación de este documento.

A. Entidades

Las entidades representan una unidad de negocio, en ISML las entidades están compuestas de una sección de importación de elementos necesarios, declaración y unos atributos, tal y como se puede revisar en la figura 6:

```
package co.edu.javeriana.entities;

entity Dieta  {

    String desayuno must be Size(1,10);
    String almuerzo;
    String cena;
    String merienda;
    Integer patologia must be NotNull;
    Preferencias preferencia opposite preferencias;

}
```

Figura 7. Entidad Dieta

Las entidades se definen con la palabra clave **entity** que hace parte de la gramática de ISML. En la figura se puede apreciar que los atributos de la entidad están definidos según unos tipos primitivos de datos similares a los de Java. Asimismo las entidades de ISML, pueden ser abstractas y tener relaciones de herencia con otras entidades. Por lo demás con la palabra clave **must be** se denotan las restricciones de tamaño y obligatoriedad de cada uno de los campos. La palabra **opposite** denota la relación de atributos entre dos entidades, en este caso Preferencias como se ve en la siguiente figura que detalla dicha entidad.

```
package co.edu.javeriana.entities;

entity Preferencias {
    String preferencia;
    Dieta preferencias;
    String otro;

}
```

```
package co.edu.javeriana.entities;

entity Gustos extends Preferencias{

    String tipoDieta must be NotNull;
    String moda;
}
```

Figura 8. Entidades Gustos y Preferencias.

En esta entidad se encuentra un atributo de tipo Dieta que denota la cardinalidad de esta relación entre las entidades. Por otro lado para reflejar las relaciones maestro detalle a manera de herencia entre las entidades. Donde por medio de la palabra **extends** se denotan la relación entre las entidades Gustos y Preferencias entidades.

B. Páginas

Las páginas representan la capa de presentación de la aplicación en el modelo. Las páginas corresponden a un archivo con extensión .ISML y pueden contener distintos widgets propios del lenguaje. Un ejemplo de una página en ISML se puede visualizar en la figura 8:

```
package co.edu.javeriana;
import co.edu.javeriana.prenat.resource.*;
import co.edu.javeriana.entities.* ;

page DietaList( Collection<Dieta> dietaList) controlledBy DietaManager  {

    Form {
           Panel("Collection<Dieta>") {

               DataTable("Collection<Dieta>", null) {
                   header : {
                       Label("Desayuno");
                       Label("Almuerzo");
                       Label("Cena");
                       Label("Merienda");
                       Label("Patologia");
                       Label("View");
                       Label("Edit");
                       Label("Delete");
                   }
                   body :
                   for(Dieta dieta in dietaList) {
                        Label(dieta.desayuno);
                        Label(dieta.almuerzo);
                        Label(dieta.cena);
                        Label(dieta.merienda);
                        Label(dieta.patologia);
                        Button("View")-> viewDieta(dieta);
                        Button("Edit") -> editDieta(dieta);
                        Button("Delete") -> deleteDieta(dieta);


                   }

               }


                Button("Dieta Adaptativa") -> showDietaAdaptativa();
       }
    }

}
```

Figura 9. Pagina DietaList

La página tiene la estructura de un archivo ISML definida anteriormente. Se define por medio de la palabra clave **page**, una página en ISML además puede recibir o no atributos. Por medio de la palabra clave **controlledBy** se especifica el controlador a la cual está asociada. A su vez la página contiene una serie de elementos gráficos entre los que se encuentran: Form, Paneles, Datatables, labels y botones. Los botones utilizan acciones del controlador, lo cual se modela por medio del operador `->`

C. Controladores

Los controladores especifican flujo de las páginas y soportan por medio de acciones las funcionalidades especificadas en las mismas. Además tienen la inyección de servicios que aportan la persistencia y otras funcionalidades. La estructura de un controlador corresponde a la definida de manera general para archivos ISML. Un ejemplo de un controlador se puede apreciar en la figura 8:

```
package co.edu.javeriana ;
import co.edu.javeriana.entities.* ;

controller DietaManager {
    has Persistence<Dieta> persistence;
    has Persistence<Lugares> persistenceLug;
    has DietaPersonal dietapersonal;
    has Query query;

    /**
    * Lists all instances of Dieta.
    */
    default listAll() {
        show DietaList(persistence.findAll());
    }

    default initTest(){
        -> listAll();
    }

    /**
    * Lists instances of Dieta.
    * @param dietaList The list of instances of Dieta to show.
    */
    listDieta(Collection<Dieta> dietaList) {
        show DietaList(dietaList);
    }

    createDietaToAdd(Any container, Collection<Dieta> collection) {
        show CreateDietaToAdd(container, collection, new Dieta);

    }

    /**
    * Views an instance of Dieta.
    * @param dieta the Dieta to open.
    */
    viewDieta(Dieta dieta) {
        show ViewDieta(dieta);
    }
```

```
showTiendas() {
     Query q = query.select().from("Lugares").where("tipo = 'lug'");
     List<Lugares> model = persistenceLug.execute(q);

     show RestaurantMapView(model, new MapModel);
 }
```

Figura 10. Controlador DietaManager

Los controladores se declaran mediante la palabra clave del lenguaje **controller,** así mismo por medio de la palabra **has** se denota la inyección de servicios para este ejemplo se encuentran los servicios: Persistence<Dieta>, Persistence<Lugares>, DietaPersonal y Query. Estos servicios proveen distintas funcionalidades de persistencia y construcción de queries en SQL. Posteriormente en el controlador se encuentran definidos una serie de métodos que invocan estos servicios o re direccionan a otras páginas por medio de la palabra clave **show,** la cual especifica un flujo de páginas dentro de la aplicación.

D. Servicios

En un modelo ISML se cuenta inicialmente con el servicio genérico Persistence, el cual se parametriza con una entidad. Este servicio tiene los métodos básicos de persistencia sobre la base de datos, entre los que se encuentran update, delete, create y load, complementados por los métodos findAll y isPersistent. A continuación se muestra la enumeración de métodos que constituyen el

servicio Persistence:

```
service Persistence < T > {
/**
     * Deletes an object from the database.
     * @param obj The object to delete
     */
    native Void remove(T obj);
    /**
     * Deletes an object from the database.
     * @param id The id of the object to delete
     */
    native Void remove(Integer id);
    /**
     * Creates a new object in the database.
     * @param obj The object to create
     */
    native Void create(T obj);
    /**
     * Returns all the objects of type T in the database.
     */
    native Array < T > findAll();
    /**
     * Returns all the objects of type T in the database, except the given collect
     * @param elementsToExclude The instances to not retrieve from the database
     */
    native Array < T > findAllExcept(Collection < T > elementsToExclude);

    /**
     * Determines if an object has already been saved in the database or not.
     */
    native Boolean isPersistent(T obj);

    /** Finds an object by its id.
     * @param id The id of the object to find
     */
    native T find(Integer id);
```

Figura 11. Servicio Persistence

Aparte de Persistence se pueden definir otros servicios de negocio. Estos servicios enumeran métodos de negocio (al estilo de una interfaz) para cada entidad de negocio que lo use, sin proveer ninguna implementación, la cual será aportada por el generador de código. Por lo demás existen servicios como el llamado Query, que provee implementaciones para generar queries en una aplicación mediante el estandar de jpql.

```
service Query  {

    native Query select(String query);

    native Query select();

    native Query from(String ent);
    native Query and(String ent);

    native Query where(String condition);



}
```

Figura 10. Servicio Query

Cada uno de los métodos del servicio Query está definidos de tipo **native**, lo cual indica que la implementación de los mismos se realizará una vez generada la aplicación. No obstante el generador ZOE-GEN provee la implementación de estos servicios básicos.

3.2 trabajos relacionados

Existen varias herramientas que sirven para la generación de código que usan las los principios y estándares de MDE. No obstante existen algunas diferencias y características propias de cada una. En esta sección se realizará una descripción de los trabajos relacionados con el transformador Zoe-Gen, realizando una descripción y comparación con base en unos criterios definidos.

3.2.1 Descripción general de trabajos:

En primer lugar entre los trabajos que se van a tomar como referencia según su relación con el transformador Zoe-Gen, se encuentra DynJava. Este es un lenguaje está basado en Java para generación de código dinámico. Además provee tipos precisos de fragmentos de código dinámico, para garantizar la seguridad de tipos de códigos de forma dinámica y compuesta (Oiwa, Y., Masuhara, H., & Yonezawa, A , 2001).

También se encuentra el trabajo Sculptor en el que se expresa un diseño en una especificación textual, de la que el transformador Sculptor genera código Java de alta calidad. Utiliza los conceptos de conceptos de Domain-Driven-Design (DDD) en el lenguaje específico de dominio textual (DSL). Por ejemplo este transformador puede generar servicios, módulos, entidades y repositorio (Sculptor, 2015).

Otro de los trabajos consultados es ArcStyler: es un sistema basado en uso de cartuchos para descripción de transformaciones que permite generar aplicaciones de n capas codificadas en

java/J2EE y c#/.NET a partir de diagramas UML y la especificación de los procesos del negocio (Warmer Jos, Kleppe Anneke, 2003).

De igual manera se encuentra OptimalJ: este producto de la compañía Compuware genera aplicaciones J2EE partiendo de los modelos. Implementa completamente la especificación MDA. Está desarrollado en Java, lo que le hace portable a cualquier plataforma para su ejecución (Corredera de Colsa Luis Enrique, 2007).

La herramienta AndroMDA es un sistema basado en cartuchos, que admite como entrada descripciones XMI de diagramas UML, y usa XDoclet como tecnología de marcado para el acceso a datos desde las clases Java. Admite como entrada ficheros XMI versión 1.1, y como herramienta de modelado la comunidad de desarrollo aconseja el uso de Poseidón para UML, de Gentleware.

3.2.2 Descripción específica de trabajos:

DynJava es un lenguaje extendido para Java en la que el usuario puede escribir fragmentos de código dinámicos en una sintaxis que es similar a la de Java. Además, los fragmentos de código dinámico en DynJava se escriben de forma estática, y la seguridad del código escrito forma dinámica compuesta está garantizada de forma estática (Oiwa, Y., Masuhara, H., & Yonezawa, A, 2001). Los creadores de este lenguaje implementaron un sistema que traduce el programa DynJava a un programa Java. Cuando las solicitudes de los usuarios para generar una clase y una instancia de las especificaciones del código, el sistema de ejecución invoca el generador de código para producir byte code, luego le pide cargador de clases de Java para cargar la clase generada.

La herramienta OptimalJ admite XMI versión 1.1 tanto para la importación de ficheros como para su salida. OptimalJ es una herramienta MDA que utiliza MOF para soportar estándares como UML y XMI. Se trata de un entorno de desarrollo que permite generar aplicaciones J2EE completas a partir de un PIM.

Con la herramienta Sculptor el código generado se basa en marcos muy conocidos, tales como JPA, Hibernate, Spring Framework y Java EE. Sculptor se encarga de los detalles técnicos, el trabajo repetitivo tedioso, y permiten a los ingenieros centrarse en ofrecer un mayor valor de negocio (Sculptor, 2015).

3.2.3 Fortalezas y debilidades de trabajos relacionados

Para describir las fortalezas y debilidades de cada trabajo, se definieron una serie de variables. Estas variables se asignaron según una escala definida con los ítems: Alta, Media y Baja. Esta ponderación fue asignada según las características, fortalezas y debilidades de cada herramienta.

Para el caso de la herramienta DynJava el usuario puede escribir dinámico fragmentos de código utilizando construcciones de lenguaje de alto nivel. Mediante la introducción de un contexto que tiene varios información sintáctica, así como asociaciones de variables, sistema de tipos de DynJava garantiza de modo pasivo el typesafety de que los fragmentos de código dinámicamente compuestas. La actual implementación demuestra que la sistema se puede utilizar para poner en práctica fácilmente la optimización dinámica de un programa de FFT (Oiwa, Y., Masuhara, H., & Yonezawa, A, 2001). Una de las principales ventajas del trabajo mencionado, radica precisamente en la mantenibilidad que se le puede asignar al código generado ya que la generación dinámica, puede funcionar cuando se realizan múltiples generaciones de código. No obstante el uso de la herramienta dificulta el versiona miento del código. De igual manera los componentes que se pueden modelar con esta herramienta, no contienen la totalidad de los componentes necesarios para una aplicación web estándar.

	Lic.	**Com. Adaptativo**	**Entidades**	**Servicios**	**Controladores**	**Pág.**	**Repos**	**Diagramas UML**	**Multiplataforma**
DynJava	Open Source	Media	Media	Media	Media	Baja	Baja	Baja	Baja

Tabla 1. Trabajo DynJava

El trabajo AndroMDA admite cualquier lenguaje de programación como salida, y admite código propio para la generación de código. No obstante Aunque se basa en MDA, no basta sólo con los modelos para llegar a un despliegue, es necesario que el desarrollador intervenga el código y por lo tanto requiere que éste tenga un buen conocimiento de la plataforma(Cuesta, López, Joyanes ,2009). De esta manera se evidencia una fortaleza y una desventaja con este trabajo, puesto que si bien tienen una alta portabilidad porque genera código en cualquier plataforma, se requiere la necesidad de expertos en cada tecnología para completar el código y su funcionalidad.

	Lic.	**Com. Adaptativo**	**Entidades**	**Servicios**	**Controladores**	**Pág.**	**Repos**	**Diagramas UML**	**Multiplataforma**
AndroMDA	BSD license	Baja	Media	Alta	Media	Baja	Alta	Baja	Alta

Tabla 2. Trabajo AndroMDA

La herramienta OptimalJ a partir de un modelo de clases permite crear de forma sencilla y en poco tiempo, una aplicación básica para la plataforma J2EE, generando código de buena calidad (Cuesta, López, Joyanes, 2009). Lo anterior lo realiza a partir de un modelo típico de clases de UML. En

ArcStyler el esfuerzo de desarrollo es mayor, ya que el programador utiliza PIM's "marcados" en reemplazo de PSM's así como también la creación de etapas intermedias para poder llegar al modelo de destino (código o PSM)(Cuesta, López, Joyanes ,2009).

	Lic.	**Com. Adaptativo**	**Entidades**	**Servicios**	**Controladores**	**Pág.**	**Repos**	**Diagramas UML**	**Multiplataforma**
OptimalJ	Comercial.	Baja	Alta	Alta	Media	Baja	Baja	Alta	Baja

Tabla 3. OptimalJ

En lo que se refiere a las ventajas de la herramienta Sculptor la aplicación puede desarrollarse gradualmente con un bucle eficiente de ida y vuelta. Sculptor es útil en el desarrollo de aplicaciones empresariales o web típicas que se benefician de un modelo de dominio rico y persistente.
En cuanto al transformador ZOE-GEN, objeto del presente trabajo, permite modelar aplicaciones en el lenguaje ISML, con la mayoría de componentes típicos de una aplicación web. La construcción de estos modelos es particularmente sencilla, puesto que tiene elementos comunes con los de las aplicaciones normales y su configuración no es complicada. Por otro lado la generación de código es bastante completa y la intervención del programador es mínima, puesto que la aplicación se genera prácticamente lista para desplegar, basada en un arquetipo predefinido. El transformador está liberado bajo la licencia Apache 2.0, lo cual facilita su enriquecimiento. Por lo demás provee algunas configuraciones básicas de componentes adaptativos, que integrados con otras herramientas, como el framework MIDAS, podrían modelar todo el ciclo de vida de una aplicación adaptativa. No obstante, el alcance de este trabajo no contempla modelar todo el ciclo de vida y realizar ingeniería inversa. Los transformadores están diseñados para realizar el mapeo a la tecnología específica de Java siguiendo el patrón vista controlador. Tampoco se incluye una generación dinámica de código, ni se pueden realizar modelos en UML.

	Lic.	**Com. Adaptativo**	**Entidades**	**Servicios**	**Controladores**	**Pag**	**Repos**	**Diagramas UML**	**Multiplataforma**
Sculptor	Apache 2.0	Baja	Alta	Alta	Media	Baja	Alta	Baja	Baja

Tabla 4. Trabajo Sculptor

ArcStyler permite extender las capacidades de transformación, generando nuevos cartuchos a partir de UML, cuyo objetivo sea cualquier plataforma o lenguaje. No soporta diagramas de componentes ni diagramas de despliegue, pero admite código propio para la generación de código. ArcStyler de

iO-Software es una herramienta MDA que también utiliza MOF para soportar estándares como UML y XMI, y además JMI para el acceso al repositorio de modelos. Integra herramientas de modelado (UML) y desarrollo (ingeniería inversa, explorador de modelos basado en MOF, construcción y despliegue) con la arquitectura CARAT que permite la creación, edición y mantenimiento de cartuchos MDA (MDA-Cartridge) que definen transformaciones. También incluye herramientas relacionadas con el modelado del negocio y el modelado de requisitos por lo que cubre todo el ciclo de vida (Cuesta, López, Joyanes ,2009).

	Lic.	**Com. Adaptativo**	**Entidades**	**Servicios**	**Controladores**	**Pag**	**Repos**	**Diagramas UML**	**Multiplataforma**
ArcStyler	Comercial	Baja	Alta	Alta	Media	Baja	Baja	Alta	Alta

Tabla 5. Trabajo ArcStyler

A continuación se presenta una tabla en la que se realiza una comparación de los trabajos seleccionados, teniendo en cuenta una serie variables: La variable licencia se refiere a si es comercial u open source y finalmente se evalúa si existe la posibilidad de modelar componentes adaptativos para la aplicación. Seguido a esta se encuentra completitud, desglosada en una serie de componentes que cada herramienta genera o no, teniendo en cuenta la cantidad de características que tiene el código generado y la necesidad de intervención del desarrollador. Entre estas se encuentran entidades, servicios, controladores, páginas, repositorios. De igual manera se utiliza un criterio que indica si el trabajo relacionado soporta diagramas UML y finalmente si genera código en varias tecnologías.

Finalmente se encuentra el transformador objeto de este trabajo, que describe las mismas variables. Se puede decir que si bien cada una de las herramientas tomadas que tienen relación con el transformador Zoe-Gen, tienen ventajas a la luz de las variables analizadas también presentan falencias en algunos aspectos.

	Lic.	**Com. Adaptativo**	**Entidades**	**Servicios**	**Controladores**	**Pag**	**Repos**	**Diagramas UML**	**Multiplataforma**
Zoe-Gen	Apache 2.0	Alta	Alta	Alta	Alta	Alta	Baja	Baja	Baja

Tabla 6. Trabajo Zoe-Gen

Sobre todo es de resaltar la poca disposición para construir componentes adaptativos a partir de los elementos gráficos y la generación de queries. Por otro lado la completitud del código en las herramientas con la ponderación media, implican que se debe completar el código o que no generan una aplicación completa. Por lo demás se evidencia que en cuanto a complejidad de uso de transformación las herramientas que permiten el modelado de manera textual, resultan ser más complicadas de usar. Finalmente se observa una comparación desde el punto de vista del licenciamiento.

4. Requerimientos

Este documento contiene el listado de los requerimientos correspondientes enmarcados dentro del proyecto: ZOE-GEN: Un transformador para facilitar la generación de aplicaciones basado en modelos. La especificación recoge únicamente requerimientos funcionales propios del transformador, partiendo del concepto del transformador como base para poder manipular estos modelos se usan transformadores, entendiendo por transformador un artefacto que, al igual que los modelos, se construyen durante un proyecto (Kent, 2002).

Los requerimientos tienen en cuenta los elementos necesarios para poder generar aplicaciones web, recogiendo los artefactos típicamente más usados tales como: Controladores, servicios, Páginas y elementos gráficos para estas últimas. Estos mismos tienen como base las experiencias previas de trabajos similares y el proceso de recolección llevado a cabo a manera de reutilización.

De igual manera y con el objetivo de favorecer la flexibilidad del generador se tienen requerimientos que favorecen la generación de aplicaciones adaptativas, como la generación de queries en JPQL y elementos gráficos parametrizables. Lo anterior se llevó a cabo por medio de reuniones con expertos en software adaptativo y en el framework Midas, del cual será parte el transformador ZOE-GEN.

4.1 Suposiciones y Dependencias

- El código generado estará basado en la plataforma de destino Java Enterprise Edition 7
- La aplicación generada con el transformador ZOE-GEN se ejecutará desde cualquier navegador.
- La aplicación generada usará la base de datos PostgreSQL.
- El código generado se basará en un arquetipo siguiendo el patrón MVC.

- Se utilizará el servidor de aplicaciones WildFly 10.0.0 Final para el despliegue de la aplicación generada con el transformador Zoe-Gen
- Se utilizará jpql y EclipseLink para la generación de sentencia y como proveedor de persistencia de la plataforma específica respectivamente.

4.2 Legal, Copyright y licenciamiento.

El software producto de este trabajo de grado se liberará al público bajo licencia Apache 2.0. (Apache, 2004) Todo lo que no sea código fuente será liberado al público bajo licencia Creative Commons Attribution 4.0 International (Creative Commons, 2004).

4.3 Interfaces

4.3.1 Interfaces de usuario

Los elementos utilizados para ingresar, procesar y entregar los datos: teclado, ratón y pantalla visualizador.

4.3.2 Interfaces de Software

Eclipse Modeling Tools: Construido en el marco de proyecto de modelado de Eclipse, se usará para construir los modelos, plantillas y componentes del generador, para las implementaciones en tecnologías específicas. El transformador se relaciona con eclipse en la medida que es la herramienta principal de modelado de componentes en modelos independientes de plataforma (Eclipse Modeling Framework, 2008). La relación con el transformador de eclipse y sus componentes, es la realización en tiempo de ejecución de los modelos independientes de plataforma a código en java.

ISML: Los modelos son el insumo del transformador para mapear las características modeladas a código. El lenguaje de dominio específico denominado ISML (Information Systems Modeling Language) permite modelar aplicaciones web en un lenguaje textual sencillo. En el marco del generador ZOE-GEN, permitirá modelar aplicaciones en modelos independientes de plataforma. ISML provee una suerte de lenguaje de específico de modelado con el que se modelan los componentes de un modelo independiente de plataforma. Además provee las características básicas de los componentes de una aplicación web.

Xtext: El lenguaje de la gramática es la piedra angular de xtext. Es un lenguaje de dominio específico, cuidadosamente diseñado para la descripción de lenguajes textuales (Xtext, 2005). Los componentes del lenguaje ISML se encuentran especificados con Xtext y junto con las herramientas de modelado de Eclipse, se utiliza como base para la construcción de las plantillas del generador y el modelado.

4.3.3 Estándares aplicables

Meta Object Facility (MOF) proporciona un marco de gestión de metadatos, y un conjunto de servicios de metadatos para habilitar el desarrollo y la interoperabilidad de los sistemas de metadatos basado en modelos y. Ejemplos de estos sistemas que utilizan MOF incluyen el modelado y herramientas de desarrollo, sistemas de almacenamiento de datos, repositorios de metadatos, etc. (Omg, 2008).

MOF es el estándar oficial que usa EMF (Eclipse Modeling Framework). El proyecto EMF es un mecanismo marco de modelado y generación de código para herramientas y otras aplicaciones basadas en un modelo de datos estructurado construcción. A partir de una especificación del modelo se describe en XMI, EMF proporciona herramientas y soporte de ejecución para producir un conjunto de clases Java para el modelo, junto con un conjunto de clases de adaptadores que permiten la visualización y edición de comandos basados en el modelo, y un editor básico. Contiene además EMF (core) es un estándar común para los modelos de datos, muchas de las tecnologías y los marcos. Esto incluye soluciones de servidor, frameworks de persistencia, los marcos de la interfaz de usuario y soporte para transformaciones (EMF, 2008). En el contexto del transformador ZOE-GEN, el estándar MOF, se aplica siguiendo los lineamientos de MDA, en donde se especifican los distintos niveles de modelado. Específicamente para especificar elementos de la meta modelo que dan soporte a la ejecución del transformador.

4.4 Lista de requerimientos

En la figura 12 presentada a continuación se presenta una relación consolidada de la mayoría de los requerimientos definidos para el transformador, junto con una descripción breve. El detalle de los mismos se puede encontrar en el documento de especificación de requerimientos anexo a este documento. Por lo demás los requerimientos se encuentran clasificados en: funcionales para el transformador y solicitados para facilitar el uso del transformador por el framework MIDA

Nombre	Descripción
Req01	El sistema debe generar una aplicación con entidades que contengan atributos, métodos de encapsulamiento, métodos equals, hash code.
Req02	El sistema debe generar una aplicación con entidades que manejen herencia, constraints y relaciones de cardinalidad.
Req03	El sistema debe generar una aplicación con un servicio genérico de persistencia, con las opciones de crear, editar, eliminar, consultar además de filtrar y ordenar.
Req04	El sistema debe generar controladores con atributos y métodos especificados en el modelo.
Req05	El sistema debe generar archivos de configuración como faces-config.xml para garantizar la navegación.
Req06	El sistema debe generar archivos de configuración como persistence.xml para garantizar el funcionamiento de la persistencia.
Req07	El sistema debe generar páginas con elementos gráficos como botones.
Req08	El sistema debe generar páginas con elementos gráficos como cajas de texto.
Req09	El sistema debe generar páginas con elementos gráficos como labels.
Req10	El sistema debe generar páginas con elementos gráficos como spinners.
Req11	El sistema debe generar páginas con elementos gráficos como radio buttons.
Req12	El sistema debe generar páginas con elementos gráficos como paneles.
Req13	El sistema debe generar páginas con elementos gráficos como data tables.
Req14	El sistema debe generar páginas con elementos gráficos como menús.
Req15	El sistema debe generar páginas con elementos gráficos como mapas.
Req16	El sistema debe generar páginas con elementos gráficos como calendarios.
Req18	El sistema debe generar páginas con elementos gráficos como mensajes.
Req19	El sistema debe generar páginas con elementos gráficos como contenedores de imágenes.
Req20	El sistema debe permitir parametrizar los componentes gráficos para cambiar sus propiedades.
Req21	El sistema debe permitir generar servicios personalizados a partir de un modelo.

Req22	El sistema debe permitir cambiar el orden de los componentes desplegados en una página.
Req23	El sistema debe permitir cambiar el color o forma de los componentes desplegados en una página.
Req24	El sistema debe permitir generar servicios que apoyen este proceso de decisión.

Tabla 7. Tabla de requerimientos del transformador.

4.5 Documentación y ayuda para requerimientos de usuario

ID	Detalle	Restricciones	Solicitante
22	Se debe generar un manual para el mantenimiento del generador e ilustrar cómo agregar nuevos componentes.		ZOE-GEN
23	Se debe generar un manual del usuario que ilustre el modelamiento, generación y despliegue de una aplicación generada.		ZOE-GEN

Tabla 8. Tabla de requerimientos de ayuda para el usuario..

5. Diseño de la solución

A continuación se presentan las consideraciones técnicamente más relevantes que se han tenido en cuenta para el funcionamiento del transformador. Describiendo la arquitectura del transformador y visualizar los componentes de los que está compuesto. De igual manera se puede consultar el anexo para tener las especificaciones más detalladas.

5.1 Vista de procesos

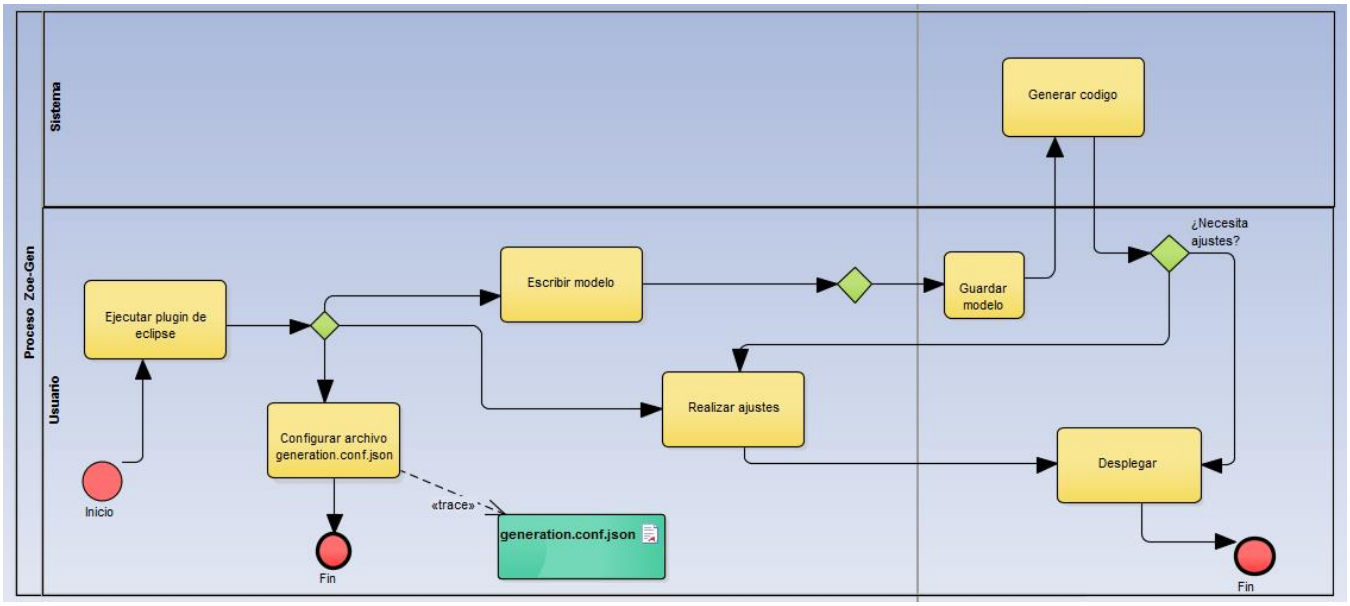

Figura 12: Vista de procesos de Zoe-Gen.

Como se puede apreciar en la figura 12 el proceso inicia cuando se ejecuta el plugin de eclipse que tiene los componentes del transformador. Lo anterior ejecuta una nueva instancia de eclipse en la que se encuentra el ambiente para el modelado de artefactos en isml independientes de plataforma. Paso seguido el usuario puede editar el archivo de configuración generation.conf.json para definir las generalidades sobre el uso del transformador y su lugar de ejecución. De igual manera el flujo indica que el usuario puede escribir modelo para posteriormente generar código o en caso contrario editarlo. Si el código requiere ajustes propios concernientes a las funcionalidades de negocio se ajustan y posteriormente se desplegará la aplicación generada.

5.2 Vista Física

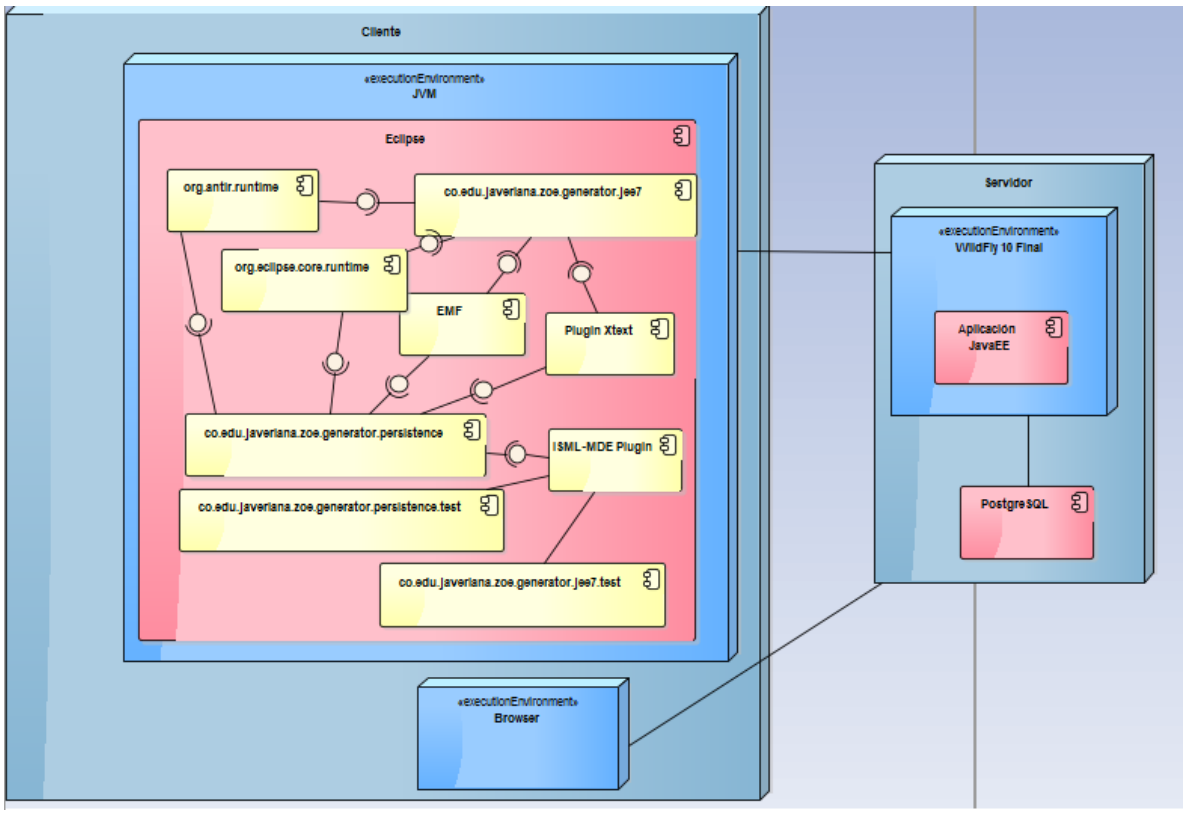

Figura 13: Vista física Zoe-Gen.

En la figura dos se encuentran los principales componentes del transformador Zoe-Gen y su respectiva ubicación. En este contexto se encuentran dos nodos que se ejecutaran en una sola maquina: el primero corresponde al entorno de ejecución de la máquina virtual de java que contiene donde se ejecutan los plugins e instancias de eclipse necesarios para la generación de código en Java Empresarial y el navegador con el que se accede a la aplicación. En el nodo servidor esta contenido Wildfly 10 Final se encuentra el servidor de aplicaciones que alberga el código generado en java. Adicionalmente se encuentra un componente que representa la base de datos PostgreSQL. En la siguiente tabla se encuentra una breve descripción de estos componentes.

5.3 Vista Lógica

El propósito de esta vista es mostrar las decisiones arquitecturales para la implementación del transformador ZOE-GEN, se incluyen todos los subsistemas identificados. Esta vista, también conocida como de desarrollo, está orientada a la especificación de cómo se construirá y configurará el sistema en términos de software según sus componentes y estilos arquitecturales escogidos.
Esta vista presenta cada uno de los componentes identificados, de acuerdo al siguiente diagrama en el cual se aprecian los principales componentes junto con paquetes que tienen la función de realizar las pruebas unitarias al sistema.

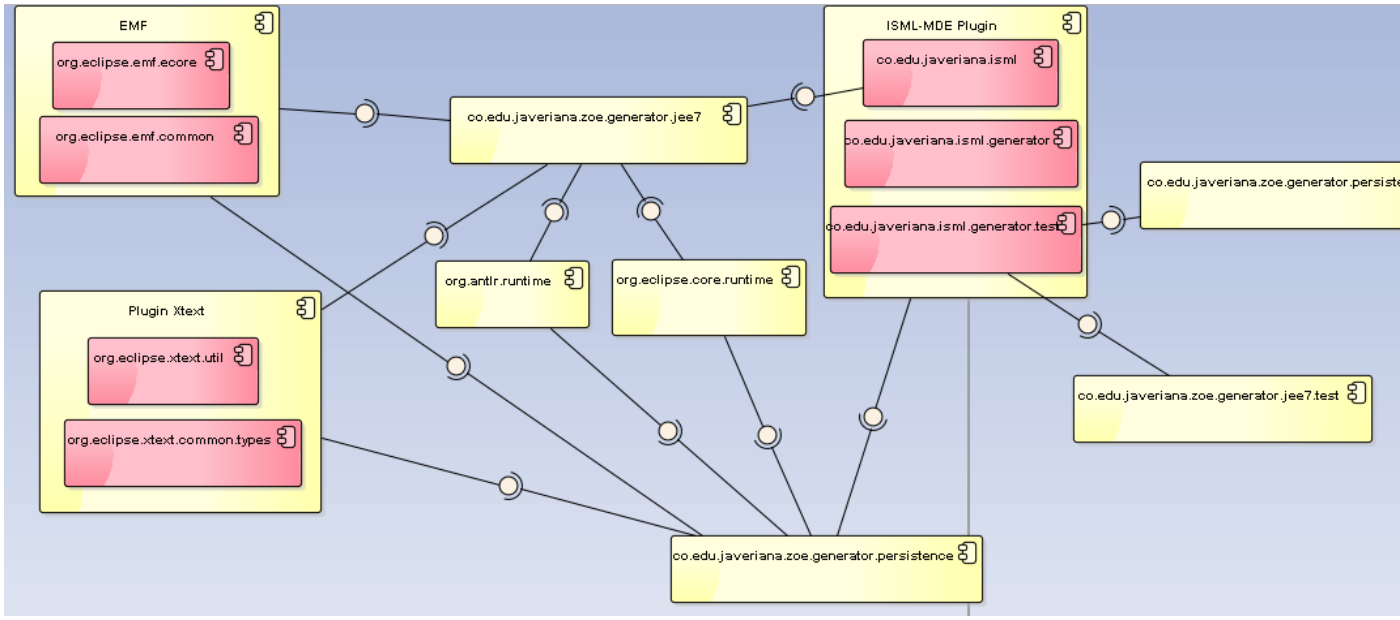

Figura 13: Vista Lógica.

Los principales componentes del transformador ZOE-GEN contiene la configuración principal del funcionamiento del generador. En estos se encuentran principios y características propias del lenguaje de modelado específico ISML tales como: abstracción, automatización, flexibilidad y control de versiones, integración con otros frameworks (Pavlich-Mariscal, 2015). En este orden de ideas los componentes corresponden a conjuntos de clases que pretenden definir las reglas de transformación para el mapeo de un modelo independiente de plataforma a una tecnología en particular sin pasar por un modelo específico de plataforma. Los componentes están definidos de tal manera que agrupan distintos elementos del transformador según los elementos que se vayan a utilizar del lenguaje ISML.

5.3.1 Paquetes de arquitectónicamente significativos.

Los componentes que sobresalen por su importancia en el contexto de la implementación del transformador según se puede observar en la figura 16 son los siguientes:

Componente	Descripción
co.edu.javeriana.zoe.generator.jee7	El componente que incluye las utilidades y funcionalidades para la transformación de entidades, servicios, recursos de idioma de un modelo independiente de plataforma a código en Java.
co.edu.javeriana.zoe.generator.jee7.test	El componente hace parte del generador y contiene las funcionalidades para ejecutar las pruebas unitarias de entidades, servicios, recursos de idioma, modelados en ISML y con su respectiva correspondencia a código en una tecnología específica.
co.edu.javeriana.zoe.generator.persistence.	En este se encuentran las utilidades y funcionalidades para mapear los elementos del lenguaje ISML a código en Java. Las utilidades que contiene este componente son: reglas de transformación para el mapeo de controladores, páginas, servicios específicos, interfaces para los mismos, archivos de configuración para navegabilidad en las páginas.
co.edu.javeriana.zoe.generator.jee7.	El componente test contiene a su vez las utilidades necesarias para realizar pruebas unitarias y del sistema para controladores, páginas, servicios específicos, interfaces para los mismos y archivos de configuración para navegabilidad en las páginas. Estos modelados

	en ISML y con su respectiva correspondencia a código en una tecnología específica.

Tabla 9. Paquetes de arquitectónicamente significativos.

A continuación se detallarán las particularidades y las clases de cada paquete expuesto como arquitecturalmente significante. Esto con el fin de dar más claridad sobre el diseño del transformador y su implementación.

A continuación se presenta la vista de implementación del transformador ZOE-GEN, con el fin de proveer la guía y detalle de la implementación de cada componente del sistema. Se presenta el detalle de los paquetes y clases incluidos en cada componente. Además se presentan las relaciones entre cada uno.

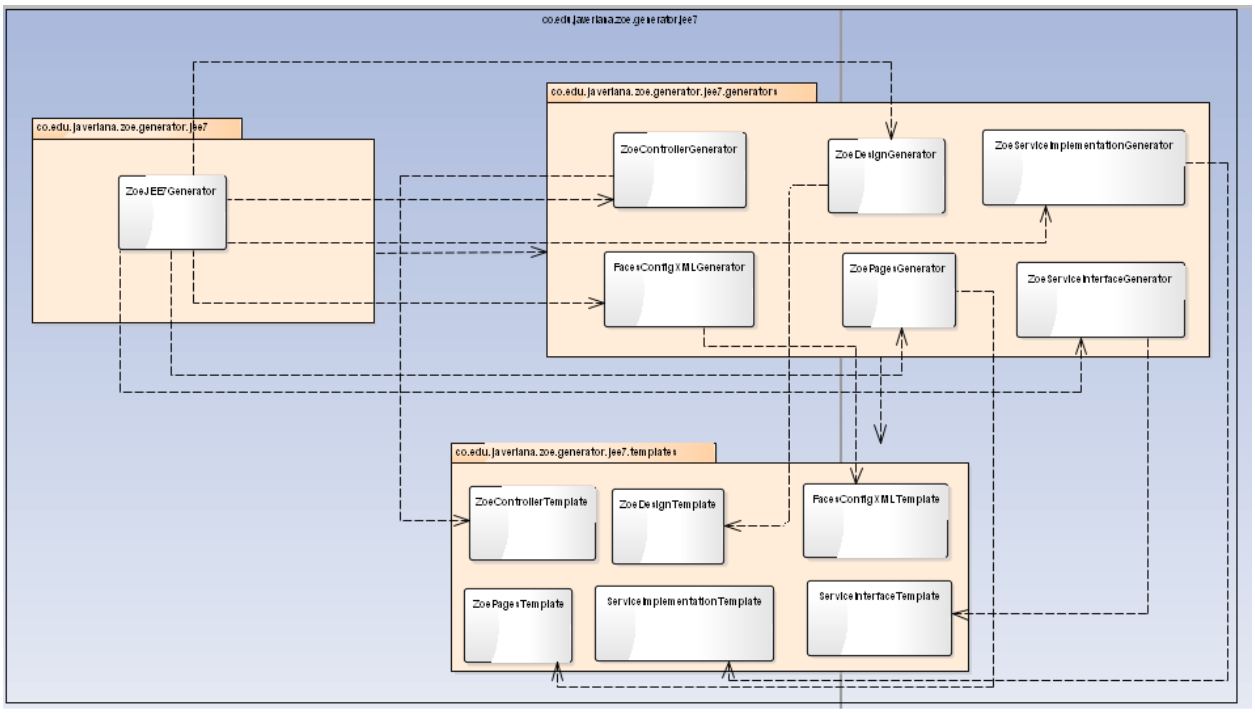

Figura 14: Vista lógica componente co.edu.javeriana.zoe.generator.jee7

En la figura 14 se pueden apreciar distintos paquetes que agrupan clases relacionadas entre sí correspondientes al componente co.edu.javeriana.zoe.generator.jee7. Entre sus principales clases se encuentran:

Clase	Descripción
co.edu.javeriana.zoe.generator.jee7	Es la clase con la que interactúa directamente el transformador, puesto que contiene referencias del conjunto de generadores incluidos en este componente, esta clase a su vez extiende de la clase GeneratorSuite que hace parte del lenguaje ISML.
ZoeControllerTemplate, **ZoeDesignTemplate,** **FacesConfigXMLTemplate,** **ZoePagesTemplate,** **ZoeServiceImplementationTemplate,** **ZoeServiceInterfaceTemplate.**	Cada una de estas clases heredan de una clase del lenguaje ISML SimpleTemplate que recibe una instancia de un componente determinado definido en el lenguaje que puede ser: Controller, Pages, Entity, Service, para el caso del transformador ZOE-GEN.

Tabla 10.Componentes co.edu.javeriana.zoe.generator.jee7

Cada una de estas clases está relacionada con las clases del paquete co.edu.javeriana.zoe.generator.jee7.generators. Dependiendo del tipo de instancia al que se refieren. Por ejemplo la clase ZoeControllerTemplate está relacionada con la clase ZoeControllerGenerator, para ser referenciada en la configuración del generador.

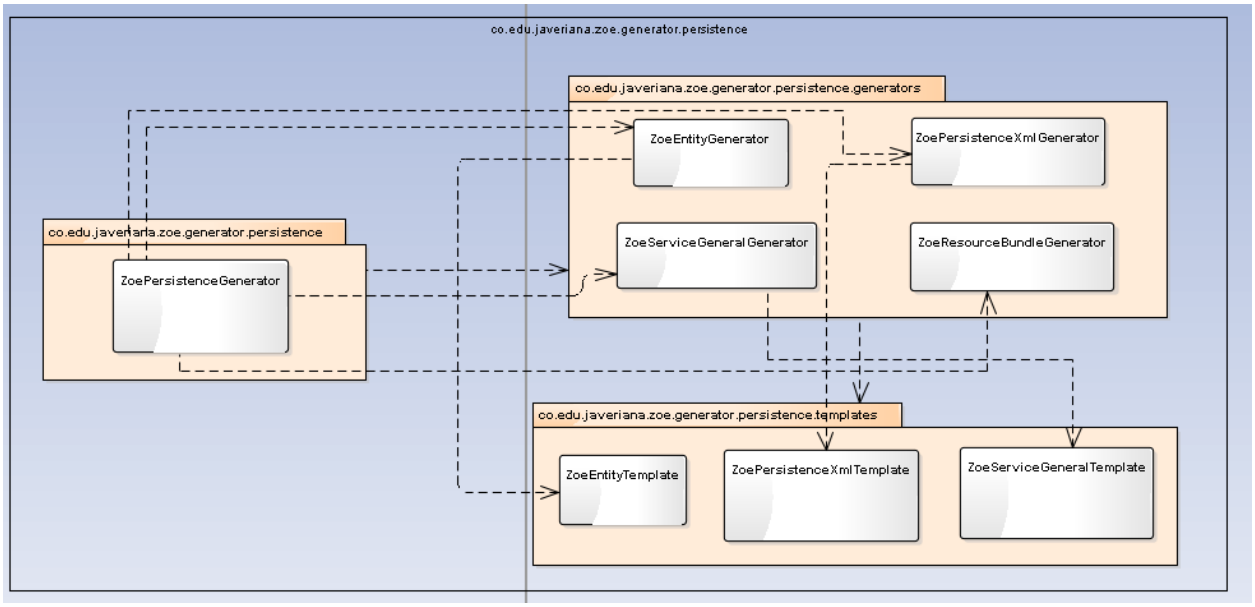

Figura 15: Vista lógica del componente: co.edu.javeriana.zoe.generator.persistence

En la figura 15 se puede apreciar la estructura del componente **co.edu.javeriana.zoe.generator.persistence,** el cual tiene tres paquetes: el paquete **co.edu.javeriana.zoe.generator.persistence.generators,co.edu.javeriana.zoe.generator.persistence** y el componente co.edu.javeriana.zoe.generator.persistence.templates.

Clase	Descripción
ZoePersistenceGenerator	Es la clase con la que interactúa directamente el transformador, puesto que contiene referencias del conjunto de generadores incluidos en este componente, esta clase a su vez extiende de la clase GeneratorSuite que hace parte del lenguaje ISML.
ZoeServiceGeneralTemplate, ZoeEntityTemplate, ZoePersistenceXmlTemplate.	Cada una de estas clases heredan de una clase del lenguaje ISML SimpleTemplate que recibe una instancia de un componente determinado definido en el lenguaje que puede ser: Entity y Service, para el caso del transformador ZOE-GEN.

Tabla 11. Clases del componente co.edu.javeriana.zoe.generator.persistence

Dependiendo del tipo de instancia al que se refieren. Por ejemplo la clase ZoeEntityTemplate está relacionada con la clase ZoeEntityGenerator, para ser referenciada en la configuración del generador.

6. Desarrollo del transformador

6.1 Metodología

Según se mencionó anteriormente la metodología para el desarrollo de este proyecto es una adaptación de scrum. Las principales razones del uso de un ciclo de desarrollo iterativo e incremental de tipo scrum para la ejecución de este proyecto son: las características del trasformador Zoe-Gen permiten desarrollar una base funcional mínima y sobre ella ir incrementando las funcionalidades o modificando el comportamiento o incluyendo nuevas características anteriormente implementadas. Las entregas frecuentes y continuas de los requerimientos implementados, de forma que puede disponer de una funcionalidad básica en un tiempo mínimo y a partir de ahí un incremento y mejora continua del sistema.

Fueron implementadas las prácticas de la metodología, llevando a cabo reuniones de manera semanal o incluso más periódica a manera de seguimiento. De esta manera se planearon las iteraciones con la construcción de un product backlog como se puede apreciar en la siguiente tabla.

ID	Nombre	Descripción	timac	Responsable	pender	Prioridad
2	Req01	El sistema debe generar una aplicación con entidades que contengan atributos, metodos de encapsulamiento, metodos equals, hash code.	30	John Olarte	-	Media
3	Req02	El sistema debe generar una aplicación con entidades que manejen herencia, constraints y relaciones de cardinalidad.	30	John Olarte	1	Media
4	Req03	El sistema debe generar una aplicación con un servicio generico de persistencia, con las opciones de crear, editar, eliminar, consultar ademas de filtrar y ordenar.	30	John Olarte	-	Media
5	Req04	El sistema debe generar controladores con atributos y metodos especificados en el modelo.	30	John Olarte	3	Media
6	Req05	El sistema debe generar archivos de configuración como faces-config.xml para garantizar la navegación.	30	John Olarte	3	Media
8	Req06	El sistema debe generar archivos de configuración como persistence.xml para garantizar el funcionamiento de la persistencia.	30	John Olarte - Jose, Jaime	-	Media

Tabla 12. Product Backlog

A cada requerimiento se le asigna una estimación en horas junto con un responsable y una prioridad. Por lo demás algunos requerimientos tienen dependencias entre sí por lo que también es un factor que se tomó en cuenta en el orden de implementación de los distintitos componentes.

6.2 Sprints realizados

En la siguiente tabla se encuentran detallado el spring backlog en donde se detallan los pormenores de los sprints realizados durante el desarrollo del transformador Zoe-Gen. En la primera se encuentra el nombre e identificador del requerimiento. La segunda columna representa una descripción breve de la funcionalidad que tiene que realizar el transformador. La segunda y tercera columna representa el originador del requerimiento que puede ser el transformador Zoe-Gen o el framework Midas como se explicó anteriormente. De igual manera se encuentra el responsable de la realización del requerimiento. La cuarta columna se refiere al estado de la tarea. Esta columna solamente puede tomar los siguientes valores: “Sin Comenzar”, “En progreso” o “Completa”. Las columnas subsiguientes representan el Sprint a que corresponde cada tarea respectivamente.

Nombre	Descripción	Originador	Responsable	Estado	Spring
Req01	El sistema debe generar una aplicación con entidades que contengan atributos, métodos de encapsulamiento, métodos equals, hash code.	Zoe-Gen	John Olarte	Completado	1
Req02	El sistema debe generar una aplicación con entidades que manejen herencia, constraints y relaciones de cardinalidad.	Zoe-Gen	John Olarte	Completado	1
Req03	El sistema debe generar una aplicación con un servicio genérico de persistencia	Zoe-Gen	John Olarte	Completado	1
Req04	El sistema debe generar controladores con atributos y métodos especificados en el modelo.	Zoe-Gen	John Olarte	Completado	2
Req05	El sistema debe generar archivos de configuración como faces-config.xml para garantizar la navegación.	Zoe-Gen	John Olarte	Completado	2
Req06	El sistema debe generar archivos de configuración como persistence.xml para garantizar el funcionamiento de la persistencia.	Zoe-Gen	John Olarte	Completado	2
Req07	El sistema debe generar páginas con elementos gráficos como botones.	Zoe-Gen	John Olarte	Completado	3
Req08	El sistema debe generar páginas con elementos gráficos como cajas de texto.	Zoe-Gen	John Olarte	Completado	3
Req09	El sistema debe generar páginas con elementos gráficos como labels.	Zoe-Gen	John Olarte	Completado	3
Req10	El sistema debe generar páginas con elementos gráficos como spinners.	Zoe-Gen	John Olarte	Completado	3
Req11	El sistema debe generar páginas con elementos gráficos como radio buttons.	Zoe-Gen	John Olarte	Completado	3
Req13	El sistema debe generar páginas con elementos gráficos como data tables.	Zoe-Gen	John Olarte	Completado	3
Req14	El sistema debe generar páginas con elementos gráficos como menús.	Zoe-Gen	John Olarte	Completado	3
Req15	El sistema debe generar páginas con elementos gráficos como mapas.	Zoe-Gen	John Olarte	Completado	4
Req16	El sistema debe generar páginas con elementos gráficos como calendarios.	Zoe-Gen	John Olarte	Completado	3
Req18	El sistema debe generar páginas con elementos gráficos como mensajes.	Zoe-Gen	John Olarte	Completado	4
Req19	El sistema debe generar páginas con elementos gráficos como contenedores de imágenes.	Zoe-Gen	John Olarte	Completado	4
Req20	El sistema debe permitir parametrizar los componentes gráficos para cambiar sus propiedades.	MiDAS	John Olarte	Completado	4
Req21	El sistema debe permitir generar servicios personalizados a partir de un modelo.	MiDAS	John Olarte	Completado	5
Req22	El sistema debe permitir cambiar el orden de los componentes desplegados en una página.	MiDAS	John Olarte	Completado	5
Req23	El sistema debe permitir cambiar el color o forma de los componentes desplegados en una página.	MiDAS	John Olarte	Completado	6
Req24	El sistema debe permitir generar servicios que apoyen este proceso de decisión.	MiDAS	John Olarte	Completado	6

Tabla 13. Sprint backlog.

6.3 Inconvenientes y resolución

En el proceso de implementación del transformador se encontraron inconvenientes al momento de realizar el mapeo de modelos independientes de plataforma en ISML a código en Java. En primer lugar al momento de especificar las reglas de transformación para la construcción de requerimientos

como los correspondientes a la generación de servicios, entre los cuales se encontraban las opciones de generar implementaciones distintas de un mismo servicio por medio de qualifiers, lo cual ocasiono inconvenientes con el administrador de persistencia, por lo tanto se decidió implementar una interfaz hacia el servicio de persistencia por cada una de las entidades que lo requerían.
Adicionalmente cuando se desarrolló el requerimiento correspondiente a la implementación de queries en jpql, inicialmente se contempló utilizar librerías con licenciamiento opensource para la generación de queries como QueryDSL, no obstante tras realizar varias pruebas de concepto y la dificultad de contemplar distintos escenarios de generación, dificultaron el proceso de construcción de los transformadores y al contemplar el impacto que tendrían con el modelamiento, se decidió implementar un servicio de que se encargara de esta funcionalidad.

7. Evaluación de los resultados

Los casos de prueba están especificados como pruebas técnicas del sistema que garantizan el correcto funcionamiento de los transformadores. Estos se especifican para garantizar el correcto funcionamiento de los elementos más comunes de las aplicaciones web y que corresponden a un modelado en un modelo ISML. Entre estos se encuentran controladores, servicios, páginas y elementos gráficos para estas últimas.

De la misma manera se validan los requerimientos correspondientes al framework MIDAS (J. Bocanegra, J. Pavlich-Mariscal, and A. Carrillo-Ramos, 2015). Estos favorecen la generación de aplicaciones adaptativas, como la generación de queries en JPQL y elementos gráficos parametrizables. Los requerimientos fueron probados mediante el uso del framework JUnit, evaluando un valor de entrada y un resultado, retornando un valor positivo o negativo de la prueba dependiendo del código generado.

7.1 Caso de estudio

Los elementos de la aplicación tales como: servicios, páginas, controladores y entidades, serán tomados de una aplicación de referencia construida en JavaEE7. La aplicación de referencia que contiene estos elementos es la aplicación Prenat que consiste en un sistema adaptativo de control prenatal. La aplicación fue desarrollada en el marco del curso de Adaptación de la Información durante el segundo semestre de 2014, ofrecido en la Pontificia Universidad Javeriana (José Bocanegra, John Olarte, Claudio Hernández, Ángela Carrillo, 2014). La aplicación es un sistema adaptativo de control prenatal que usa mecanismos de adaptación con el fin para ofrecerle a la

madre gestante y a su grupo familiar consejos sobre la preparación del parto, recomendarle una dieta y sugerirle tiendas de maternidad.
En Prenat se han definido tres servicios adaptativos principales que pueden traducirse como requerimientos funcionales: (i) notificar consejos de preparación para el parto; (ii) indicar las tiendas especializadas en maternidad; y (iii) sugerir una dieta (José Bocanegra, John Olarte, Claudio Hernández, Ángela Carrillo, 2014). Para efectos de esta validación, no se modelaran los perfiles de contexto y usuario como se realizó en el desarrollo de esta aplicación, si no que se contara con la información de estos perfiles a nivel de un modelamiento de entidades y de datos precargados en la base de datos.

En este orden de ideas se tomaran del perfil de usuario los datos acerca de los distintos tipos de dietas asociados a una patología y preferencia, a su vez detallada de los gustos. Adicionalmente de los perfiles de contexto solo se tomaran los datos correspondientes a la ubicación geográfica y se completaran con información sobre las tiendas y hospitales respectivamente.

7.2 Resultados de evaluación

A continuación se presentan un par de ejemplos de los resultados de la validación realizada y ejemplos de los casos de prueba que se realizaron, para consultar el total de las pruebas realizadas se puede consultar el documento anexo informe de pruebas. En uno de los casos se realiza el modelado de la entidad Dieta en lenguaje ISML, se tiene una entidad modelo generada con los atributos y métodos necesarios. Esta entidad hace parte de la aplicación de referencia Prenat. Mediante el framework Junit se verifica que la generación sea igual a la de la implementación de referencia.

No. caso de prueba	1
Proyecto	ZOE-GEN
Modulo	Requerimientos de entidades N0. 1,2,3,4,5,6.
Fecha de entrega	02/05/2016
Bases de datos	N/A

Datos de entrada	El modelado de la entidad Dieta, con los atributos de tipo String desayuno, almuerzo, cena, merienda y de tipo Integer el atributo patología.
	
	`package co.edu.javeriana.entities;`
	
	`entity Dieta {`
	
	`String desayuno;`
	`String almuerzo;`
	`String cena;`
	`String merienda;`
	`Integer patologia;`
	
	`}`

Salida Esperada	Entidad de referencia para generación: La entidad de referencia implementada en Java contiene la declaración de la entidad, los atributos de tipo String desayuno, almuerzo, cena, merienda y de tipo Integer el atributo patología, con sus respectivos comentarios y métodos de encapsulamiento.
	
	`ackage test;`
	
	`mport javax.persistence.*;`
	`mport javax.validation.constraints.*;`
	`mport java.io.Serializable;`
	`mport javax.xml.bind.annotation.XmlRootElement;`
	
	`XmlRootElement`
	`Entity`
	
	`ublic class Dieta implements Serializable {`
	
	`/**`
	`* The serialVersionUID`
	`*/`
	`private static final long serialVersionUID = 1L;`
	
	`/**`
	`* The unique id for the entity`
	`*/`
	`private Long id = null;`
	
	`/**`
	`* The desayuno for the Dieta`
	`*/`
	`private String desayuno;`
	
	`/**`
	`* The almuerzo for the Dieta`
	`*/`
	`private String almuerzo;`

	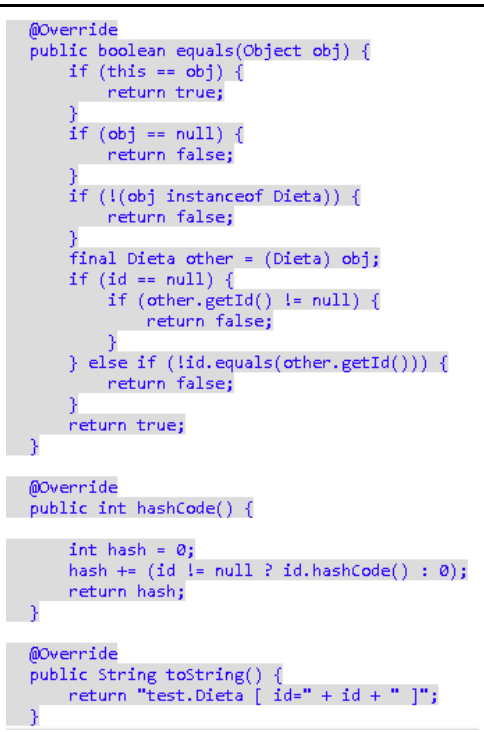
Tiempo de ejecución	1 min

Tabla 14. Tabla de resultados del caso de prueba.

También se realizó la validación de los requerimientos asociados a continuación corresponden a elementos que el transformador ZOE-GEN incorpora para facilitar la generación de aplicaciones adaptativas desde modelos ISML, en el marco del proyecto MiDAS. Estos requerimientos se obtuvieron a partir de reuniones con expertos del proyecto mencionado y a partir de una priorización, se escogieron cuales implementar. En este caso se realiza el modelado de una página en lenguaje ISML con un componente gráfico que genera la implementación de un mapa, con una lista de lugares definidos. Mediante el framework Junit se verifica que la generación sea igual a la de la implementación de referencia.

No. caso de prueba	4
Proyecto	ZOE-GEN
Modulo	Requerimientos de entidades N0. 5-16.
Fecha de entrega	02/05/2016
Bases de datos	N/A

Datos de entrada	Modelado de la página RestaurantMapView, con elementos gráfico GMap. Adicionalmente la página tiene parámetros de tipo Collection de Lugares. Se especifica que la funcionalidad de la página es soportada por el controlador DietaManager.
	`package co.edu.javeriana;`
	`import co.edu.javeriana.prenat.resource.*;`
	`import co.edu.javeriana.entities.* ;`
	`page RestaurantMapView( List<Lugares> model, MapModel simpleModel) controlledBy DietaManager {`
	`Form {`
	`Panel("Panel") {`
	`GMap("4.627447, -74.063842","15","HYBRID", simpleModel );`
	`}`
	`}`
	`}`

Salida esperada	La página modelo en Prime Faces 5.1, contiene las etiquetas de los elementos gráficos: form, paneles y gmap. En esta las funcionalidades de la página son soportadas por el controlador DietaManager.
	`<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">`
	`<html xmlns="http://www.w3.org/1999/xhtml"`
	`xmlns:ui="http://java.sun.com/jsf/facelets"`
	`xmlns:f="http://java.sun.com/jsf/core"`
	`xmlns:h="http://java.sun.com/jsf/html"`
	`xmlns:c="http://java.sun.com/jsp/jstl/core"`
	`xmlns:p="http://primefaces.org/ui">`
	`<ui:define name="metadata">`
	`<f:metadata>`
	`<f:event type="preRenderView" listener="#{dietaManager.init()}" />`
	`</f:metadata>`
	`</ui:define>`
	`<ui:composition template="/template.xhtml">`
	`<ui:define name="content">`
	`<h:form id= "form1">`
	`<script src="http://maps.google.com/maps/api/js?sensor=true"></script>`
	`<p:draggable for="panel2" />`
	`<p:panel id= "panel2" >`
	`<p:gmap id= "gMap3" center="4.627447, -74.063842" zoom="15" type="HYBRID"`
	`model= "#{dietaManager.simpleModel}"`
	`style="width:100%;height:400px" />`
	`</p:panel>`
	`</h:form>`
	`</ui:define>`
	`</ui:composition>`
	`</html>`

Tiempo de ejecución	1 min

Tabla 15. Tabla de resultados del caso de prueba.

Como se puede apreciar en las tablas 3 y 4 se han aplicado los formatos con unos datos de entrada y salida esperados para el tipo de elemento que se quiera generar. Cada prueba genera código ejecutable en las herramientas definidas y se constituye como la validación de la generación con base en una aplicación de referencia en Java EE.

Número de caso de prueba	Descripción	Aprobada	No aprobada
1	Generación de entidades.	X	
2	Generación de controladores	X	
3	Generación de paginas.	X	
4	Generación de requerimientos para Midas.	X	
5	Generación de servicios.	X	
6	Generación de jpql.	X	

Tabla 16. Tabla de resultados de los casos de prueba.

La tabla 5 corresponde a la consolidación de los resultados de validación correspondientes a los casos de prueba ejecutados junto con sus resultados. Con los casos de prueba ejecutados se valida la correcta generación de los componentes modelados y que responden a los requerimientos del transformador Zoe-Gen como son la construcción de entidades, servicios, controladores y elementos gráficos y archivos de configuración. En cuanto a los requerimientos asociados con el framework MiDAS, se comprobó la correcta generación de componentes como los mapas, la generación de queries con jpql a partir de un servicio, generación de plantillas con las páginas denotadas con sentencias especiales.

8. Conclusiones, recomendaciones y trabajo futuro

8.1 Conclusiones

Los generadores cierran la brecha entre el modelo y el mundo del código. El generador específica como la información es extraída de los modelos y transformada en código. En términos generales, cada símbolo del modelo produce cierta especificación de código con sus valores (Kelly Tolvanen, 2008). En este sentido la construcción de transformadores como Zoe-Gen, facilita los procesos de desarrollo de software, sin necesidad de ser experto en tecnologías específicas, lo cual permite que los ingenieros se dediquen a cumplir otros elementos de arquitecturas y a modelar el negocio.

El generador en sí mismo es usualmente invisible a los modeladores y la construcción y modificación del generador es hecha por pocos programadores expertos (Kelly Tolvanen, 2008). El resultado de la construcción de este transformador, cumple con los objetivos propuestos y con la anterior información. La simplicidad de modelamiento hace invisible la generación del código y facilita que este, se encuentre disponible prácticamente para ejecutar.

La inclusión de requerimientos asociados a implementar aplicaciones adaptativas, ayudarán a aplicar los principios de MDE a un framework más robusto como MIDAS, que permitirá incorporar el desarrollo basado en modelos a todo el ciclo de vida de la aplicación.

8.2 Trabajo Futuro

En primer lugar, se espera convertir los transformadores en un plugin para que no sea necesario abrir la instancia de eclipse. También es importante implementar un servicio o componente genérico de seguridad que se aplique a todos los modelos construidos en ISML y que se integre a la mayoría de servidores de aplicaciones.

A manera de complemento sería beneficioso, realizar uno o varios arquetipos de proyectos para generar modelos en ISML, que contengan distintos patrones de diseño para que se puedan adaptar más fácilmente a distintos tipos de necesidades.

Incorporar bien sea al lenguaje o a los transformadores que se construyan, un mecanismo para la generación dinámica de código, que permita generar únicamente los segmentos necesarios.

Finalmente realizar transformadores que contengan mecanismos de integración con distintos componentes o plataformas de otras tecnologías.

9. Listado de Anexos

- Especificación de requerimientos del transformador
- Especificación del diseño del transformador
- Manual de usuario e instalación del transformador
- Reporte de pruebas de aceptación del transformador

10. Referencias bibliográficas

Cuesta M. albeiro, lópez t. Marcelo. Joyanes a. Luis. (2009). COMPARATIVO DE HERRAMIENTAS MDA. (andromda, arcstYler, oPtImalJ). *Revista Vector.*, pag 50-58.

L. Rising y N. S. Janoff. (2000). The Scrum software development process for small teams. *IEEE Software, vol. 17, n. 4*, pp. 26-32.

Aagedal, Jan; Mohagheghi, Parastoo. (2007). Evaluating Quality in Model-Driven Engineering. *MISE '07 Proceedings of the International Workshop on Modeling in Software Engineering.* Washingtong D.C.

Ahmad, M. O., Markkula, J., & Oivo, M. (2013, September). Kanban in software development: A systematic literature review. In Software Engineering and Advanced Applications (SEAA). *Conference on EUROMICRO2013 39th* (págs. pp. 9-19). IEEE.

B. Steen, L. F. Pires, y M.E. Iacob. (2010). Automatic generation of optimal business processes from business rules. *14th IEEE International Enterprise Distributed Object Computing Conference Workshops (EDOCW)* (págs. 117-126). IEEE.

Beecham, S.,Hall, T., & Rainer, A. (2003). Software process improvement problems in twelve software companies: An empirical analysis. *Empirical software engineering*, pp. 7-42. .

Bettini, L. (2013). Implementing Domain-Specific Languages with Xtext and Xtend. Packt Publishing.

Calic, Tihomir; Sergiu Dascalu; Dwight Egbert. (2008). Tools for MDA software development: Evaluation criteria and set of desirable features. *Fifth International Conference on Information Technology: New Generations.* ITNG 2008.

Fleurey, F., y Solberg, A. (2009). A domain specific modeling language supporting specification, simulation and execution of dynamic adaptive systems. . *Model Driven Engineering Languages and Systems* (págs. pp. 606- 621). Springer.

France, R.; Rumpe, B. (2007). Model-driven development of complex software: A research roadmap. *IEEE Computer Society*, pp. 37-54.

Frankel David S. (2003). Model Driven Architecture Applying MDA to Enterprise Computing. Wiley Publishing.

Franky, María Consuelo, Pavlich-Mariscal Jaime A. (2014). A Method to Achieve Automation in the Development of Web-Based Software Projects. *The Ninth International Conference on Internet and Web Applications and Services ICIW.*

Group, O. M. (2014). *MDA - The Architecture of Choice for a Changing World*. Obtenido de http://www.omg.org/mda/

Hashimi H, Hafez A; Beraka M; (2012). A Novel View of Risk Management in Software Development Life Cycle. *Proceedings of the 2012 12th International Symposium on Pervasive Systems, Algorithms and Networks.*

Hurtado Alegría, J. A., Bastarrica, M. C., Quispe, A., & Ochoa, S. F. (2011). An MDE approach to software process tailoring. . *In Proceedings of the 2011 International Conference on Software and Systems Process* (págs. pp. 43-52). ACM.

J. Bocanegra, J. Pavlich-Mariscal, and A. Carrillo-Ramos. (2015.).A midas: A model-driven approach for adaptive software. . *In Conference on Web. Information Systems and Technologies - WEBIST.*

John Hutchinson; Jon Whittle;Mark Rouncefield; Steinar Kristoffersen. (2011.). Empirical assessment of MDE in industry. *ICSE '11Proceedings of the 33rd International Conference on Software Engineering.* New York.

José Bocanegra, Jaime Pavlich-Mariscal, Angela Carrillo-Ramos. (2014). DMLAS: A Domain-Specific Language for Designing Adaptive Systems. Bogotá, D.C.: Pontificia Universidad Javeriana.

Kelly, S., & Tolvanen, J. P. (2008). Domain-specific modeling: enabling full code generation. John Wiley & Sons.

Kent, S. (2002). Model driven engineering. In Integrated formal methods. *Springer Berlin Heidelberg*, 286-298.

Kleppe, A. J., Warmer, J., Bast, W. (2003). En *The Model Driven Architecture: MDA Explained.*

Kriouile, A., Gadi, T., & Balouki, Y. (2013). CIM to PIM Transformation: A criteria Based Evaluation. *International Journal of Computer Technology and Applications*, 616.

Larman, C. (2004). Agile and iterative development: a manager's guide. *Addison-Wesley Professional.*

Lucredio, Daniel, E. Santana de Almeida y R. P. Fortes. (2012). An investigation on the impact of MDE on software reuse. *Sixth Brazilian Symposium on Software Components Architectures and Reuse (SBCARS).*

María Consuelo Franky, J. A.-M. (2015). ISML: Un Lenguaje y un Ambiente MDE para modelar y generar aplicaciones web con integración de componentes existentes. *10CCC.*

Mellor Stephen J, Scott Stephen J, Uhl Axel, Weise Dirk. . (2004). MDA Distilled: Principles of Model-Driven Architecture. . Addison-Wesley. .

Nasir, M. (2006). A Survey of Software Estimation Techniques and Project Planning Practices. *Seventh ACIS International Conference.*

Oiwa, Y., Masuhara, H., & Yonezawa, A. (2001, March). DynJava: Type safe dynamic code generation in Java. *The 3rd JSSST Workshop on Programming and Programming Languages .*

OMG Meta Object Facility (MOF) Core Specification Version 2.4.1. (Junio de 2013). Obtenido de http://www.omg.org/spec/MOF/2.4.1/

Parastoo Mohagheghi, Vegard Dehlen. (2008). Where Is the Proof? - A Review of Experiences from Applying MDE in Industry. *ECMDA-FA '08 Proceedings of the 4th European conference on Model Driven Architecture: Foundations and Applications.* Berlin: Springer.

Reference Documentation. (23 de Abril de 2016). Obtenido de http://www.eclipse.org/xtend/documentation/index.html

Rothenberger, M.A.; Dooley, K.J.; Kulkarni, U.R.; Nada, N. (2003). «Strategies for software reuse: a principal component analysis of reuse practices». *Software Engineering, IEEE Transactions, vol. 29, nº 9*, pp. 825,837.

SCULPTOR. (2016). *Sculptor documentation*. Obtenido de http://sculptorgenerator.org/documentation/overview

Sommerville, I., & Galipienso, M. I. A. (2005). Ingeniería del software. . En I. &. Sommerville.

Steinberg, D. (March de 2005-2008). Fundamentals of the Eclipse Modeling Framework. Toronto, Canada.

Warmer Jos, Kleppe Anneke. (2003). Object Constraint Language, The: Getting Your Models Ready for MDA. Second Edition. Pearson Education.