UCLV
Universidad Central
"Marta Abreu" de Las Villas

MFC
Facultad de Matemática
Física y Computación

TRABAJO DE DIPLOMA

Título "Desarrollo del módulo *PIM-PSM* versión 5.0 de la herramienta *jMDA*"

Autores : Eduardo Ariel Orozco Alvarez

Tutores : Dr. Rosendo Moreno Rodríguez.

Santa Clara, junio 2018

DEDICATORIA

A mi madre Clara Irene

Por haberme apoyado incondicionalmente en todo momento, por sus consejos, por la motivación constante que me ha permitido ser una mejor persona, pero más que nada, por su amor y cariño incondicional.

A mi padre Oscar

A mi padre quien me ha apoyado desde la distancia en todo momento me ha alentado en los momentos difíciles a seguir adelante y ha sido mi ejemplo y mi guía para lograr mis metas.

AGRADECIMIENTOS

Quiero agradecer a aquellas personas que compartieron sus conocimientos conmigo para hacer posible la realización de esta tesis. Especialmente agradezco a mi tutor el Dr. Rosendo Moreno Rodríguez por brindarme su apoyo en todo momento.

A todos mis amigos especialmente a Osviel Peraza, Carlos Miguel García, Luis Alberto Jiménez, Yanet González Pérez por sus críticas durante el proceso de implementación de la herramienta y por estar siempre en los buenos y malos momentos, a todos en general muchas gracias.

A todas las personas que han hecho posible de una forma o de otra que fuera posible la realización de este momento.

Resumen

El desarrollo de un software es un proceso complejo y difícil de gestionar en el que intervienen múltiples elementos. En ocasiones el ciclo de vida de este producto se ve permeado de disímiles problemáticas que lo afectan, hasta no responder a las necesidades identificadas.

El *Object Management Group (OMG, por sus siglas en inglés)*, es un consorcio informático que está encaminado potenciar el desarrollo de aplicaciones orientadas a objetos. Desde su surgimiento le presta especial atención al problema de interoperabilidad e integración de software, definiendo numerosas especificaciones y estándares. En el 2001, *OMG* establece el *framework "Model Driven Architect" (MDA, por sus siglas en inglés)*, como arquitectura para el desarrollo de aplicaciones. En este paradigma de desarrollo de software, denominado Ingeniería de Modelos o Desarrollo Basado en Modelos, los modelos guían todo el proceso.

La versión 5.0 del módulo PIM-PSM de la herramienta "jMDA", constituye un perfeccionamiento de las anteriores, centrado en la implementación de los diagramas de casos de uso, actividades, estado y secuencia al nivel de "*Platform Independent Model"* (*PIM por sus siglas en inglés*); así como la transformación asistida por computadoras de dichos diagramas y la creación de los diagramas de artefacto y despliegue al *"Platform Specific Model" (PSM, por sus siglas en inglés)*. El desarrollo de la investigación no sólo se centra en la implementación de la solución, sino también en la generación de la documentación de la metodología de desarrollo de software utilizada, así como la de uso de la herramienta y los análisis de factibilidad asociados al desarrollo de la solución propuesta.

Abstract

The development of a software is a complex and difficult process to manage in which multiple elements intervene. Sometimes the life cycle of this product is permeated by dissimilar problems that affect it, until it does not respond to the identified needs.

The Object Management Group (OMG) is a computer consortium that aims to enhance the development of object-oriented applications. From its emergence, it pays special attention to the problem of interoperability and software integration, defining numerous specifications and standards. In 2001, OMG establishes the framework "Model Driven Architect" (MDA), as architecture for the development of applications. In this paradigm of software development, called Model Engineering or Model-Based Development, the models guide the entire process.

Version 5.0 of the PIM-PSM module of the "jMDA" tool is a refinement of the previous ones, focused on the implementation of the diagrams of use cases, activities, status and sequence at the level of "Platform Independent Model" (PIM by its acronyms in English); as well as the computer-assisted transformation of these diagrams and the creation of artifact diagrams and deployment to the "Platform Specific Model" (PSM). The development of the research is not only focused on the implementation of the solution, but also in the generation of documentation of the software development methodology used, as well as the use of the tool and the feasibility analysis associated with the development of The proposed solution.

Tabla de contenido

Introducción

En la actualidad, el desarrollo de software se enfrenta a continuos cambios en las tecnologías de implementación, lo que implica realizar esfuerzos importantes tanto en el diseño de la aplicación, para integrar las diferentes tecnologías que incorpora, así como en su mantenimiento. Todo esto para adaptar la aplicación a cambios en los requisitos y en las tecnologías de implementación.

La interoperabilidad y la integración son elementos cada vez más necesarios en los sistemas de software que se construyen (Medicine, 2004). Los cambios constantes en el negocio y en las tecnologías de implementación exigen de esfuerzos constantes en la búsqueda de mejores modos de diseñar las aplicaciones, de integrarlas y adaptarlas de manera continua a los nuevos requisitos que surjan. (Pérez, 2015)

Uno de los principios básicos de la ingeniería de software es la abstracción, para separar lo esencial de lo no esencial en términos de negocio lo esencial es la funcionalidad y lo no esencial es la plataforma tecnológica estas abstracciones no las proveen los modelos. El modelado y la transformación de modelos, hasta el nivel de abstracción requerido, constituye el núcleo del desarrollo dirigido por modelos.

Es por eso que el *Object Management Group (OMG por sus siglas en inglés)* en búsqueda de una alternativa para resolver estos problemas en el año 2001 establece el *framework* Arquitectura Dirigida por Modelos *(MDA, por sus siglas en inglés)* como arquitectura para el desarrollo de aplicaciones. Creando un nuevo paradigma en el que los modelos constituyen la guía de todo el proceso de desarrollo de software. Con este *framework OMG* demuestra que el enfoque de modelado es una forma potente de especificar sistemas (Pereira, 2008), permitiendo que se obtengan beneficios importantes en aspectos fundamentales como son la productividad, la portabilidad, la interoperabilidad y el mantenimiento

A pesar de las ventajas de esta nueva filosofía de trabajo resulta imprescindible la existencia de herramientas que le den soporte. En este contexto el Centro de Investigaciones de Informática de la UCLV trabaja en una línea de investigación encaminada al “desarrollo de métodos y tecnologías avanzadas de Ingeniería de Software para el desarrollo de Sistemas de Información” desde el año 2010 se propone desarrollar una herramienta que implemente el paradigma *MDA*. En ese mismo año se comienza a desarrollar la primera versión de la herramienta jMDA, con funcionalidades

orientadas a la elaboración y edición de diferentes diagramas (caso de uso, clases y actividades). Significando un gran avance en esta línea de investigación.

A pesar de los resultados que se han ido obteniendo existen carencias en cuanto a la representación de un conjunto de diagramas y funcionalidades que aún no han sido implementadas. En el modelo *PSM* es necesario incorporar los diagramas de artefacto y despliegue; así como permitir la conversión asistida de los diagramas de actividad, secuencia, estado y caso de uso del "*Platform Independent Model" (PIM* por sus siglas en inglés*),* al "*Platform Specific Model*" *(PSM,* por sus siglas en inglés*).*

Todo lo anteriormente abordado circunscribe el problema de investigación a que la existencia de una herramienta soberana que cumpla con los principios de MDA y apoye a la industria nacional de la informática es de gran importancia en la actualidad. El proceso de desarrollo de la herramienta jMDA inició en el año 2010 en la Universidad Central "Marta Abreu" de Las Villas, sin embargo las versiones actuales no brindan cobertura para todos los diagramas UML requeridos, carece de la fase de conversión al PSM y la interfaz de usuario no facilita el proceso de modelado.

Desde hace algún tiempo se han desarrollado en el mundo algunas herramientas de apoyo a la Ingeniería de Software que se adhieren al paradigma *MDA* y que cuentan con licencias de software libre. Cabe mencionar el proyecto *AndroMDA,* fundado en el 2003 según (Bollati & Vara, 2012) y *GMF* [1] de Eclipse, que pueden servir de base al desarrollo de futuras aplicaciones con similar fin. Sin embargo, aún no se pueden considerar herramientas puras *MDA* pues no incluyen algunas fases y conversiones, o se apoyan en otras herramientas como *MagicDraw* para el diagramado que son propietarias.(Bollati & Vara, 2012).

La industria de software nacional no cuenta con ninguna herramienta profesional que implemente el paradigma *MDA* que supere las limitantes con que cuenta las existentes en la actualidad, ya sea el pago de una licencia en el caso de las propietarias o limitantes de orden técnico en las de código abierto. Por lo que significaría un aporte invaluable al proceso de desarrollo de software en Cuba la implementación de una herramienta con estas características.

Todo lo anteriormente mencionado lleva a establecer el siguiente **problema de investigación**:

La herramienta prototípica *CASE jMDA* desarrollada hasta su versión 4.0 en el módulo *PIM-PSM* presenta insuficiencias en la interacción del usuario con la misma desde el

[1] ***GMF*** *"Graphical Modeling Framework"* es un acrónimo que nombra a un marco de trabajo de Eclipse.

punto de vista gráfico y conceptual, así como problemas en la transformación asistida del *PIM* al *PSM* con uso de *Java*. Específicamente esa versión logró una mejora en el modelado de Diagramas de Clases en el PIM y su conversión al PSM en arquitectura Cliente-Servidor para apoyar el modelado de problemas del tipo Sistemas de Información, pero aun en ese resultado carece de algunos aspectos profesionales de una herramienta de este tipo.

Para solucionar este problema de investigación se define como **objetivo general**:

Desarrollar una nueva versión del módulo *PIM-PSM* de la herramienta *CASE jMDA* que soporte una mejor y más amplia representación gráfica tanto en el *PIM* como en el *PSM*, facilitando el proceso de modelado asistido al usuario y permitiendo la transformación automática del *PIM* al *PSM* con uso de *Java* de varios diagramas.

En consonancia con el objetivo general anteriormente mencionado se definen los siguientes **objetivos específicos**:

1. Analizar los referentes teórico-metodológicos para el diseño e implementación de la versión 5.0 de jMDA PIM-PSM, así como el estado actual de esta herramienta.
2. Diseñar la versión 5.0 de de jMDA PIM-PSM fundamentando las reglas de transformación adicionales a implementar.
3. Implementar la versión 5.0 de jMDA PIM-PSM, lo que incluye la creación de la lógica de transformaciones propuesta para pasar de un modelo al otro de varios diagramas de UML.

Estos objetivos constituyen el alcance de la investigación que se desarrolla y generan las siguientes preguntas técnicas:

Preguntas de Investigación:

- ¿Qué posible marco de trabajo para la programación en java se puede utilizar, de modo que facilite el proceso de codificación y que mejore la calidad gráfica de la aplicación obtenida?
- ¿Qué mecanismo de transformación de modelo utilizar para garantizar la correcta conversión del PIM al PSM?
- ¿Cuáles criterios utilizar para realizar una valoración adecuada de las anteriores versiones?

Hipótesis:

El perfeccionamiento de la herramienta jMDA es un paso de avance al desarrollo de un producto nacional que implemente el paradigma MDA, posibilitando la mejora en la

usabilidad y transformación iterativa de diagramas UML con vistas a incrementar la productividad, interoperabilidad, y calidad de las empresas que desarrollan aplicaciones.

Estructura de la tesis:

Para su mejor comprensión el informe escrito del presente trabajo de investigación (Tesis) se divide en 4 capítulos cuyos contenidos se detallan a continuación:

1. El capítulo 1 denominado **"Análisis del marco teórico conceptual sobre el paradigma MDA"** incluye la revisión del estado del arte mediante el análisis bibliográfico de la temática correspondiente a la teoría y práctica del MDA. También incluye un análisis crítico de las versiones desarrolladas hasta ahora de este módulo PIM-PSM de la herramienta jMDA.
2. El capítulo 2 denominado **"Análisis y diseño del Módulo PIM-PSM 5.0 de la herramienta CASE jMDA"** incluye la descripción de los caso de uso y de los requisitos funcionales y no funcionales de la aplicación además se tratan temas relacionados con la arquitectura del sistema a implementar, se modelan los diagramas que permitan comprender su funcionamiento de forma sencilla. También se definen las reglas de transformación para todos los tipos de diagramas. Se da explicación a las tecnologías utilizadas en la implementación del módulo correspondiente y se describe el proceso de implementación. Además, se describe cómo se lleva a cabo el tratamiento de errores.
3. El capítulo 3 denominado **"Descripción del uso del módulo PIM-PSM 5.0 de la herramienta CASE jMDA".** Sirve para explicar brevemente cómo usar la aplicación además de ejemplos de modelado de todos los diagramas en PIM y su transformación al modelo PSM.
4. El capítulo denominado **"Diseño de pruebas y análisis de factibilidad del módulo PIM-PSM 5.0 de la herramienta CASE jMDA".** Incluye las pruebas de caja negra de la aplicación además de la estimación de la misma.

Capítulo 1: "Análisis del marco teórico conceptual sobre el paradigma MDA"

Ante el gran reto que tienen las empresas desarrolladoras de software de mejorar el desempeño para obtener un mayor índice de ganancias, se plantea la necesidad de aplicar el reúso en un alto nivel de abstracción. Cuba no está ajena a esta situación y trabaja fuertemente en potenciar la industria de software. Algunos esfuerzos se evidencian en la formación de personal calificado y la creación de organizaciones productoras de software tales como: GET en 1995, DeSoft en 1995, TranSoft en 1996, Universidad de la Ciencias Informáticas (UCI) en 2002 y Datys en el 2006.(Lazo Alvarado et al., 2016).

Los procesos de desarrollo de software son complejos, en (Lazo Alvarado et al., 2016) se realiza un análisis de un conjunto de problemáticas que están presentes en la empresas productoras de software a lo largo de su implementación. En este punto, *MDA* definida por la *OMG* proponen pensar tempranamente en formas de reúso basadas en la manera como se hace la abstracción de un problema y se especifica una propuesta de solución.

Para los desarrolladores es un gran reto definir una estructura en la que el actor principal sean los modelos; los que definen la lógica del negocio. Estos permiten por medio de mecanismos predefinidos y herramientas, la transformación progresiva hasta llegar al producto final; elementos característicos de *MDA*. En los procesos de desarrollo donde está presente el paradigma *MDA;* inician por la idea, ya conocida y ampliamente establecida, de separar la especificación de la operación de un sistema, de los detalles sobre la forma en que dicho sistema usa las capacidades de la plataforma.

Los siguientes epígrafes se orientan a realizar un análisis de los conceptos relacionados con *MDA.*

1.1 Análisis conceptual de la Arquitectura Dirigida por Modelos

MDA es un concepto promovido (pero no creado) por la OMG, que propone basar el desarrollo de software en modelos especificados utilizando UML, para que, a partir de esos modelos, se realicen trasformaciones que generen código u otro modelo, con características de una tecnología particular (o con menor nivel de abstracción). MDA define un *framework* para procesar y relacionar modelos. Suele escucharse que MDA es la evolución natural de UML, ya que tiende a incrementar la cantidad de código generado, a partir de especificaciones detalladas en UML. La idea principal que subyace en *MDA* es "separar la especificación de la funcionalidad de un sistema software de los detalles sobre cómo se lleva a cabo en una determinada plataforma, de manera que los

desarrolladores escriban modelos centrados en la lógica de la aplicación y de forma automática se genere el código con los detalles propios de una plataforma" (Franky, 2010). A este nuevo paradigma se le ha denominado Ingeniería de Modelos o Desarrollo Dirigido por Modelos *"Model Driven Development"* o *MDD* por sus siglas en inglés, refiriéndose a las actividades que llevan a cabo los desarrolladores utilizando *MDA*. Según (T. Gardner, 2006) la especificación promovida por *OMG* se centra en la creación de un marco de trabajo formal donde *MDD* pueda operar.

Para lograr un mejor entendimiento se detallan algunos conceptos primarios.

- **Arquitectura:** Arquitectura de un sistema: especificación de las partes y conexiones de un sistema y de las reglas de interacción entre sus partes. (Albeiros, 2009), (Fernández Sáez, 2009).

  El término arquitectura está definido por la norma ISO/IEC 42010:2007, como la organización fundamental de un sistema, reflejada en sus componentes, sus relaciones entre sí y con el entorno y los principios que rigen su diseño y evolución. (Open Group, 2009)
- **Modelo:** Un modelo de un sistema es una descripción o especificación de un sistema realizado con un lenguaje determinado. Abstracción semánticamente completa de un sistema. (Fernández Sáez, 2009) Representación abstracta de un sistema. 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.(Albeiros, 2009)
- **Meta modelado:** Definición precisa de los constructores y reglas necesarios para definir la semántica de los modelos, que se utilizan para el modelado de clases predefinidas de problemas. (Albeiros, 2009), (Fernández Sáez, 2009).
- **Dirigido por modelos:** Aproximación que incrementa el poder de los modelos. Se dice que MDA 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.(Albeiros, 2009), (Fernández Sáez, 2009).

1.1.1 Aspectos generales y fundamentos de MDA:

Una de las iniciativas de mayor importancia en el OMG es MDA, que es un marco de trabajo de arquitecturas para desarrollo de software, con tres metas principales: portabilidad, interoperabilidad y reusabilidad. Un aspecto fundamental de MDA es su habilidad para contemplar el ciclo completo de desarrollo, cubriendo análisis, diseño, programación, pruebas, despliegue y mantenimiento. (Mike Armas, 2012)

El marco de trabajo que propone la especificación MDA está representado en tres capas, donde se muestran las tecnologías, especificaciones y estándares que lo componen según la visión que tiene OMG del mismo, así como el alcance, o sea, los marcos donde se pretende usar para ofrecer soluciones. (Consuelo, 2010)

En la figura 1 se propone una representación gráfica de los principales elementos que conforman a MDA. A continuación se realiza una descripción detallada de cada uno de estos para un mejor entendimiento:

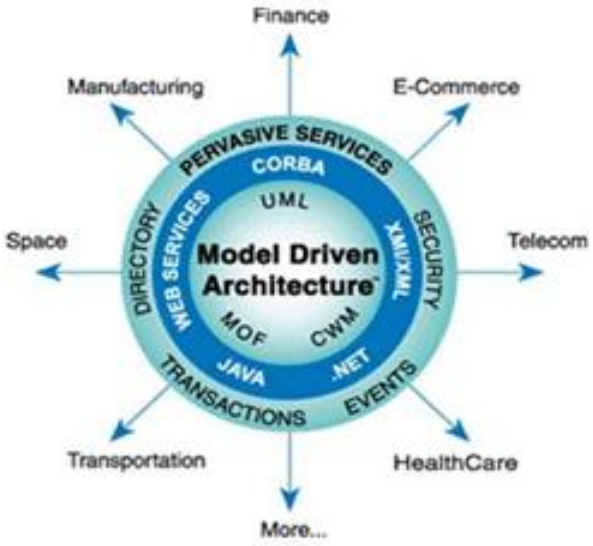


Figura 1: Arquitectura dirigidas por modelos de OMG.

Fuente: Tomado de (Quintero, 2003), (Consuelo, 2010) y (OMG, 2003)

Como se muestra en la figura anteriormente presentada MDA se divide en diferentes capas. Estas son:

1. **La capa interna o capa de modelado:** está representada por el lenguaje de modelado *UML*[2], el más utilizado para este fin; el estándar para la ingeniería dirigida por modelos *MOF*[3], diseñado en sus inicios como una arquitectura de metamodelado para definir *UML* utilizándose también para definir los principios creación y manipulación de modelos y metamodelos usando interfaces CORBA[4] que describen esas operaciones; y por *CWM* [5] que define una especificación para el modelado de metadatos para la mayoría de los objetos encontrados en el entorno de un almacén de datos, tales como, objetos relacionales, no relacionales, entre otros. (Quintero, 2003),(OMG, 2003)

[2] *UML:* "*Unified Modeling Language*"
[3] *MOF:* "*Meta-Object Facility*"
[4] *COBRA:* "*Common Object Request Broker Architecture*"
[5] *CWM:* "*Common Warehouse Metamodel*"

2. **La capa intermedia o capa de herramientas:** Representa las tecnologías y plataformas de software implicadas en MDA, tales como las plataformas JAVA, *CORBA*, *.NET* y Webs así como la especificación *XMI* "*XML Metadata Interchange*" utilizada para intercambiar diagramas *UML* entre aplicaciones de modelado.(Quintero, 2003),(OMG, 2003)
3. **La capa externa o capa de objetivos:** Equivale a las ramas o las utilidades hacia donde *MDA* se orienta, donde *MDA* puede ser utilizada para solucionar problemas. Podemos encontrar servicios penetrantes, seguridad, transacciones, eventos, directorio, etc. (Quintero, 2003),(OMG, 2003)

Según (Booch, 2004) los principios básicos que distinguen a MDA están definidos como:

1. **La representación directa**, para enfocarse más en el dominio del problema que en la tecnología.
2. **La automatización** de las tareas mecánicas que no requieran de intervención humana.
3. **Estándares abiertos** que posibiliten la interoperabilidad de las herramientas y plataformas.

1.1.2 Debilidades y fortalezas de MDA:

Cuando se adopta un nuevo paradigma resulta inexcusable realizar un estudio sobre las características específicas que lo distinguen, para explotar al máximo todos los aspectos positivos que implica su uso. A continuación se muestra de forma resumida las debilidades y fortalezas del paradigma MDA. (Córdova, 2011),(OMG, 2003)

Tabla 1: Debilidades y fortalezas del paradigma MDA.

Fuente: Elaboración propia.

Característica	Descripción	Debilidad	Fortaleza
Productividad	Los desarrolladores no tienen que escribir mucho código, ya que gran parte de ese código se genera automáticamente a partir de los modelos PIM.		**X**
Potencial para la inconsistencia del modelo	Un concepto puede ser presentado en diferentes niveles de abstracción y puntos de vista a través de diferentes modelos, como los modelos evolucionan gradualmente, su sincronización se convierte en una tarea compleja y difícil.	**X**	
Transformación automática	Las herramientas de transformación son responsables de implementar las transformaciones de modelos.		**X**

Portabilidad e independencia de plataforma	Los modelos PIM muestran una vista independiente de la solución para que puedan transformarse en múltiples modelos PSM para diferentes plataformas.		**X**
La falta de un proceso de desarrollo específico	MDA no define ni prescribe un proceso de desarrollo concreto. Las metodologías basadas en MDA necesitan definir sus propias actividades, ciclos de vida, artefactos, roles y directrices; lo que puede resultar en procesos que se desvían de las normas de la MDA.(R. Quintero, Pelechano, Fons, & Pastor, n.d.)(R. Quintero, Pelechano, Fons, & Pastor, n.d.)(R. Quintero, Pelechano, Fons, & Pastor, n.d.)	**X**	
Interoperabilidad	Diferentes modelos PSM se pueden construir a partir de un modelo PIM, por lo que la interoperabilidad multiplataforma es mucho mayor.		**X**
Aumentar el nivel de abstracción	MDA separa los modelos PIM de sus contrapartes PSM, disminuyendo la complejidad mediante la promoción de modelos a unidades de abstracción.		**X**
Problemas de la calidad del modelo	Calidad de las herramientas para la transformación del modelo; calidad del lenguaje de modelado y sus criterios de calidad; la conciliación de los aspectos de calidad conflictivos entre los modelos; medición, mejora, gestión y control de calidad de los modelos.	X	
Complejidad de la trazabilidad del modelo	Debido a la multiplicidad de modelos y la calidad requerida, la trazabilidad es una característica difícil de implementar en los procesos de desarrollo basados en MDA.	X	
Mayor facilidad de mantenimiento	En MDA, el mantenimiento es apoyado por la separación de la funcionalidad general (PIM) de las características específicas de la plataforma (PSM).(PADILLA HERNÁNDEZ, 2010)		X
Limitaciones en la escalabilidad	A medida que avanza el proceso de desarrollo, una gran cantidad de modelos son producidos por lo que el mantenimiento y la manipulación se hacen cada vez más difíciles.	X	
Roles especializados	Cada fase del desarrollo puede ser desempeñado por distintos expertos en cada campo y así dividir el trabajo dejando que cada experto se encargue solo de lo que sabe.		X

Como se puede observar en la tabla 1 MDA, según (OMG, 2003), (Córdova, 2011), presenta cinco debilidades frente a siete fortalezas. Resultado de un análisis realizado en base a doce criterios alrededor del 60 % constituyen elementos positivos en este paradigma. Siendo estos criterios elementos claves para la presente investigación.

En un estudio realizado por (Bernardo & Duitama, 2011) se presenta un amplio conjunto de reflexiones en cuanto a la adopción de enfoques de desarrollo centrado en modelos, que evidencia las debilidades y las fortalezas encontrados en diversas investigaciones

al respecto. Ilustrando de igual manera las principales problemáticas de la transformación de modelos y los trabajos previos en lo que concierne a técnicas, lenguajes y herramientas de transformación de modelos.

Lo anterior crea un antecedente para emprender trabajos que planteen soluciones a una de las principales falencias de los enfoques centrados en modelos, concretada en la siguiente frase: las técnicas, los lenguajes y las herramientas disponibles en la actualidad impiden frecuentemente a los involucrados tener control sobre el proceso y describir completamente el tipo de detalles que la aplicación final debe poseer. Estos detalles se refieren a características como la plataforma, la alineación con los procesos de negocio, los requisitos, las particularidades del código generado.(Bernardo & Duitama, 2011)

1.2 Modelos independientes de la plataforma y su transformación

El concepto de plataforma se define por (Fernández Sáez, 2009) como 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 como se lleva a cabo la misma dentro de la plataforma.

La independencia de la plataforma es una cualidad que un modelo puede exhibir cuando es expresado independientemente de las características de otra plataforma (**Mike Armas**, 2012).

En este sentido los marcos de trabajo son fundamentales. Un marco de trabajo o *framework* puede ser visto genéricamente como un conjunto de procesos y tecnologías que se utilizan para dar solución a un problema que resulta complejo. En la informática y en especial en el desarrollo de software es una estructura conceptual y tecnológica de soporte definido, compuesta por módulos de software concretos, permitiendo organizar y desarrollar otros proyectos de software de manera más fácil (Mora, 2006). La Ingeniería de Software define un *framework* o marco de trabajo como una estructura de soporte definida en la cual otro proyecto de software puede ser organizado y desarrollado. Típicamente, un *framework* puede incluir soporte de programas, librerías y un lenguaje para ayudar a desarrollar y unir los diferentes componentes de un proyecto. En el contexto organizacional un *framework* representa una arquitectura que modela las relaciones generales de las entidades del dominio. Provee una estructura y "manera de trabajo" las cuales extienden y/o utilizan las aplicaciones. (Bonillo, 2014).

La definición de transformación o *mapping MDA* proporciona la especificación de la transformación de un *PIM* en un *PSM* para una plataforma determinada se distinguen dos tipos de definiciones de transformaciones según (J. B. Quintero & Anaya, 2007):

- **Transformaciones de tipos *Model Type Mapping:*** según la especificación, "una transformación de tipos especifica permite transformar cualquier modelo construido con tipos del *PIM* a otro modelo expresado con tipos del *PSM*". En el caso de *UML*, estas reglas pueden estar asociadas a tipos del metamodelo (clase, atributo, relación) o a nuevos tipos definidos mediante estereotipos. También pueden definirse reglas en función de valores de instancias en el *PIM*.
- **Transformaciones de instancias *Model Instance Mapping*:** identifica elementos específicos del *PIM* que deben ser transformados de una manera particular, dada una plataforma determinada. Esto se puede conseguir mediante marcas.

Una marca representa un concepto del *PSM*, y se aplica a un elemento del *PIM* para indicar cómo debe ser transformado. Las marcas, al ser específicas de la plataforma, no son parte del *PIM*. El desarrollador marca el *PIM* para dirigir o controlar la transformación a una plataforma determinada.(Mora, 2006)

La mayoría de las definiciones de transformación consisten en alguna combinación de los dos enfoques. Una transformación de tipos sólo es capaz de expresar transformaciones en términos de reglas sobre elementos de un tipo en el *PIM* que se transforman en elementos de uno o más tipos en el *PSM*. (Mora, 2006)

Toda transformación de instancias del modelo tiene restricciones implícitas de tipo que deben cumplirse al marcar el modelo para que la transformación tenga sentido. Implícitamente a cada tipo de elemento del *PIM* sólo pueden aplicarse determinadas marcas, que indican qué tipo de elemento se generará en el *PSM*. Las transformaciones basadas en marcas deben establecer qué marcas son aplicables a qué tipos del *PIM*.(Mora, 2006)

1.3 Lenguaje Unificado de Modelado

UML (Unified Model Language) por sus siglas en ingles es un lenguaje estándar para visualización, especificación, construcción y documentación de sistemas software y otros sistemas no software. Representa una colección de buenas prácticas que proporcionan un éxito acreditado en el modelado de grandes y complejos sistemas. Fusiona conceptos de las metodologías de Booch, de OMT (Object Modeling Technique) y de OOSE (Object-Oriented Software Engineering) fundamentalmente aunque posteriormente incorporó conceptos de otras metodologías, consiguiendo como

resultado un lenguaje de modelado común, sencillo y ampliamente utilizado por los usuarios de éstos y otros métodos de desarrollo, ampliando sus posibilidades. (Fernández Sáez, 2009)

Los diferentes elementos del modelado *UML* permiten especificaciones estáticas y dinámicas de un sistema orientado a objetos. Los modelos estáticos incluyen la definición de clases, atributos, operaciones, interfaces y relaciones entre clases, como pueden ser la herencia, asociación, dependencia. La semántica de comportamiento de un sistema puede representarse por medio del lenguaje *UML* gracias a sus diagramas de secuencia y comunicación. Para modelados más complejos, *UML* también proporciona mecanismos para la representación de máquinas de estado. Este lenguaje proporciona una notación para representar el agrupamiento del diseño lógico por medio de componentes y el despliegue y ubicación de esos componentes en nodos dentro de una arquitectura distribuida.(Mora, 2006)

La versión 2.0 de *UML* cuenta con trece diagramas divididos en dos grupos de acuerdo a sus funcionalidades. A continuación se realiza un análisis de algunos de ellos, que se tendrán en cuenta en esta investigación:

1.3.1 Diagramas Estructurales:

Estos diagramas hacen énfasis en los elementos que deben existir en el sistema de modelado mostrando como están relacionados los mismos:

- **Diagrama de Clases**: Es el diagrama más utilizado en el modelado de sistemas orientados a objetos. Se usa para modelar el vocabulario de un sistema, las colaboraciones simples, y un esquema lógico de base de datos. Se utiliza para mostrar clases, interfaces, colaboraciones y las relaciones entre todas ellas. Las relaciones entre clases pueden ser de dependencia, generalización o herencia, y de asociación, las relaciones de asociación deben tener una multiplicidad para señalar cuantos objetos pueden conectarse a través de una instancia a una asociación. Hay dos tipos de asociaciones especiales: agregación y composición.
- **Diagrama de Despliegue**: Muestra la configuración de nodos de procesamiento en tiempo de ejecución y los artefactos que residen en ellos. Cubren la vista de despliegue estática de una arquitectura. Se relacionan en los diagramas de artefactos en que un nodo incluye, por lo común, uno o más artefactos.
- **Diagrama de Componentes**: Muestra las clases encapsuladas y sus interfaces, puertos y estructuras internas, consistentes de componentes anidados y conectores. Cubren la vista de implementación estática de diseño de un sistema. Son importantes para la construcción de sistemas grandes por partes pequeñas.

1.3.2 Diagrama de Comportamiento:

Son un grupo de diagramas utilizados para mostrar como es el comportamiento llevado a cabo por el sistema que se modela, las partes del mismo y otros agentes externos.

- **Diagrama de Casos de Uso**: Muestra un conjunto de casos de uso y actores (un tipo especial de clases) y sus relaciones. Cubren la vista de casos de uso estática de un sistema. Estos diagramas son especialmente importantes en el modelado y organización del comportamiento de un sistema. Esto implica, la mayoría de las veces, modelar el contexto del sistema, subsistema o clase, o el modelado de los requisitos de comportamiento de esos elementos.
- **Diagrama de Actividades**: Muestra el flujo de control entre actividades, por lo que constituye fundamentalmente un diagrama de flujo, que además puede mostrar concurrencias y ramas de control. Es utilizado para modelar los aspectos dinámicos de un sistema, más exactamente, los pasos secuenciales, incluida la concurrencia, de un proceso computacional. Los diagramas de actividades pueden utilizarse para visualizar, especificar, construir y documentar la dinámica de una sociedad de objetos, o también, para modelar el flujo de control de una operación.
- **Diagrama de Estados**: Muestra una máquina de estado, o sea, un comportamiento que especifica las secuencias de estados por las que pasa un objeto durante su vida, en respuesta a eventos, junto con sus respuestas a esos eventos. Estos diagramas son junto con los diagramas de actividades útiles para modelar la vida de un objeto, pero a diferencia de estos que muestran el flujo de control entre actividades ellos lo muestran entre estados. Pueden estar asociados a las clases, casos de uso o a sistemas completos para visualizar, especificar, construir y documentar la dinámica de un objeto individual.
- **Diagrama de Secuencia**: Utilizado para modelar interacciones entre un conjunto de objetos de una aplicación a través del tiempo. Se modela para cada método de una clase y contiene detalles de la implementación del escenario que no es más que una parte del sistema (relación entre los objetos más interesantes) que se quiere resaltar, así como, de los elementos que lo componen, o sea, objetos y clases, y de los mensajes que son intercambiados entre los objetos, ordenándolos temporalmente.

1.4 Proceso de desarrollo de software MDA

El desarrollo de sistemas de software es una labor compleja en la que intervienen múltiples elementos a lo largo del proceso (Calisoft, 2014; Lazo Alvarado et al., 2016). La diversidad de lenguajes y técnicas de programación, así como los disímiles entornos

integrados de desarrollo crean una indecisión en el momento de su selección. En la actualidad la selección de una tecnología o una combinación de estas es una tarea compleja y permeada de incertidumbre (Pérez Armayor, 2014; Pérez Armayor, Abreu Fong, Hernández Lantigua, León Alen, & Díaz Batista, 2016).

Diversas investigaciones han demostrado que es muy frecuente el fracaso de los proyectos de software (Pressman, 2001) debido a los continuos cambios que sufren los requerimientos del usuario hasta la obtención del producto final. Ocasionando que se entreguen productos que no satisfacen las necesidades del cliente o que los sistemas deban implementarse una y otra vez.(Bernardo & Duitama, 2011)

En los enfoques tradicionales de software, incluso las nuevas tendencias de enfoques ágiles, en cada etapa del ciclo de vida del producto participan diferentes actores que asumen distintos roles. Provocando que en cada fase se hagan ajustes o interpretaciones de acuerdo a las expectativas de su ejecutor. Conllevando a que no se obtenga lo que realmente se quería. Esto ha impulsado la búsqueda de soluciones que permitan, de manera asistida por computadoras, realizar conversiones invisibles a los usuarios para transformar modelos independientes de una plataforma en su equivalente en un modelo específico de una determinada plataforma de implementación.

OMG no propone un modelo estándar de desarrollo de software para MDA. Sin embargo tal modelo es necesario ya que puede permitir organizar los pasos necesarios para el desarrollo del software de una forma definida, sistemática, y repetible (Quintero, 2003). Previo a la definición del ciclo de desarrollo de software se deben realizar las siguientes actividades:

1. Identificar los aspectos estructurales y de comportamiento del dominio de la aplicación y definir el perfil UML que permita expresar cualquier modelo del mismo en un PIM. Este perfil UML sería el metamodelo del dominio.
2. Identificar la plataforma o plataformas tecnológicas destino y definir el perfil UML que permita expresar su posterior modelo en un PSM.
3. Definir las técnicas de correspondencia entre los metamodelos PIM y PSM. Estás Técnicas de correspondencia se deben definir en términos de transformaciones de los elementos de los metamodelos más abstractos PIM hacia el metamodelo más concreto PSM.
4. Para cada una de las técnicas de correspondencia definir los Modelos de Información complementaria requeridos y provistos, buscando, con el primero definir sin ambigüedades la correspondencia y preservar la semántica del nivel de

abstracción más alto y proveyendo a los niveles más bajos de abstracción los aspectos necesarios para su final implementación.

5. Implementar las técnicas de correspondencia ya sea manualmente o asistida mediante una herramienta CASE de soporte.

En este contexto surge *MDA* para brindar una alternativa que se centra en los modelos los cuales se encargan de guiar todo el proceso de desarrollo, permitiendo la generación de código de manera automatizada, a partir de modelos y las reglas de transformación establecidas. En la figura 3 se presenta el proceso de desarrollo de software siguiendo el paradigma *MDA*.

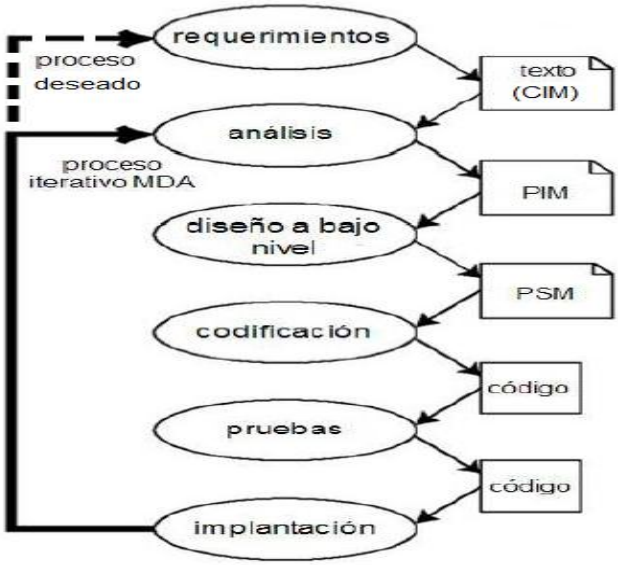


Figura 2: Etapas del desarrollo de software con MDA.
Fuente: Tomado de (Pons, 2010)

La figura muestra el proceso de desarrollo de software siguiendo el paradigma *MDA*, el cual no dista mucho del proceso tradicional. La diferencia reside en la naturaleza de los artefactos creados durante el proceso de desarrollo. A continuación se describen estos artefactos que no son más que modelos y que constituyen el núcleo de *MDA*:

1. **Modelo Independiente de la Computación "*Computationally Independent Model*" *(CIM):*** Surge en la fase inicial del proceso de desarrollo comprendiendo la modelación del negocio en su totalidad; donde se modelan los requisitos del sistema, se describe la situación en la que el sistema será utilizado y sirve de enlace entre los expertos en el dominio del problema y sus requisitos con los expertos en el diseño y construcción de software. Los requisitos pueden ser representados mediante diagramas de caso de uso, de actividad y de secuencia. Entre los tipos de requisitos más comunes obtenidos en CIM se encuentran los

requisitos de usuario, los funcionales, los no funcionales y los organizacionales. (Martínez y cols. 2015)

2. **Modelo Independiente de la Plataforma *(PIM):*** Representa los modelos que describen una solución de software que no contiene detalles de la plataforma concreta donde será implementada la solución, de ahí su nombre de modelos independientes de la plataforma. Estos modelos surgen como resultado del análisis y diseño. Tal modelo tiene cierto nivel de abstracción que no cambia; sin importar la plataforma que sea elegida para su implementación. (Martínez y cols. 2015)

   Las principales características que debe tener el PIM necesarias para la transformación de PIM a PSM según (Pons, 2010) son:

   - Formación del modelo abstracto.
   - Describir el comportamiento del sistema, aspectos funcionales y no funcionales independientes del entorno de computación y tecnologías de implementación. Y que puedan ser reutilizados en múltiples plataformas.
   - Los requisitos del negocio se especifican utilizando diagramas *UML*.
   - El sistema es modelado desde el punto de vista que mejor soporte los requisitos del usuario final.
   - Que sea independiente de la implementación de la plataforma/tecnología.

3. **Modelos Específicos de la Plataforma *(PSM)*:** Los modelos específicos de la plataforma. Surgen del PIM y se crean entre las fases del diseño y la codificación de la solución. Luego, él código se genera después de la codificación y las pruebas. (Martínez y cols. 2015)

1.5 Herramientas MDA Propietarias

1.5.1 OptimalJ

OptimalJ es una herramienta MDA desarrollada por la empresa *Compuware*, que utiliza MOF para dar soporte a UML y XMI, permitiendo generar aplicaciones JEE completas a través de un Modelo Independiente de la Plataforma. En *OptimalJ* están presentes los siguientes modelos:

- **Modelo del Dominio:** Modelo que describe el sistema sin detalles de la implementación o sea a un nivel alto de abstracción, es equivalente al PIM y su elemento principal es un modelo de clases del negocio.

- **Modelo de la Aplicación:** Constituye el PSM de la aplicación el cual es generado automáticamente a partir del modelo del dominio y se encuentra conformado por el modelo de base de datos, modelo de interfaz web y el modelo EJB.
- **Modelo de Código:** El código fuente de la aplicación, generado a partir del modelo de la aplicación.

El proceso de desarrollo con OptimalJ está organizado de la siguiente manera.

- Se genera automáticamente los PSM de la capa de presentación (web), capa de negocio EJB y base de datos manteniendo la conexión entre las capas como se muestra en la Figura 4.
- Se distingue los bloques libres y protegidos del código para evitar la modificación del código generado. Una nueva generación de la aplicación mantiene el código generado en los bloques libres.
- La interfaz web generada proporciona una navegación por defecto para cada objeto de negocio, que permite el mantenimiento de los datos asociados a las clases del Modelo del Dominio. Esa interfaz es muy pobre pero existe la posibilidad de crear un patrón de presentación que se ajuste a unas necesidades concretas, o bien manualmente modificar el código de la aplicación.

Generación de puentes de comunicación

Un esquema general sería aquel en el que se tiene varios *PSMs* derivados del mismo *PIM*. Deberían generarse también los puentes de comunicación entre las distintas partes.

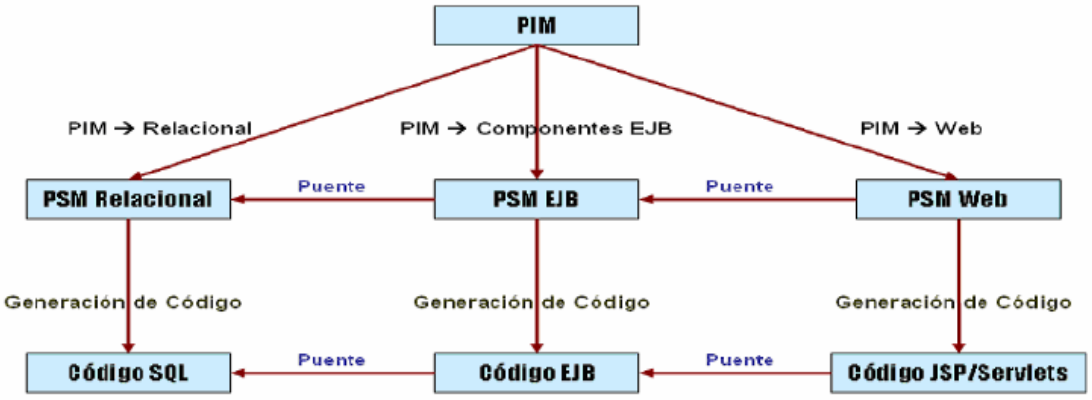


Figura 3: Conexión entre los modelos OptimalJ.

Fuente: Tomado de (García, 2004)

Un esquema típico de desarrollo con *MDA* usando varios *PSMs* es el que aparece en la Figura 4 a partir del *PIM* se generan tres *PSMS*:

- Un modelo relacional del sistema, que describe el esquema de base de datos del sistema mediante diagrama de Entidad - Relación.
- Un modelo *EJB*, mostrando los aspectos relativos a la plataforma *EJB*
- Un modelo *Web* para describir las interfaces Web del sistema.

Los puentes de comunicación ente *PSMs* y código permiten a la capa web comunicarse con los componentes *EJB*, y a estos comunicarse con la base de datos del sistema.

1.5.2 GMF

ArcStyler está desarrollada por la empresa *iO-Software*. Utiliza *MOF* para soportar estándares como *UML* y *XMI* y también *JMI* (*Java Metadata Interface* o Interfaz de Metadatos Java) para acceder al repositorio de modelos. ArcStyler logra la integración de herramientas de modelación *UML* y herramientas de desarrollo (ingeniería inversa, explorador de modelos, construcción y despliegue) con la arquitectura CARAT que permite crear, editar y mantener cartuchos *MDA* (*MDA-Cartridge*) para definir transformaciones. También cubre todo el ciclo de vida de un proyecto al brindar herramientas para el modelado del negocio y el de requisitos.

Los cartuchos creados utilizando la arquitectura CARAT contienen un conjunto de reglas de transformación y se instalan como un *plug-in*. Se utiliza el lenguaje de script Jphyton para programarlos, los existentes brindan soporte para las plataformas JEE, .NET, servicios web, etc. También un cartucho se puede definir a partir de otro cartucho permitiendo la llamada herencia de cartuchos.

ArcStyler proporciona los dos modos de anotar un *PIM* con información específica de la plataforma establecidos por la especificación de *MDA*, los cuales son: Correspondencia de tipos (*Model type mapping*) y Correspondencia de instancias (*Model instance mapping*).

1.6 Herramientas MDA de código abierto

1.6.1 Análisis conceptual de las pruebas de software

AndroMDA es un *framework MDA open source*, recibe como entrada un modelo UML en formato XMI, los combina con los *plugins* de *AndroMDA* (*cartridge* y *translation libraries*) y produce código fuente. Surge en el 2003 como continuación y ampliación del proyecto *UML2EJB* con el objetivo la generación de código para programar con *EJB2.0* (Enterprise JavaBean). Es aprobado en *SourceForge* también en el 2003 teniendo en

su arquitectura algunos cambios significativos para que pueda crecer con el futuro. Las fortalezas de la versión 3.2 de *AndroMDA* son:

- Soporte para UML 2.0 y herramientas basadas en Eclipse EMF (*MagicDraw* 11.6, RSM, etc.).
- Integración con Maven2.
- Generación de código para PSM clases de metamodelo.
- Soporte para el *Freemarker template engine*.
- Generación de código para *Java Generics*.
- Mejora en la documentación y nuevos tutoriales.
- Corrección de errores y pequeñas mejoras.
- Recibe una retroalimentación de la comunidad de usuarios y documentadores que forma parte del desarrollo del mismo.
- Incluye un conjunto de cartuchos enfocados a los kits de desarrollo actuales como son *Apache Axis, jBPM, Apache Struts, JSF, Spring e Hibernate.*
- Incluye un *Kit* que permite desarrollar cartuchos generadores de código o personalizar los existentes, el cartucho Meta, con él se puede construir un generador de código empleando una herramienta de UML.

La principal desventaja de *AndroMDA* es que no brinda soporte para la edición directa de UML y pese a ser una herramienta de código libre depende de otras herramientas propietarias (como MagicDraw) para esta función, aunque también es compatible con *ArgoUML* que, si es libre, pero esta herramienta se encuentra en una etapa primaria de su ciclo de vida.

1.6.2 GMF

El marco de trabajo para el modelado gráfico *(*"*Graphical Modeling Framework*", *GMF* por sus siglas en inglés) es un acrónimo que nombra a un marco de trabajo de Eclipse que constituye una alternativa para la generación de editores gráficos que utilizan *MDA*, está basado en *EMF "Eclipse Modeling Framework*" y *GEF "Graphical Editing Framework*". Con él se pueden crear editores que utilicen el lenguaje *UML* u otro lenguaje de modelado ya que parte de un metamodelo donde se especifica el lenguaje de modelado que utilizará. Como resultado final se genera un plug-in que añadido a un proyecto de Eclipse permite editar gráficamente el modelo especificado con anterioridad. (Plante, 2006)

Está dividido en dos grandes componentes según (Sáez, 2009):

- ***GMF Tooling*:** que es el componente de instrumentación con el que se definen los aspectos de notación y semánticos y también las funcionalidades del editor gráfico, así como la parametrización del generador que se va a encargar de construir el código del mismo y generar el *plug-in* correspondiente.
- ***GMF Runtime*:** es el encargado de ejecutar el *plug-in* generado por el *GMF Tooling*

En la Figura 5 se detalla cómo es la arquitectura para el desarrollo basado en *GMF*. Está compuesta por varios modelos, partiendo del modelo del dominio, la definición gráfica de los elementos que van a componer el editor y la definición de las herramientas que se asocian a cada objeto. Con el modelo de mapeo se unifican los tres modelos anteriores para luego ser usado como entrada para la transformación modelo a modelo dando como resultado al modelo generador mediante el cual se genera el código que permite la ejecución del editor.

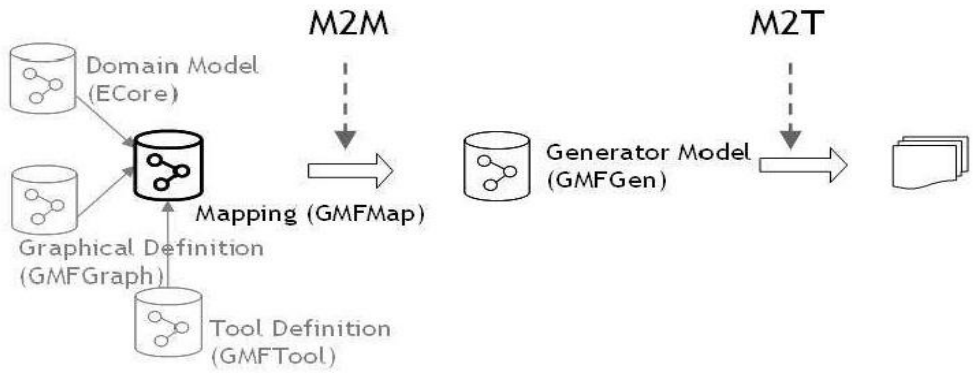


Figura 4: Arquitectura para el desarrollo basado en GMF.

Fuente: Tomado de (Moreno, 2009)

El modelo del dominio se define en base a *EMF* y establece la información no gráfica gestionada por el editor, o sea es el metamodelo en el cual se va a basar todo el desarrollo. Existen tres formas de conseguirlo: importando un metamodelo *Ecore* de EMF, mediante el uso del editor estándar de EMF o haciendo uso del propio editor *Ecore* que está incorporado en GMF.

El modelo de definición gráfica como se menciona anteriormente, se utiliza para definir los elementos gráficos que se van a visualizar con el editor, por ejemplo: figuras, nodos, conexiones, etiquetas.

El modelo de definición de herramientas permite definir elementos tales como el cuadro de herramientas, menú principal, la paleta y el resto de acciones que se van a asociar a cada elemento.

El modelo de mapeo establece la correlación existente entre los elementos del dominio representado por el diagrama .ecore y los elementos gráficos representados por los diagramas .gmfgraph y .gmftool.

El modelo de generación permite definir los detalles de implementación para la fase de generación del código. El resultado final es el antes mencionado *plug-in* personalizable y mejorable, mediante el cual los desarrolladores pueden modelar software basándose en un metamodelo concreto.

El *GMF Runtime* permite trabajar al desarrollador con el editor gráfico generado. En la siguiente figura se detalla las dependencias entre el editor gráfico generado, el *runtime* de GMF y los componentes GEF y EMF, todos ellos soportados por la plataforma Eclipse.

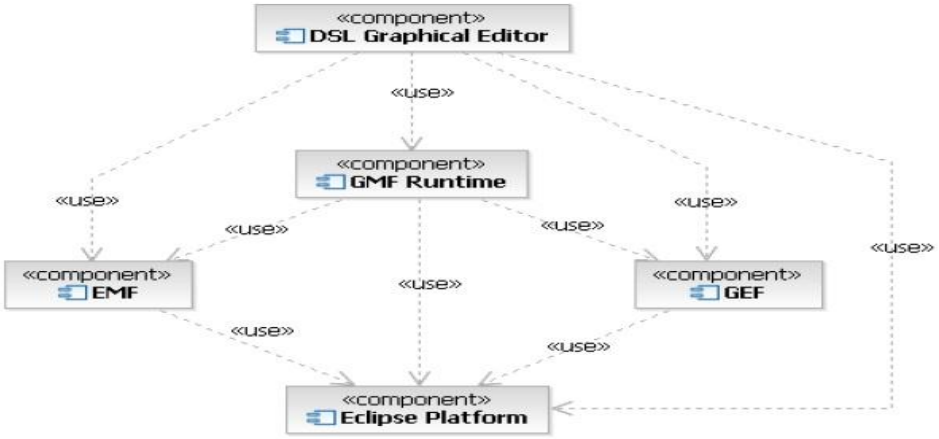


Figura 5: Dependencias de la plataforma GMF.

Fuente: Tomado de (Plante, 2006)

En la Figura 6 se puede observar que el editor gráfico tiene una dependencia de uso de *GMF Runtime* ya que este es el encargado de ejecutar el *plug-in*, además también depende de los modelos de dominio y los componentes gráficos definidos en EMF y GEF respectivamente y todos a su vez dependen de la plataforma Eclipse.

1.6.3 Comparación entre las herramientas de código abierto

El resumen de las dos herramientas tratadas anteriormente comparándolas a ambas con determinados criterios y destacando sus mayores debilidades como herramientas MDA. Se presenta en la siguiente tabla:

Tabla 2 Comparación entre AndroMDA y GMF. Fuente: Elaboración propia.

Criterios	AndroMDA	GMF
Código abierto	Si	Si
Soporte	UML 2.0, Freemarker Template Engine	UML 2.0
Herramientas necesarias	MagicDraw 9.5 y Apache`s Maven	Eclipse,EMF "*Eclipse Modeling Framework*" y GEF "*Graphical Editing Framework*"
Integración	Marven2	Otras
Niveles que cubre	Implementa los niveles PIM y PSM	Permite definir CIMs, PIMs y PSMs
Ingeniería inversa	No	No
Grado de interacción con el usuario	Alta	Alta
Ámbito de Aplicación	El desarrollo orientado a servicios y SI Web.	Al desarrollo de sistemas de propósito general.
Debilidades	Depende de otras herramientas propietarias para brindar soporte la edición directa de UML.	Dependiente de la arquitectura Eclipse.

Como se puede apreciar en la Tabla 2 *AndroMDA* y *GMF* tienen varios elementos en común, pero *GMF* incorpora funcionalidades más abarcadoras que *AndroMDA* por lo que en la presente investigación se asumen las buenas prácticas utilizadas en las dos herramientas, pero prestándole mayor atención a *GMF* de Eclipse.

1.7 Análisis de la herramienta CASE jMDA v1.0, v2.0, v3.0 y v4.0

Para el desarrollo de la versión 5.0 de la herramienta CASE jMDA resulta imprescindible hacer un análisis crítico de las versiones anteriores. Es preciso comenzar diciendo que dicha herramienta *CASE* se puede clasificar por su funcionalidad como un híbrido de editor de *UML* y soporte de MDA, la conjunción de estos dos aspectos, es muy factible ya que UML es el lenguaje de modelado por excelencia y es absolutamente compatible con MDA.

Durante esta fase del trabajo es importante la identificación de las dimensiones y criterios de evaluación de herramientas *MDA*. En la evaluación realizada a la herramienta *OptimalJ* y a *ArcStyler*, estudio realizado por (García, 2004) se aplica una metodología de análisis de características y se elige un conjunto de propiedades extraídas de la especificación de *MDA*. En la Tabla 3 aparece una relación de estas propiedades. El presente estudio comparativo se realiza en función de estas propiedades, que además sirven como guía de elementos a tener en cuenta en la implementación de la nueva versión.

Tabla 3: Propiedades o criterios de comparación.

Fuente: Adaptado de (García, 2004).

Id	Propiedad	Descripción
P01	Soporte para *PIMs*	La herramienta permite que se especifique un sistema mediante un modelo independiente de cualquier plataforma *(PIM).*
P02	Soporte para *PSMs*	La herramienta permite construir modelos del sistema que capturan los aspectos esenciales de una tecnología de implementación determinada *(PSMs).*
P03	Permite varias implementaciones	La herramienta posibilita la generación de varias implementaciones diferentes a partir del mismo *PIM*, utilizando *PSMs*, marcas u otros mecanismos. También se tendrá en cuenta que puedan añadirse nuevas implementaciones a las disponibles en la herramienta.
P04	Integración de Modelos	Permite integrar diferentes modelos para producir una única aplicación, principalmente en la generación de los "puentes" apropiados para comunicar las distintas partes entre sí.
P05	Interoperabilidad	La herramienta puede importar y exportar información a otras herramientas.
P06	Acceso a la definición de las transformaciones	La herramienta provee de un mecanismo de definición de transformaciones entre modelos y permite al usuario crear nuevas transformaciones o modificar las existentes para satisfacer sus requisitos específicos.
P07	Verificador de Modelos	Incluye algún mecanismo para chequear la corrección de los modelos, incluidos *PIM* y *PSM*.
P08	Expresividad de los modelos	La herramienta tiene un lenguaje para representar *PIMs* y *PSMs* lo suficientemente expresivo como para capturar de forma precisa la estructura y funcionalidad en los distintos niveles de abstracción.
P09	Uso de patrones	La herramienta aplica o permite aplicar patrones de diseño en la construcción del *PIMs, PSMs* y código, y pueden definirse otros nuevos o modificar existentes.
P10	Soporte para la regeneración de modelos	La herramienta proporciona soporte para rehacer modelos, por ejemplo, regenerar *PIM* a partir de los *PSMs* y viceversa. También debe permitir conservar los cambios efectuados “manualmente” tanto a nivel de modelo como de código.
P11	Transformaciones intra-modelo	Provee soporte para transformar un *PSM* a otros *PSMs*, o un *PIM* a otros *PIMs.*
P12	Trazabilidad	Incorpora un mecanismo para seguir el rastro de determinadas transformaciones desde su origen hasta su destino.
P13	Ciclo de vida	Incluye la mayor parte posible del ciclo de vida de un desarrollo con *MDA*, esto es, el análisis, el diseño, la implementación, el ensamblado, el despliegue y el mantenimiento.
P14	Estandarización	La herramienta utiliza los estándares básicos de *MDA*. Por ejemplo expresa sus modelos en *UML*, es capaz de importar y exportar modelos en *XMI* y de guardarlos en un repositorio *MOF.*

P15	Control y refinamiento de las transformaciones	La aplicación permite dirigir o controlar las transformaciones entre modelos, entre *PIM* y *PSMs* o entre *PSM* y código. Por ejemplo, dispone de parámetros en las transformaciones, permite seleccionar elementos a ser transformados o establecer condiciones para las transformaciones.
P16	Herramientas de soporte	Además de las herramientas de transformación, la aplicación incluye otras herramientas para dar soporte completo a *MDA*: editor de código, editor de modelos, herramientas para prueba y despliegue.

A continuación se realiza una comparación entre las versiones anteriores de la herramienta *jMDA*, estudio que permite comprobar su evolución y las principales carencias que todavía mantiene. Para ello, se puntúa del 0 al 4 cada propiedad de la Tabla 3, según los siguientes criterios mostrados en la Tabla 4:

Tabla 4 Criterios de evaluación.
Fuente: Tomado de (García, 2004).

Puntuación	Descripción
0	Soporte nulo de la propiedad
1	Soporte mínimo para la propiedad
2	Soporte medio para la propiedad
3	Buen soporte para la propiedad
4	Excelente soporte para la propiedad

En base a la escala que se define en la Tabla 5 se procede a realizar una ponderación de cada una de las propiedades en base a cada una de las versiones de la herramienta *jMDA*.

Tabla 5 Comparación de las versiones anteriores de jMDA.

Fuente: Elaboración propia.

ID	JMDA PIM-PSM 1.0	JMDA PIM-PSM 2.0	JMDA PIM-PSM 3.0	JMDA PIM-PSM 4.0
P01	2	3	4	4
P02	2	3	4	4
P03	0	0	0	0
P04	0	0	0	0
P05	0	0	0	0
P06	0	0	0	1
P07	0	0	0	0
P08	2	3	4	4
P09	0	0	0	2
P10	0	0	0	1
P11	0	0	0	0
P12	0	0	0	4
P13	2	3	4	4
P14	1	1	1	2
P15	0	0	0	1
P16	1	1	1	2
Total	**10**	**14**	**18**	**28**

Como se puede observar en la Tabla 5, la herramienta jMDA fue evolucionando continuamente en las diferentes versiones. Se aprecia que los resultados los de la versión 4.0 son mucho más alto que en las versiones anteriores aunque no deja de tener algunos problemas que se intentarán corregir en la siguiente versión para así resolverlos completamente.

1.8 Patrón Modelo Vista Controlador (MVC)

La arquitectura del patrón MVC (Modelo-Vista-Controlador) originalmente fue aplicada en el modelo de interacción gráfica de usuarios, para entradas, procesamientos y salidas. Esta arquitectura descompone una aplicación en tres capas, donde cada capa es una estructura lógica de los diferentes elementos que componen el software. Las capas en que se divide el patrón MVC son el Modelo, la Vista y el Controlador según (Gulzar, 2002) y (Gaitán Torres, 2012) . A continuación se describen brevemente:

Modelo: El modelo representa los datos de una aplicación y contiene la lógica para acceder a ellos y manipularlos. Los servicios que maneja el modelo deben ser lo suficientemente genéricos como para soportar varios tipos de clientes y debe ser fácil entender cómo controlar la conducta del modelo con tan solo revisar brevemente la lista de sus métodos. El modelo notifica a las vistas cuando cambia su estado y proporciona facilidades para que las vistas consulten el modelo acerca de su estado. También proporciona facilidades para que el controlador acceda a la funcionalidad de la aplicación encapsulada por el modelo.(Gaitán Torres, 2012)

Vista: La vista se encarga de acceder a los datos del modelo, especifica cómo se deben presentar esos datos y actualiza la presentación de los mismos cuando ocurren cambios en el modelo. La semántica de presentación está dentro de la vista, por lo tanto, la información contenida en el modelo se puede adaptar a diferentes tipos de vistas. La vista se modifica cuando el modelo se comunica con ella y a su vez, la vista envía información introducida por el usuario al controlador. (Gaitán Torres, 2012)

Controlador: El controlador define el comportamiento de la aplicación. Despacha las peticiones del usuario y selecciona las vistas de presentación siguiente basándose en la información introducida por el usuario y en el resultado de las operaciones realizadas por el modelo. Es decir, interpreta las entradas del usuario y las mapea en acciones a ser efectuadas por el modelo.(Gaitán Torres, 2012)

1.9 Análisis conceptual de las pruebas de software

Las pruebas de software son investigaciones técnicas, que tienen como objetivo proporcionar información objetiva acerca de la calidad del producto. Se puede

denominar las pruebas como una actividad más en el proceso de control de calidad del software, siendo estas un conjunto de actividades dentro del desarrollo del mismo. En dependencia del tipo de prueba que se realice, estas actividades se podrán desarrollar en cualquier momento del proceso de desarrollo. Los elementos que condicionan las pruebas a realizar; deben ser seleccionados y utilizados de la manera más eficiente posible según el contexto del proyecto (Pressman, 2001; Sommerville, 2002). Uno de los tipos de pruebas existentes son los que se le realiza al software están orientadas a comprobar sus funcionalidades.

Las pruebas funcionales o de caja negra son un enfoque para llevar a cabo pruebas donde estas se derivan de la especificación del programa o componente. El sistema es una "caja negra" cuyo comportamiento sólo se puede determinar estudiando las entradas y salidas relacionadas. Otro nombre para estas es pruebas funcionales debido a que al probador sólo le interesa la funcionalidad y no la implementación del software.(Pressman, 2001; Sommerville, 2002).

Este enfoque se aplica de igual forma a los sistemas que están organizados como funciones o como objetos. El probador introduce 1as entradas en los componentes del sistema y examina las salidas correspondientes. Si las salidas no son las previstas, entonces la prueba ha detectado exitosamente un problema con el software.(Sommerville, 2002).

1.10 Conclusiones del capítulo

1. El paradigma *MDA* permite incrementar la productividad, interoperabilidad y calidad del software que se desarrolla. La bibliografía consultada sobre este marco de trabajo refleja la importancia, no solo de su aplicación por motivos de mejorar el proceso de desarrollo de software, sino de la necesidad de contar con herramientas adecuadas que cumplan con sus propósitos.
2. No se ha identificado un estándar *XMI* que permita la estandarización de los diagramas modelados para su intercambio. Esto significa que será necesario desarrollar un método de resguardo de los diagramas que se implementen en un modelo dado para su posterior transformación y edición en el otro modelo, así como su captura por el siguiente módulo de la herramienta jMDA.
3. Las herramientas de código abierto, *GMF* y *AndroMDA*, presentan características que pudieran ser buenas prácticas y serán empleadas como referentes en la presente investigación. No obstante, al revisar sus características, es apreciable que además de no ser soberanas, no cuentan con

todo lo necesario para ser usadas con el propósito deseado y planteado en la problemática.

4. Las versiones 1.0, 2.0 ,3.0 y 4.0 de la herramienta *CASE jMDA* presentan limitantes en algunas funcionalidades que no están del todo correctamente implementadas. Es por ello que se decide corregir los errores haciendo una reingeniería sobre la versión 4.0 de la herramienta *CASE jMDA* haciendo otra versión ya con los problemas corregidos esta será la 5.0.
5. La variante de generación asistida por computadora de los diagramas *UML* en el nivel *PSM*, a partir de los diagramas que se desarrollen en el *PIM* por parte de una analista de sistemas, será utilizando el Patrón Vista Controlador que se adecua a las necesidades de un Sistema de Información.

CAPÍTULO 2

Capítulo 2: *"Análisis, diseño e implementación del Módulo PIM-PSM 5.0 de la herramienta CASE jMDA"*

En este capítulo se realiza un abordaje de los principales elementos relacionados con las fases de análisis, diseño e implementación de un producto de software. Se emplea el Lenguaje Unificado de Modelado para mostrar los aspectos funcionales, estructurales y de comportamiento de la herramienta que se diseña, además se tratan temas relacionados con la arquitectura del sistema a implementar. Para ello se modelan los diagramas que permitan comprender su funcionamiento de forma sencilla. Además, se describe cómo se lleva a cabo el tratamiento de errores.

2.1 Requisitos funcionales del sistema

La definición de las necesidades del sistema es un proceso complejo, pues en él hay que identificar los requisitos que el sistema debe cumplir para satisfacer las necesidades de los usuarios finales y de los clientes. Para realizar este proceso, no existe una única técnica estandarizada y estructurada que ofrezca un marco de desarrollo que garantice la calidad del resultado. Para la definición de los requisitos que se muestran a continuación se utiliza la técnica del "lenguaje natural:"

RF1.Modelar diagrama de clases: Esta funcionalidad se encarga de permitir al usuario la creación de nuevos diagramas de clase en el modelo independiente de la plataforma.

RF2.Modelar diagrama de casos de uso: Esta funcionalidad se encarga de permitir al usuario la creación de nuevos diagramas de casos de uso en el modelo independiente de la plataforma.

RF3.Modelar diagrama de estado: Esta funcionalidad se encarga de permitir al usuario la creación de nuevos diagramas de estado en el modelo independiente de la plataforma.

RF4.Modelar diagrama de secuencia: Esta funcionalidad se encarga de permitir al usuario la creación de nuevos diagramas de secuencia en el modelo independiente de la plataforma.

RF5.Modelar diagrama de actividades: Esta funcionalidad se encarga de permitir al usuario la creación de nuevos diagramas de actividades en el modelo independiente de la plataforma.

RF6.Modelar diagrama de artefacto: Esta funcionalidad se encarga de permitir al usuario la creación de nuevos diagramas de artefactos en el modelo específico de la plataforma.

RF7.Modelar diagrama de despliegue: Esta funcionalidad se encarga de permitir al usuario la creación de nuevos diagramas de despliegue en el modelo específico de la plataforma.

RF8.Modelar PIM: El sistema debe permitir que los diagramas creados puedan ser modelados en el modelo independiente de la plataforma.

RF9.Transformar diagrama de clases: Se encarga de realizar la transformación del diagrama de clases seleccionado, al modelo específico de la plataforma java.

RF10.Transformar diagrama de caso de uso: Se encarga de realizar la transformación del diagrama de caso de uso seleccionado, al modelo específico de la plataforma java.

RF11.Transformar diagrama de secuencia: Se encarga de realizar la transformación del diagrama de secuencia seleccionado, al modelo específico de la plataforma java.

RF12.Transformar diagrama de estado: Se encarga de realizar la transformación del diagrama de estado seleccionado, al modelo específico de la plataforma java.

RF13.Transformar diagrama de actividades: Se encarga de realizar la transformación del diagrama de actividades seleccionado, al modelo específico de la plataforma java.

RF14.Modelar PSM: El sistema debe permitir que los diagramas transformados desde el modelo (PIM) puedan ser modificados en el modelo (PSM).

RF15.Configurar diagrama: En este requisito funcional abarca las funcionalidades básicas que debe tener una herramienta CASE:

RF15.1 Cambiar color de fondo de las formas.

RF15.2 Cambiar el tipo de letra.

RF15.3 Cambiar el tamaño de la letra.

RF15.4 Redimensionar una forma.

RF15.5 Copiar una forma y pegarla en el diagrama.

RF15.6 Enviar al fondo.

RF15.7 Enviar hacia el frente.

Entre otras muchas funcionalidades que incorpora esta versión de la herramienta y que facilita la interacción del usuario con la misma.

RF16.Guardar diagrama: El sistema permite guardar los diagramas creados en tres formatos principales (.png, .mxe y .jpg). Garantizando así que el trabajo realizado en un diagrama no se pierda.

RF17.Imprimir diagrama: Permite realizar una impresión del diagrama seleccionado en ese momento.

2.2 Requisitos no funcionales del sistema

RNF 1: Interfaz del sistema: Una de las fortalezas de la herramienta "jMDA" debe ser la interfaz gráfica, incorporándole una mejor distribución de los componentes.

RNF 2: Usabilidad: La aplicación debe poseer una elevada velocidad de operación, no debe presentar errores de implementación y su diseño debe permitir que el tiempo de esfuerzo requerido por los usuarios para adaptarse sea mínimo.

RNF 3: Rendimiento: El sistema deberá ser eficiente en las transformaciones realizadas en cuanto al tiempo de duración y la calidad de estas.

RNF 4: Portabilidad: El sistema podrá ser utilizado en cualquier plataforma para la que esté disponible la Máquina Virtual de Java en su versión 1.7.0 o posterior.

RNF 4: Requerimiento del Hardware: La aplicación necesita para su ejecución una terminal con al menos 100MB de memoria RAM disponibles para su uso eficiente. Con microprocesador de la familia Pentium IV o superior. No necesita tarjeta de video extra.

2.3 Diagrama de Casos de Uso del sistema

En la Figura 7 se muestra el diagrama de caso de uso del sistema de la herramienta. Este diagrama tiene un caso de uso principal ""Realizar transformación al modelo PSM".

Sobre el caso de uso recae un gran peso, pues es el encargado de realizar las transformaciones de los diagramas modelados en el *PIM* al modelo *PSM,* haciendo uso de los algoritmos de transformación implementados. Esta es la funcionalidad básica de esta versión de la herramienta.

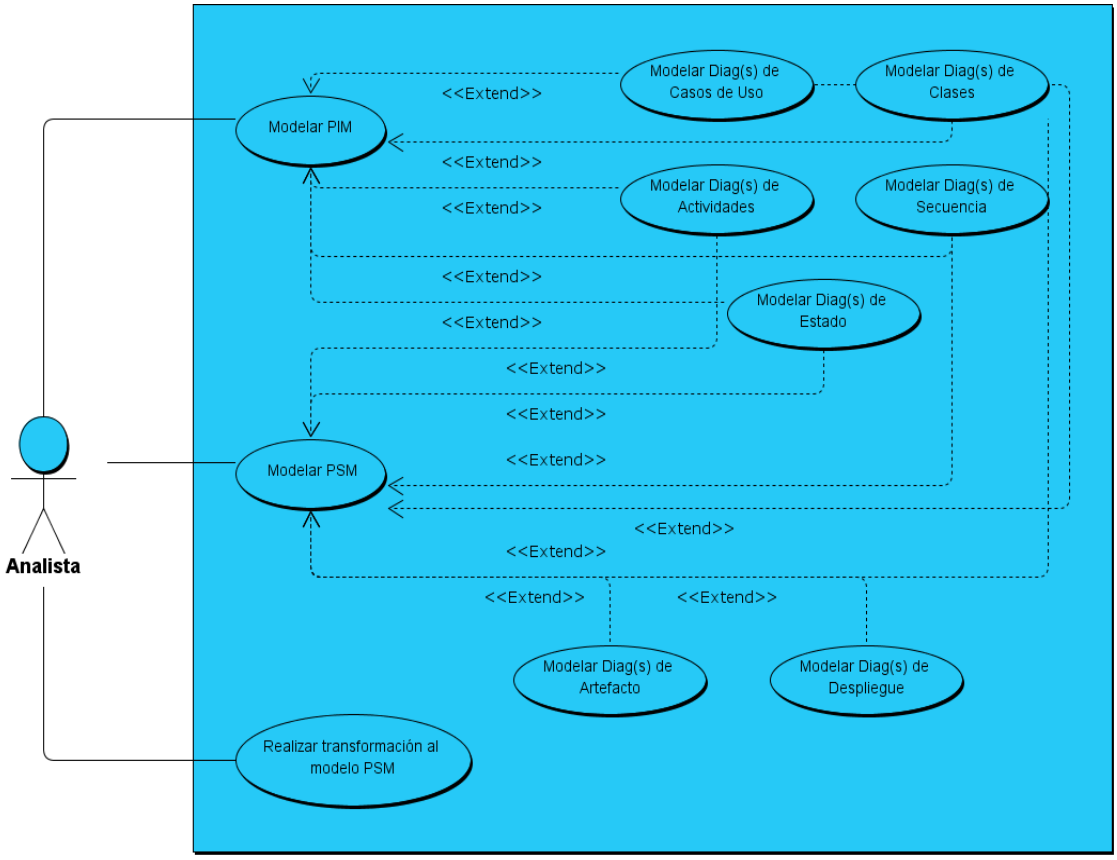


Figura7: Diagrama de casos de uso

Fuente: Elaboración propia

2.4 Relación entre los requisitos funcionales y los casos de uso del sistema.

En la tabla que se muestra a continuación se especifica la relación que existe entre los casos de uso del sistema y los requisitos funcionales que le da cumplimiento.

Tabla 6: Matriz de correlación entre requisitos funcionales y casos uso del sistema.

Fuente: Elaboración propia.

Requisitos Funcionales	Modelar PIM	Modelar PSM	Realizar Transformación
RF1	X		
RF2	X		
RF3	X		
RF4	X		
RF5	X		
RF6		X	
RF7		X	
RF8	X		
RF9			X
RF10			X
RF11			X
RF12		X	X
RF13		X	X
RF13.1	X	X	

RF13.2	X	X	
RF13.3	X	X	
RF13.4	X	X	
RF13.5	X	X	
RF13.6	X	X	
RF13.7	X	X	
RF14	X	X	
RF15	X	X	
RF16	X	X	
RF17	X	X	

2.5 Descripción de los casos de uso significativos

En este epígrafe se muestra la descripción de los Casos de Uso del Sistema más significativos.

Caso de uso: Realizar Transformación

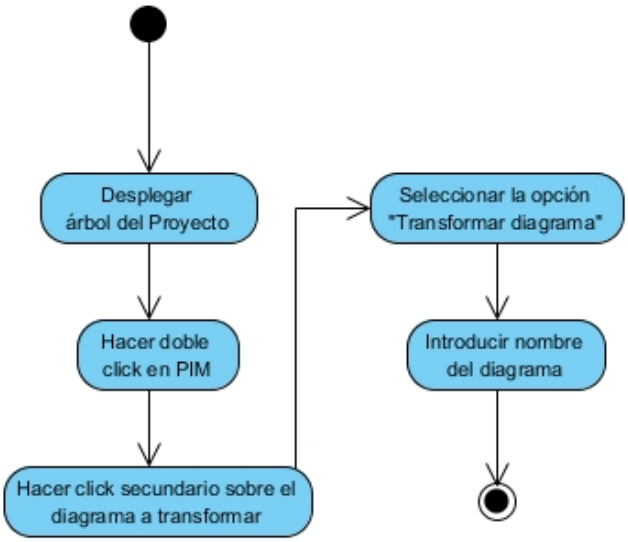


Figura8: Diagrama de actividades caso de uso "Realizar transformación".

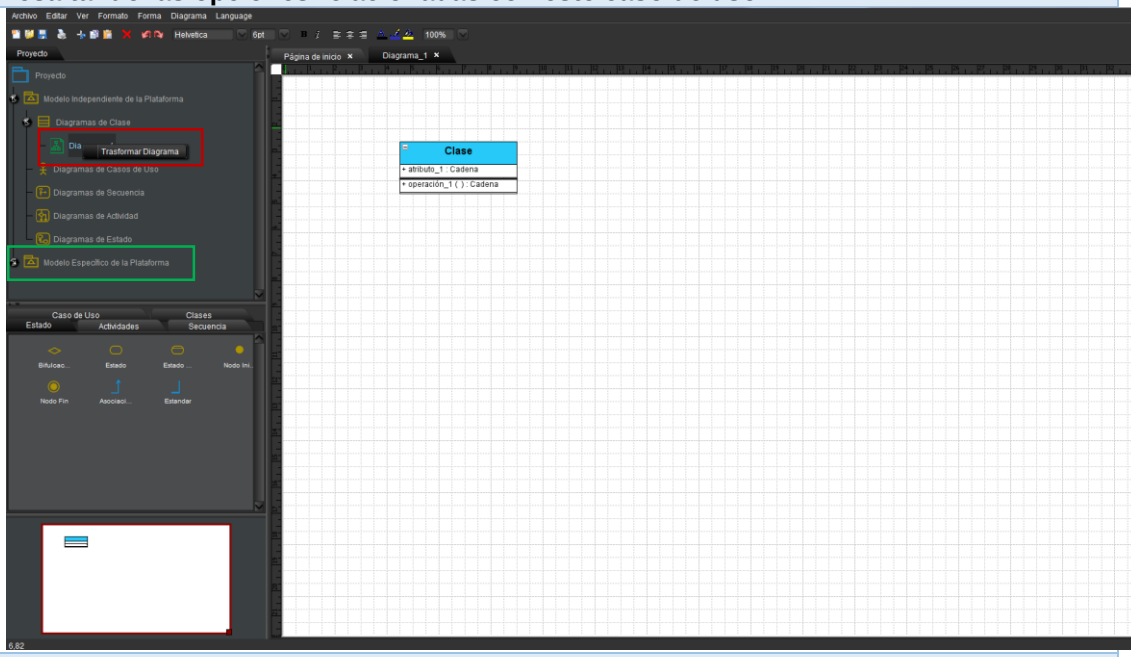
Fuente: Elaboración propia

En la Figura 10 se muestran las acciones ejecutadas por el analista para realizar la transformación de un diagrama de actividades. Es importante destacar que cuando se realizan las transformaciones el sistema pide un nombre al analista, este nombre no debe coincidir con ninguno introducido anteriormente.

Tabla 7 Descripción del Caso de Uso del Sistema "Realizar Transformación".

Fuente: Elaboración propia.

Caso de uso del sistema	Transformar diagrama
Actor	Analista
Propósito	Realizar la transformación de los diagramas correspondientes al modelo específico de plataforma (java).

Resumen	Inicia cuando el analista selecciona el diagrama a transformar del modelo específico de la plataforma y hace *click* secundario sobre este. Se muestra un menú con la opción "transformar diagrama", el cual luego de ser seleccionado llama al método encargado de realizar la transformación y crea el nodo correspondiente en el modelo *(PSM).*
Responsabilidades	Crear una hoja de trabajo en el nodo seleccionado en el modelo *(PSM)* y modelar un diagrama en esta.
Casos de uso asociados	Realizar Transformación
Requisitos especiales	El nombre introducido por el analista debe ser único, pues el sistema no permite que existan dos diagramas con el mismo nombre.
Prototipo de interfaz: En la figura se muestra el prototipo de interfaz de la aplicación, resaltando las opciones relacionadas con este caso de uso.	
	
Flujo normal de los eventos	
Acción del actor	Respuesta del sistema
1. El analista hace *click* secundario sobre un diagrama en el modelo (PIM) en el árbol del proyecto.	2. El sistema le muestra un menú con la opción "Transformar diagrama".
3. El analista selecciona esta opción.	5. El sistema le muestra una ventana para que introduzca el nombre que tendrá el diagrama transformado.
6. El analista introduce el nombre del diagrama.	7. El sistema verifica que el nombre no coincide con ninguno de los diagramas anteriormente creados realiza las transformaciones en dependencia del tipo de diagrama que sea y añade un nodo en el árbol del proyecto que referencia al diagrama transformado.
8. El analista hace doble *click* sobre la referencia creada en el árbol del proyecto.	9. El sistema carga la hoja de trabajo que contiene el diagrama transformado.
Flujo alternativo de eventos	
	7.1. El sistema encuentra una coincidencia con el nombre del diagrama.
	7.3. Muestra una notificación que explica el error provocado.
	7.4. Muestra una ventana para que introduzca un nombre nuevamente.

7.5. El analista introduce el nombre del diagrama.	7. El sistema verifica que el nombre no coincide con ninguno de los diagramas anteriormente creados realiza las transformaciones en dependencia del tipo de diagrama que sea y añade un nodo en el árbol del proyecto que referencia al diagrama transformado.
8. El analista hace doble *click* sobre la referencia creada en el árbol del proyecto.	9. El sistema carga la hoja de trabajo que contiene el diagrama transformado.

2.6 Diagrama de paquetes de la aplicación.

La figura 11 muestra el diagrama de paquetes de la herramienta jMDA PIM-PSM v5.0. Esta versión de la herramienta se diseña de modo tal que sea fácil la creación de nuevos módulos en versiones posteriores. Existen cinco paquetes que tienen gran importancia en el diseño de esta versión ellos son:

- ✓ El paquete "*mxGraph*" que contiene todas las clases de esta biblioteca, pieza clave en el proceso de diagramación e interacción del usuario con los diagramas creados.
- ✓ El paquete "*Shape*" contiene las formas implementadas que el usuario utiliza para la diagramación, cabe destacar que este paquete se subdividió en un paquete específicos para cada uno de los modelos implementados de modo tal que si se desea incorporar alguna forma nueva se realicen las modificaciones pertinentes en un solo paquete.
- ✓ El paquete "*Tree*" contiene las clases encargadas de manejar la lógica del árbol del proyecto, así como la visualización de cada uno de sus componentes.
- ✓ El paquete "Transformaciones", en él se encuentran las clases encargadas de realizar las transformaciones del modelo *PIM* al modelo *PSM*, el diseño de estas clases se realizó de modo tal que exista un clase para cada diagrama y que contengan un método de transformación, con esto se logra que si existe la necesidad de modificar la lógica de una transformación determinada no se afecten las restantes transformaciones.
- ✓ El paquete "Editor" contiene toda la parte visual de la aplicación. En este paquete se crean todos los componentes visuales del sistema y la programación de las acciones de cada uno de ellos.

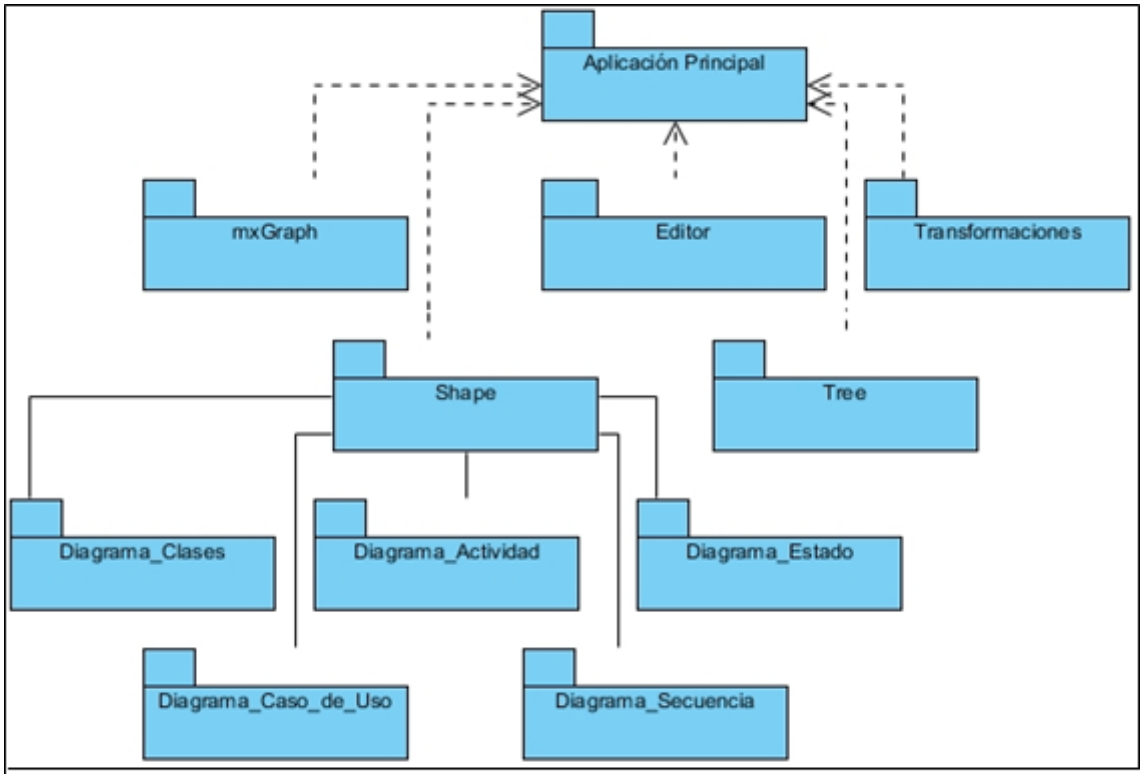


Figura9: Diagrama de paquetes de la herramienta jMDA PIM-PSM v4.0.

Fuente: Elaboración propia

2.7 Algoritmo de transformación. Definición y reglas.

En el capítulo uno se plantea que una transformación o *mapping MDA* proporciona la especificación de la transformación de un *PIM* en un *PSM* para una plataforma determinada y que se distinguen dos tipos de se distinguen dos tipos de definiciones de transformaciones según (Mora, 2006).

A continuación se definen las principales reglas de transformación utilizada en la herramienta:

Reglas mantenidas de la versión anterior.

- **Regla 1:** Todas las clases del modelo *PIM* de tipo (1) se transforman en el modelo *PSM* en clases de tipo (2).

Las clases del modelo *PIM* luego de realizar la transformación se convierten en la clase modelo en el *PSM*.

Ejemplo:

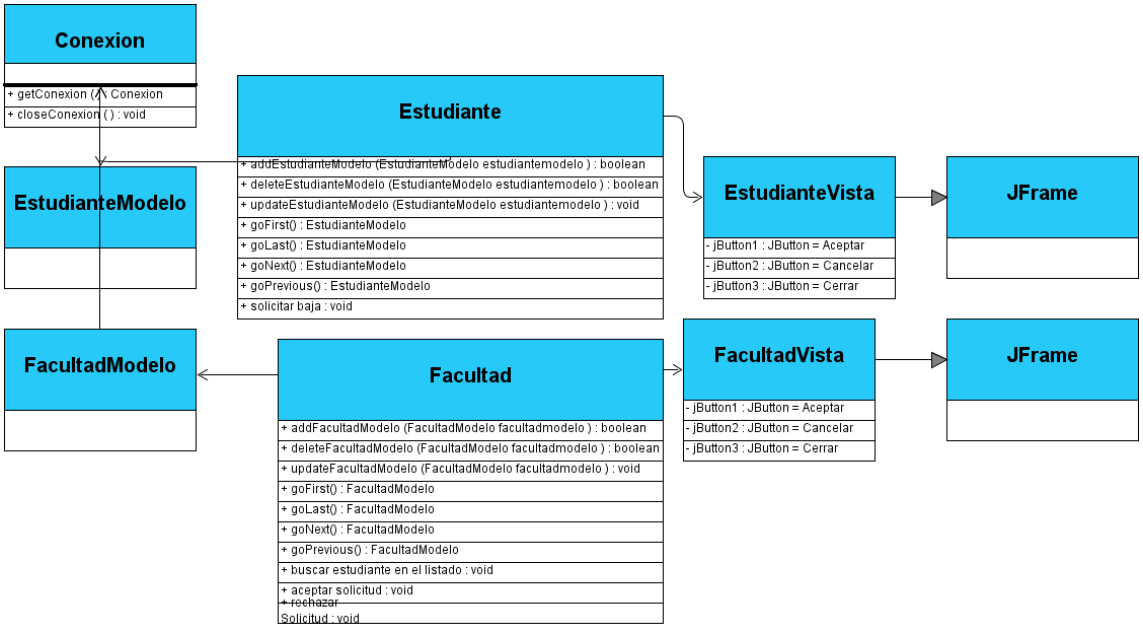


Figura9: Diagrama de clases en el modelo PSM

Fuente: Elaboración propia

- **Regla 2:** Cuando una clase en el modelo *PIM* contiene atributos se aplica el MVC y se generan las clases correspondientes en el modelo *PSM. En el caso de los diagramas de clases.*

En esta versión de la herramienta se aplica el patrón de diseño MVC para realizar las transformaciones, y en consonancia con esto cuando una clases del modelo *PIM* contiene atributos se generan las clases correspondientes en el modelo *PSM* al MVC.

Si una clases del modelo *PIM* no contiene atributos no tiene sentido aplicar el patrón de diseño antes mencionado ya que no sería necesario tener ninguna vista para manejar la información asociada a ella, es por eso que se decide aplicar este patrón a las clases que contengan atributos.

- **Regla 2.1:** A los atributos de la clase del modelo *PIM* se le cambia el tipo para la plataforma java en el *PSM.*

Los atributos del modelo *PIM* tienen un tipo de dato genérico puesto que no se define pensando en una plataforma específica, cuando se realiza la transformación estos atributos toman los tipos de datos de la plataforma java que es la que se implementa en esta versión de la herramienta.

- **Regla 2.2:** A la clase modelo creada en el *PSM* se le agregan los métodos *set* y *get* correspondientes a los atributos que tiene y en función de la plataforma java.

Cuando se realiza la trasformación se generan de forma automática en la clase modelo los métodos *set* y *get* correspondientes a los atributos declarados. Estos métodos se declaran en función de los tipos asociados a los atributos en el modelo *PSM.*

- **Regla 3:** Los métodos de la clase del modelo *PIM* se pasan para la clase controladora creada.

Los métodos que describen las responsabilidades de las clases en el modelo *PIM* son transferidos a la clase controladora correspondiente, pues esta es la que se encarga de realizar estas operaciones.

- **Regla 4:** En la clase vista se añade la declaración de cada uno de los componentes que tendrá la misma, en función de los atributos de la clase del modelo *PIM*.

Se incorpora en la clase vista que se asocia a la clase controladora, una declaración de los principales componentes que debería tener esta para el manejo de los atributos de la clase modelo, se decide que si un atributo recibe un valor predefinido no se asocie ningún componente a este pues no será necesario manejar la información asociada a él.

Reglas elaboradas para la nueva versión

- **Regla 5:** Se crea una sola clase conexión que se relaciona con todas las clases modelos y que contiene el método de conexión a las base de datos.

Como las transformaciones se realizan enfocadas hacia en un sistema de información, se crea una clase conexión que será la encargada de mantener una referencia hacia donde se guarda la información. Esta clase tiene un método *getConexion()* que se encarga de retornar la conexión a la base de datos demás del método closeConexion() que es el que se encarga de cerrar la conexión.

- **Regla 6:** Se le crea a cada clase controladora en todos los diagramas los métodos goFirst, goLast, goNext y goPrevious.

Para el mejor manejo de los diagramas se crean estos métodos que son para moverse por los registros con más rapidez.

Para realizar las trasformaciones de los diagrama de clases del modelo *PIM* al modelo *PSM* se diseña un algoritmo con alto grado de complejidad. En la figura 12 se muestra el algoritmo de transformación de diagramas de clases, haciendo uso de la notación para el modelado de procesos *(BPMN)* por sus siglas en inglés, para lograr una mejor comprensión de este.

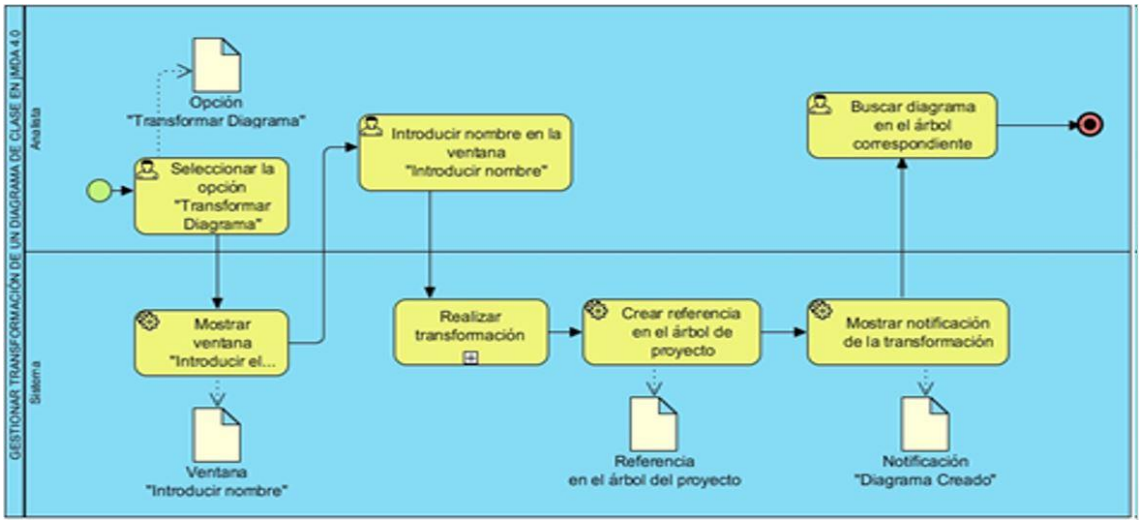


Figura10: Definición del algoritmo de transformación de diagramas de clases utilizando la notación BPMN. Fuente: Elaboración propia

Como se puede apreciar en la Figura 10 el proceso inicia cuando el analista selecciona la opción “transformar diagrama” y luego de introducir el nombre que tendrá el diagrama creado el sistema comienza a realizar la transformación. El subproceso “Realizar transformación” describe de forma detallada las acciones llevadas a cabo por el sistema para transformar el diagrama.

2.8 Descripción del proceso de implementación.

El módulo *PIM-PSM* v5.0 de la herramienta *CASE jMDA* desarrollado en la presente tesis tiene en común con todas las versiones anteriores el uso del lenguaje *Java*, seleccionado por las facilidades que brinda al estar respaldado por licencias que permiten su libre distribución, modificación y uso. Además se considera mantener a *Java* como la Plataforma Específica de Modelado en esta versión.

Para el desarrollo de esta versión resulta necesario mantener la estrategia de implementación utilizada en la versión 4.0 de la aplicación, debido a que el diseño utilizado en esta versión permite el cumplimiento de los objetivos trazados en esta.

Para la codificación se utilizaron bibliotecas contenidas en el *JDK* versión 8 tales como: *swing* y *awt*, para manipular todos los componentes visuales de la aplicación. Se emplea el entorno integrado de desarrollo *NetBeans* por las facilidades que brinda para refactorización y chequeo semántico del código.

2.9 Diagrama de clases del paquete Transformación

En este paquete se encuentran las clases encargadas de realizar las transformaciones de cada uno de sus correspondientes diagramas. Estas clases son llamadas por la clase “*createTree*” cuando se selecciona la opción “Transformar Diagrama”. Su principal

responsabilidad es realizar la transformación correspondiente al modelo *PSM* del diagrama seleccionado.

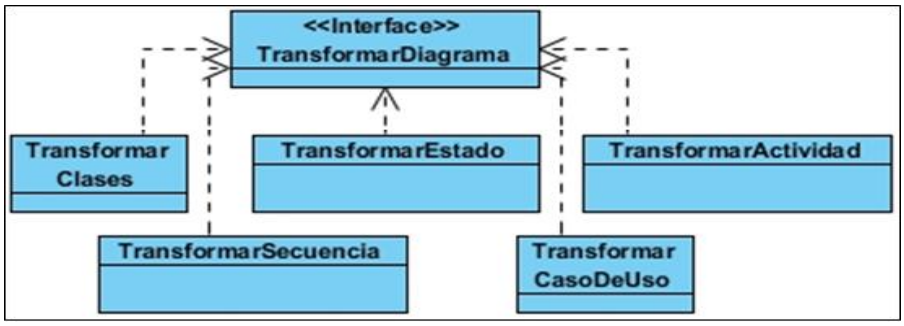


Figura 11: Diagrama de clases del paquete "Transformaciones". Fuente: Elaboración propia

En la Figura 16 se muestra el diagrama de clases del paquete “Transformaciones”, siguiendo la lógica anteriormente planteada y aplicando los principios de modularidad, se crea una clase para realizar la transformación a cada tipo de diagrama modelado en el *PIM.* Esto permite definir una transformación para cada uno de los modelos y simplificar el proceso de realizar modificaciones a las transformaciones implementadas.

2.10 Diagrama de secuencia del caso de uso Realizar Transformación

El siguiente diagrama de secuencia describe las acciones del analista y la respuesta del sistema cuando se selecciona la opción “Transformar diagrama”.

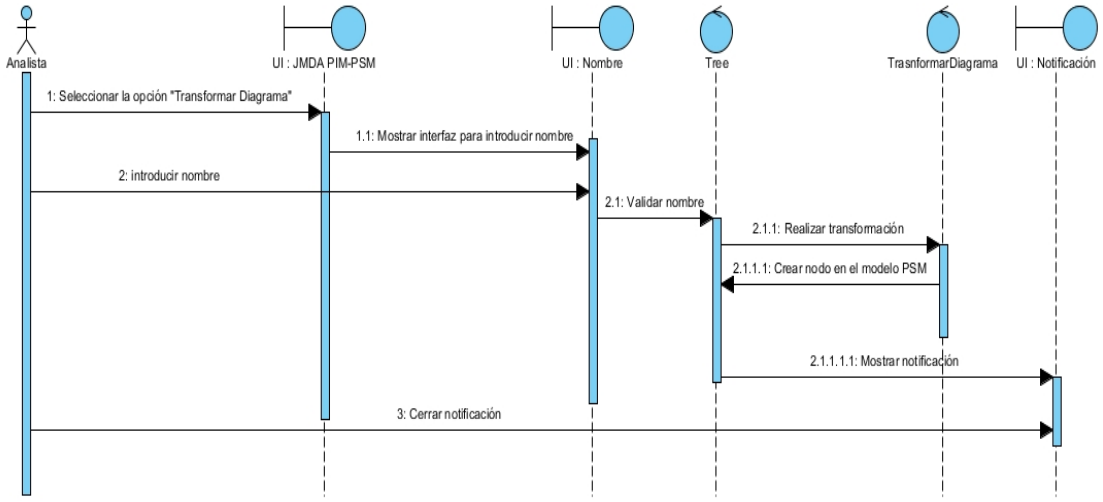


Figura 12: Diagrama de secuencia que describe la opción "Transformar diagrama". Fuente: Elaboración propia.

En la Figura 12 se puede apreciar el diagrama de secuencias que describe las acciones del analista y la interacción de este con el sistema cuando se selecciona la opción “Transformar diagrama”. Existe una relación muy interesante entre dos clases presentes en este diagrama, pues la clase “*createTree*” es la que se encarga llamar a la clase correspondiente para transformar el diagrama y espera el diagrama transformado;

después crea el nodo correspondiente en el árbol del proyecto y guarda el diagrama con la referencia creada.

2.11 Tratamiento de errores

La validación de los datos es de vital importancia en cualquier software, esto permite garantizar que toda la información que se registre en el sistema sea válida.

En la figura 13 se muestra el ejemplo de un diagrama que tiene deshabilitados los botones de modificar y eliminar, puesto que hasta que no haya un atributo o una operación agregada estos botones no se podrán utilizar.

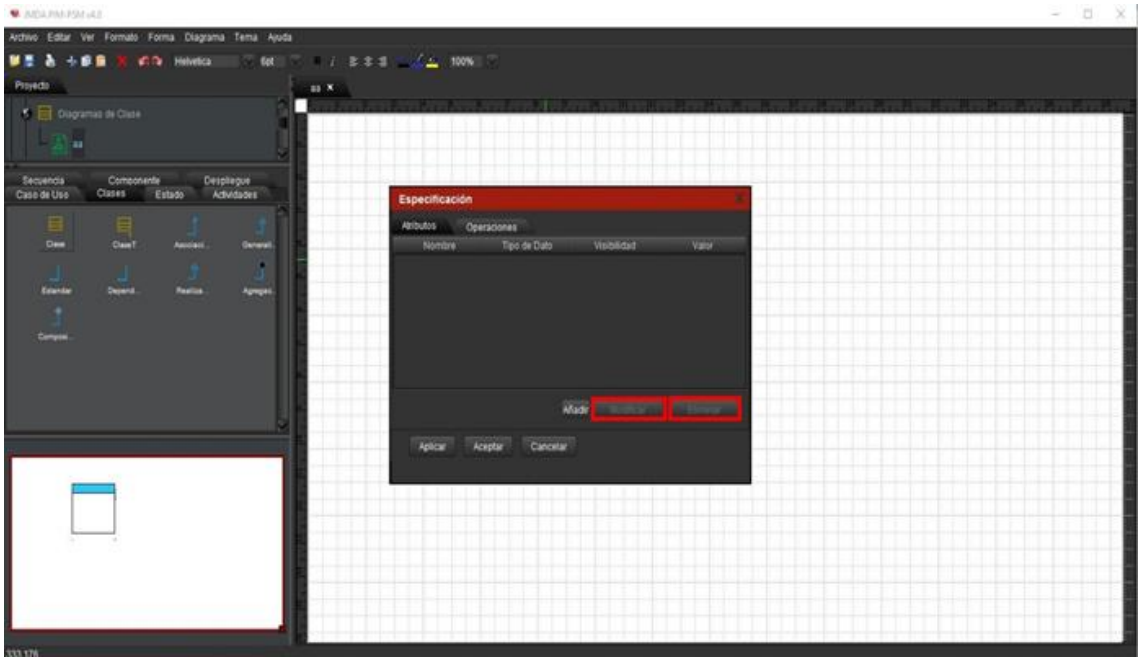


Figura 13: Notificación de error. Fuente: Elaboración propia

Como se puede ver en la figura 13 los errores de este tipo están corregidos pues el usuario no podrá utilizar botones que dependan de una acción precedente como en este caso se ve claramente pues los botones modificar y eliminar están deshabilitados porque dependen de la acción añadir pues hasta que esta no sea ejecutada estos botones no se activarán.

2.12 Diagrama de componentes

En la Figura 22 se muestra el modelo de componentes del sistema.

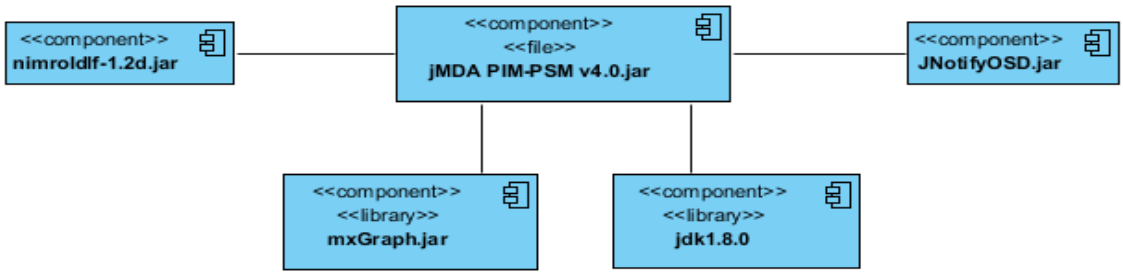


Figura 14: Diagrama de componentes de la herramienta CASE JMDA PIM-PSM 4.0.

Fuente: Elaboración propia

Como se puede observar en la Figura 14 el sistema se relaciona con varios componentes externos. El componente "nimroldf-1.2d.jar" es utilizado como "*lookandfeel"* del sistema, permitiendo que la interfaz de la aplicación tenga mejor apariencia visual. El componente "*jNotifiOSD.jar*" es usado para las notificaciones ya que con su uso se pueden crear notificaciones más agradables para el usuario y el componente *"mxGraph"* que permite optimizar la diagramación en el modelo *PIM* y *PSM.*

2.13 Conclusiones del capítulo

1. Se definieron los requisitos funcionales y no funcionales del sistema, determinando la correspondencia entre los requisitos funcionales y los casos de uso del sistema.
2. Se diseñó el diagrama de casos de uso; describiendo cada uno de los casos significativos y se describió el diagrama de paquetes que estructura el sistema, y se comenta cada uno de sus componentes para lograr un mejor entendimiento.
3. Se definió un grupo de reglas tomadas en cuenta a la hora de realizar las transformaciones del modelo PIM al modelo *PSM*.
4. Se diseñó el algoritmo de transformación que se utilizará en la conversión del diagrama de clases del modelo *PIM* al modelo *PSM* además se realizo el diseño y descripción de los diagramas de clases; separándolos según el paquete para lograr una mejor comprensión.
5. Se describieron los casos de uso significativos de la aplicación por medio de diagramas de secuencia.
6. Se analizó el procedimiento requerido para la realización del tratamiento de errores.

7. Se diseñó el diagrama de componentes que define la arquitectura del sistema tal y como fue implementado.
8. La implementación se realizó en base al lenguaje Java utilizando la librería mxgraph con el NetBeans IDE 8.0.1 y el JDK 8.

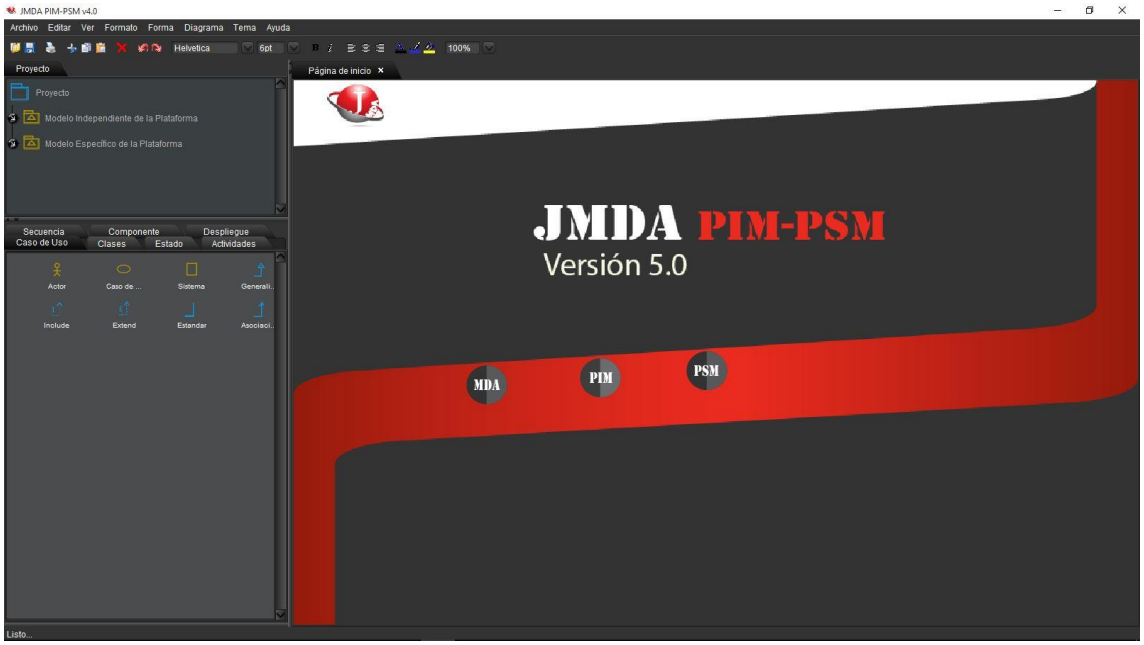
Capítulo 3

Capítulo 3: *"Manual de usuario para la implementación del módulo PIM-PSM 5.0 de la herramienta CASE jMDA."*

En este capítulo se realizara un manual de usuario de la aplicación donde se mostraran las nuevas funcionalidades de la misma basándose en diferentes ejemplos.

3.1 Vista principal de la aplicación

En este diagrama se muestra la vista principal de la aplicación después de iniciada donde se puede ver la barra de menú principal en la parte superior con opciones básicas como cambiar fondo, letra, imprimir, cambiar color de fondo etc.



3.2 Área de trabajo

En este diagrama se muestra el área de trabajo de la aplicación donde se puede destacar tres áreas principales que serían el árbol del proyecto que es donde se crean los nuevos diagramas y se pueden abrir otros que se hayan creado también está la paleta de componentes que son todos los componentes a utilizar para la elaboración de los diagramas, y la otra área importante es el área de trabajo donde se puede trabajar con los diagramas que estén en elaboración.

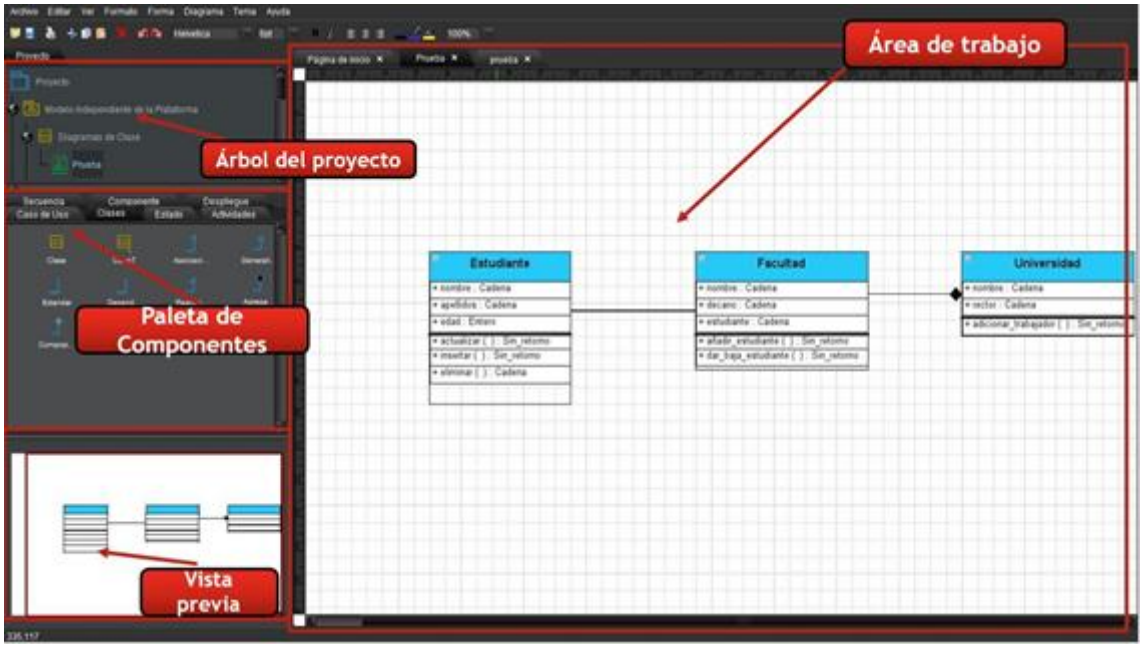


Figura 16: Área de Trabajo

3.3 Creación de los diagramas

En este diagrama se puede apreciar cómo se crea un diagrama lo que se hace en el árbol del proyecto en este ejemplo se ve cómo se crea un diagrama de clases pero asi mismo se crea cualquier tipo de diagrama.

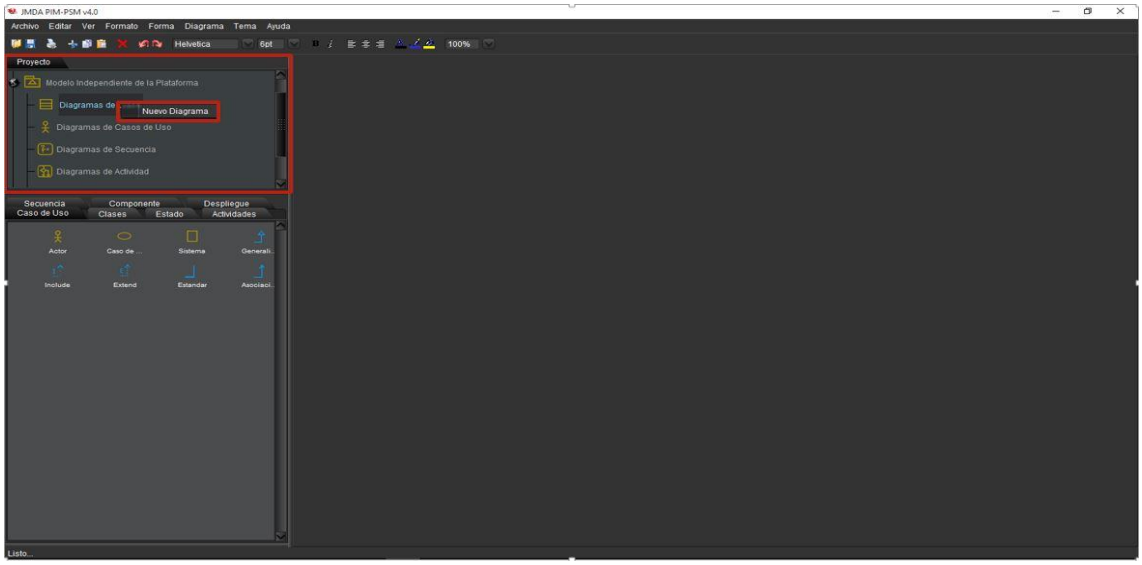


Figura 17: Creación de diagramas

3.4 Transformación de diagramas

En este diagrama se puede apreciar cómo se transforma un diagrama de clase que como se puede apreciar es en el árbol del proyecto y para transformar cualquier tipo de diagrama se procede de esta misma manera.

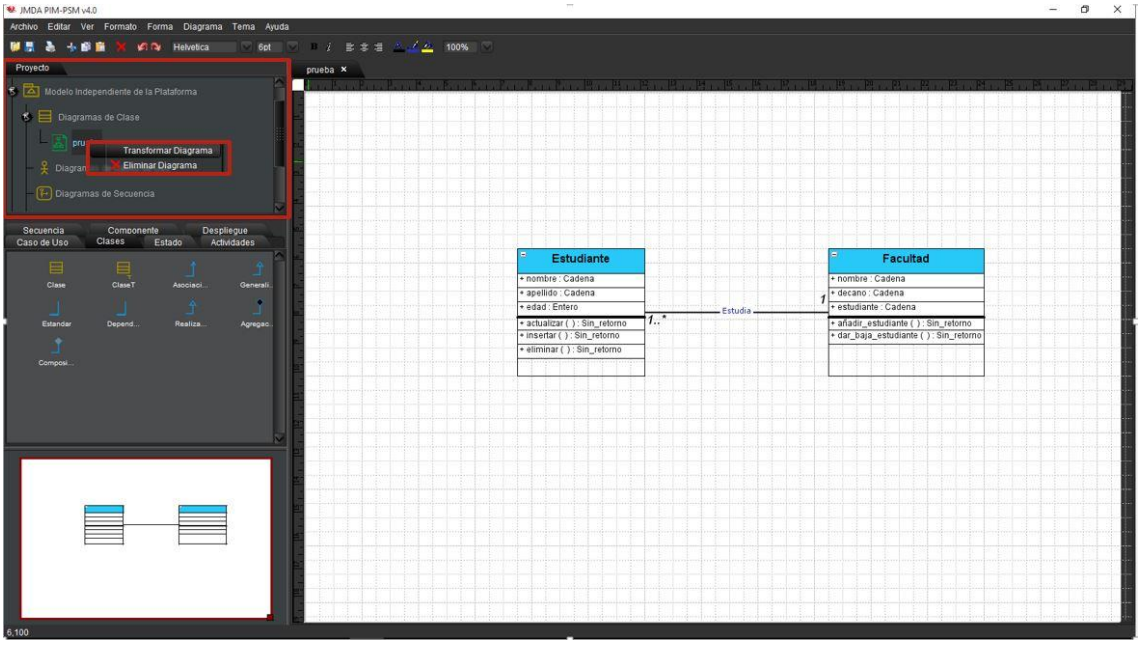


Figura 18: Transformación de diagramas

3.5 Utilización de los componentes

En el diagrama se muestra como se utilizan los componentes que solo es hacer clic izquierdo sobre ellos y arrastrarlo hacia el área de trabajo y se procede de igual forma para todos los tipos de diagramas

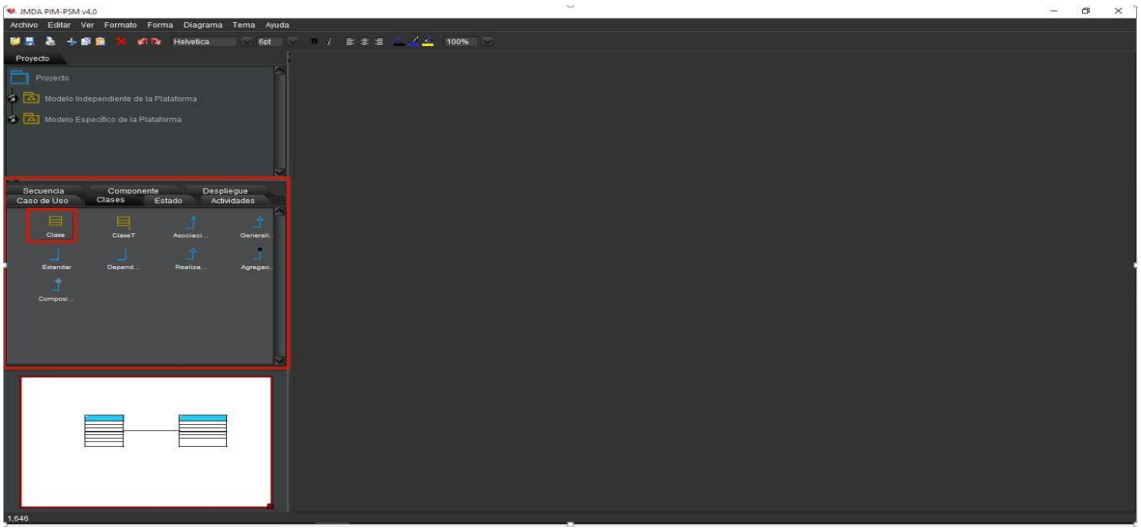


Figura 19: Utilización de los componentes

3.6 Modificación de componentes

En el diagrama se muestra como se modifica un componente lo cual se hace en el área de trabajo donde se le da clic derecho al componente que se desea modificar se va a la opción especificación y le saldrá un menú donde estará habilitadas las opciones que usted puede modificar en este caso es para un diagrama de clases pero se procede de la misma forma con todos los tipos de diagramas.

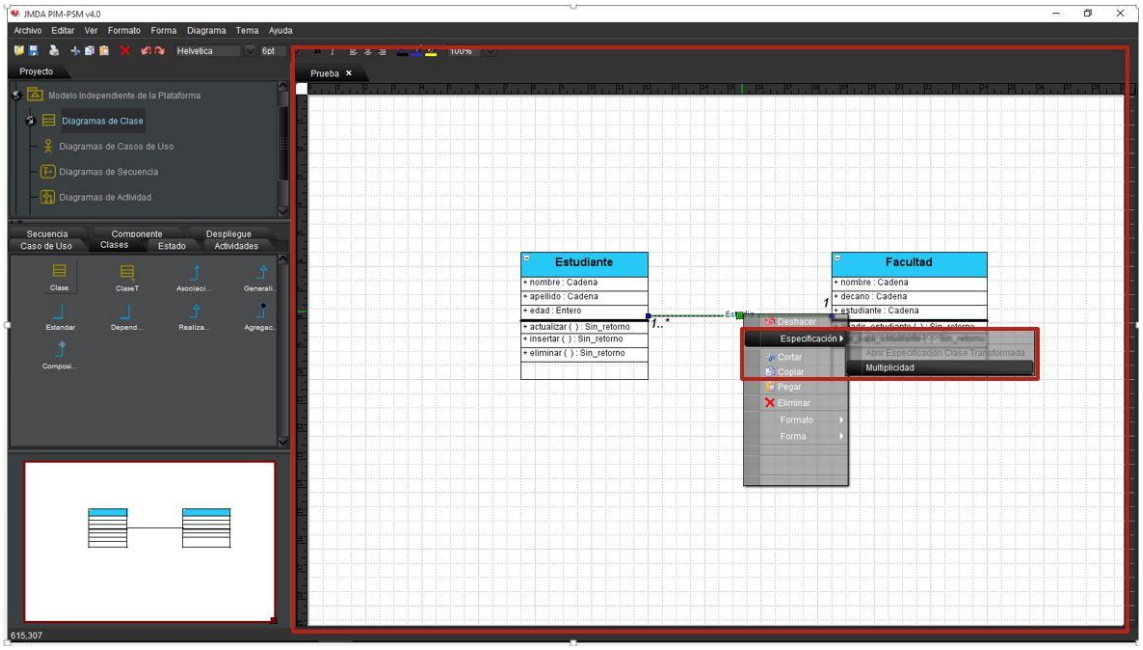


Figura 20: Modificación de componentes

3.7 Caso de Estudio para mostrar el uso

En una facultad se realiza el proceso de inserción de los estudiantes a un sistema para el mejor control de los mismos donde los estudiantes tendrán que tener su nombre, apellido y edad y la facultad tendrá que tener el nombre del decano el estudiante además de otros nombres de interés como el de la secretaria etc. La secretaria es la capacidad para operar con el sistema.

3.8 Diagrama de clases del caso de estudio modelado en PIM

En este diagrama se muestra un ejemplo de un diagrama de clases que tiene dos clases que son estudiante y profesor cada clase con sus métodos y atributos, modelado en PIM.

Figura 21: Diagrama de Clases PIM

Fuente: Elaboración propia

3.9 Diagrama de clases modelado en PSM

En este diagrama se muestra un ejemplo de un diagrama de clases modelado en el PSM utilizando el criterio de transformación del MVC.

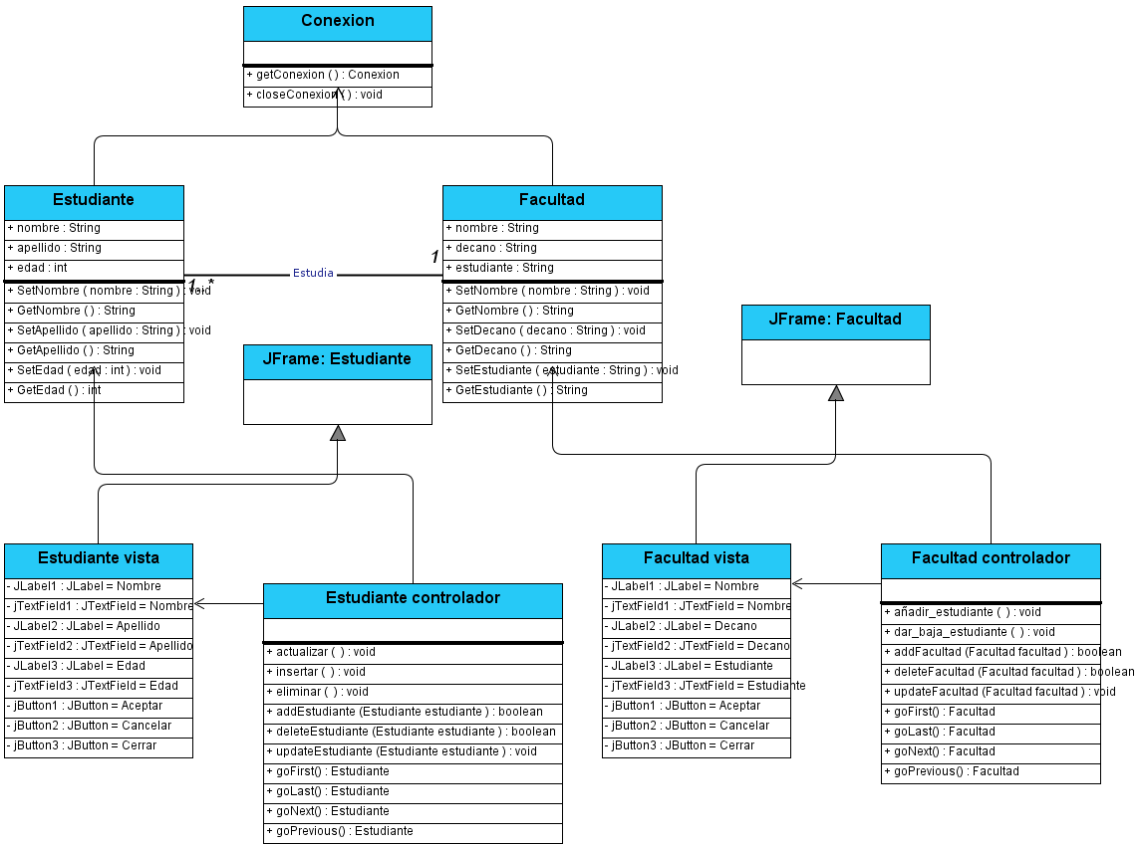


Figura 22: Diagrama de Clases PSM

Fuente: Elaboración propia

3.10 Diagrama de caso de uso modelado en PIM

En este diagrama se muestra un caso de uso donde intervienen una secretaria de una facultad que tiene como tarea la gestión de estudiantes.

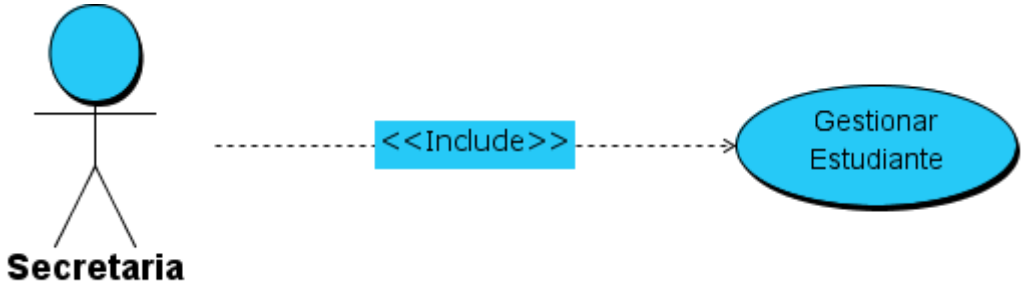


Figura 23: Diagrama de caso de uso PIN

Fuente: Elaboración propia

3.11 Diagrama de caso de uso modelado en PSM

En este diagrama se muestra el caso de uso del diagrama anterior modelado en el PSM y se puede apreciar cómo no tiene cambios porque no tiene relevancia para el analista desde el punto de vista de la codificación.

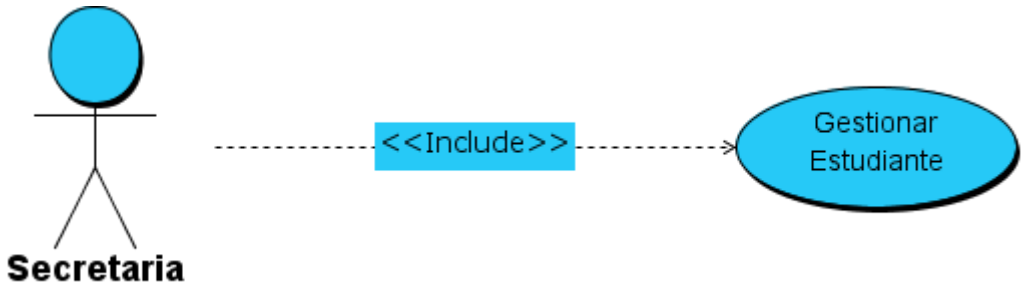


Figura 24: Diagrama de caso de uso PSM

Fuente: Elaboración propia

3.12 Diagrama de actividad modelado en PIM

En este diagrama se muestra cómo quedaría un diagrama de actividad modelado en PIM en el que se le pretende dar baja a un estudiante.

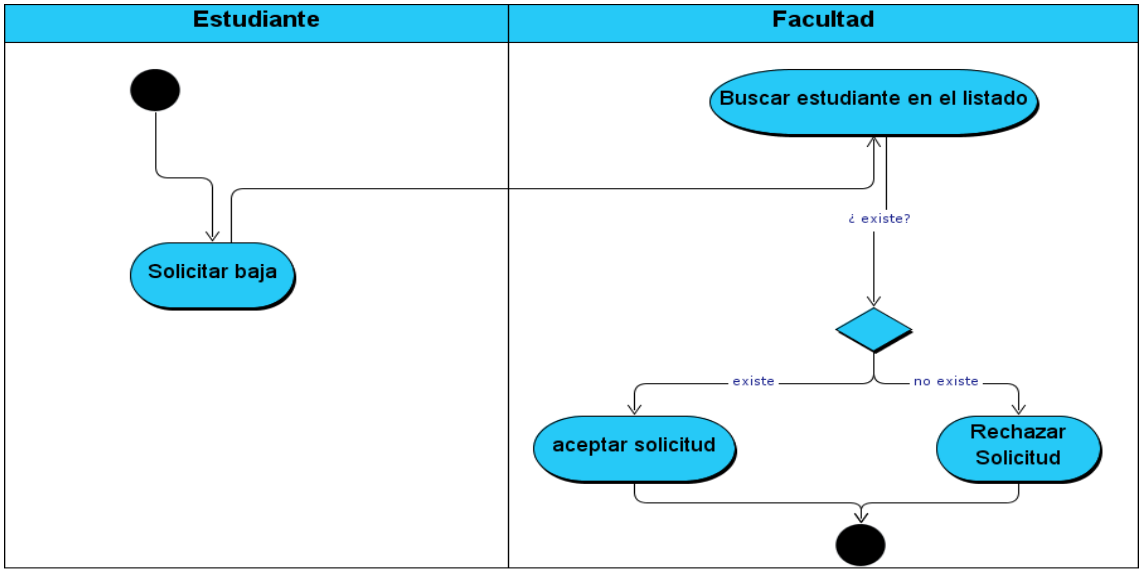


Figura 25: Diagrama de actividad PIM

Fuente: Elaboración propia

3.13 Diagrama de actividad modelado en PSM

En este diagrama se muestra el mismo que en la figura anterior pero modelado en el PSM.

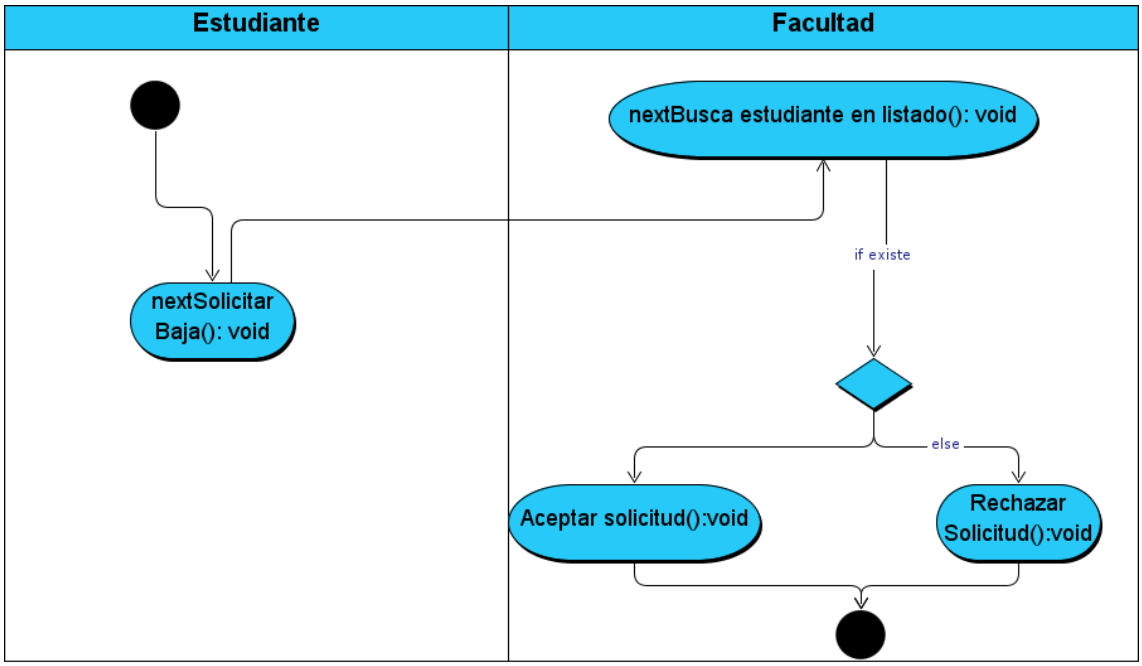


Figura 26: Diagrama de actividad PSM

Fuente: Elaboración propia

3.14 Diagrama de secuencia modelado en PIM

En este diagrama se muestra un diagrama de secuencia modelado en PIN donde se pretende dar baja a un estudiante.

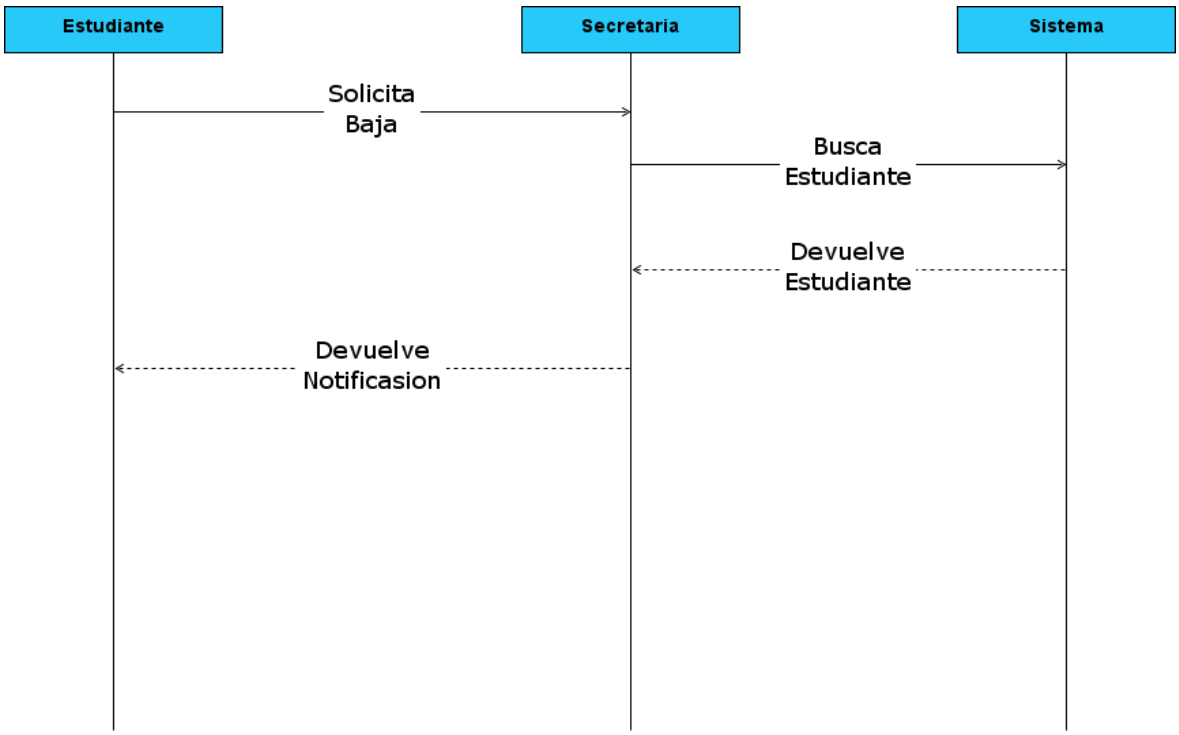


Figura 27: Diagrama de Secuencia PIM

Fuente: Elaboración propia

3.15 Diagrama de secuencia modelado en PSM

En este diagrama se muestra el mismo que la figura anterior pero modelado en el PSM

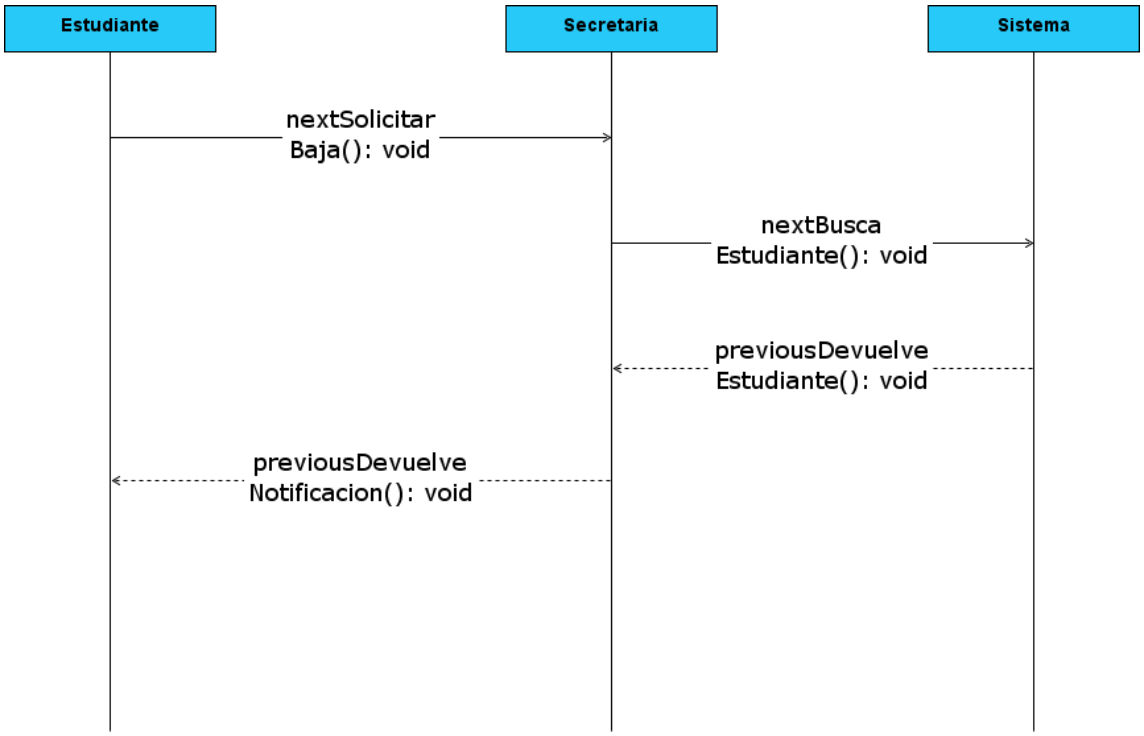


Figura 28: Diagrama de Secuencia PIM

Fuente: Elaboración propia

3.16 Diagrama de estado modelado en PIM

En esta figura se muestra un diagrama de estado modelado en PIN donde se le pretende dar baja a un estudiante.

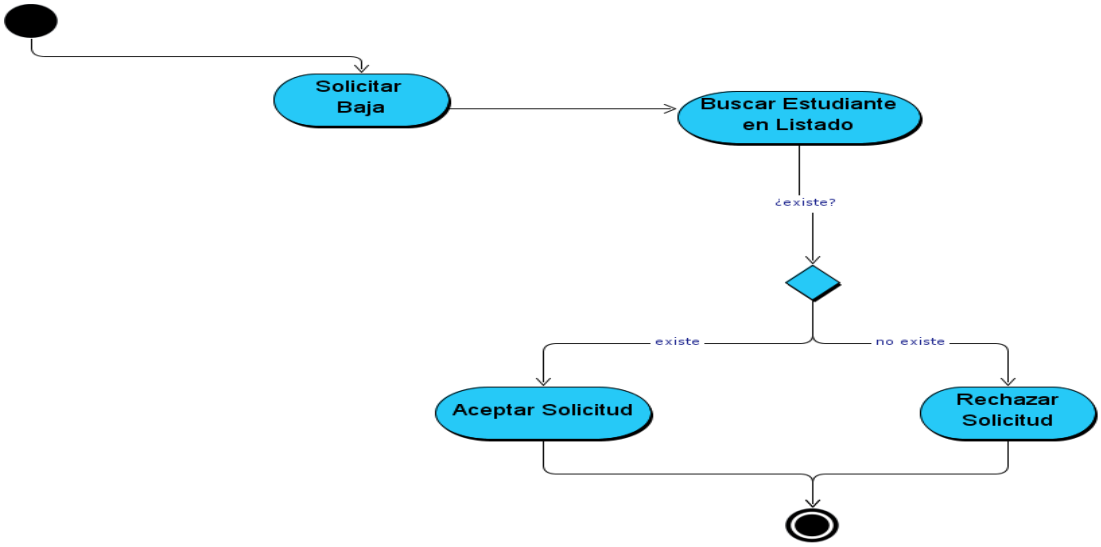


Figura 29: Diagrama de Estado PIM

Fuente: Elaboración propia

3.17 Diagrama de estado modelado en PSM

En esta figura se muestra un diagrama de estado modelado en PIM donde se le pretende dar baja a un estudiante el cual no sufre ningún cambio con respecto al modelado en PIN porque no tiene relevancia para el analista desde el punto de vista de codificacion.

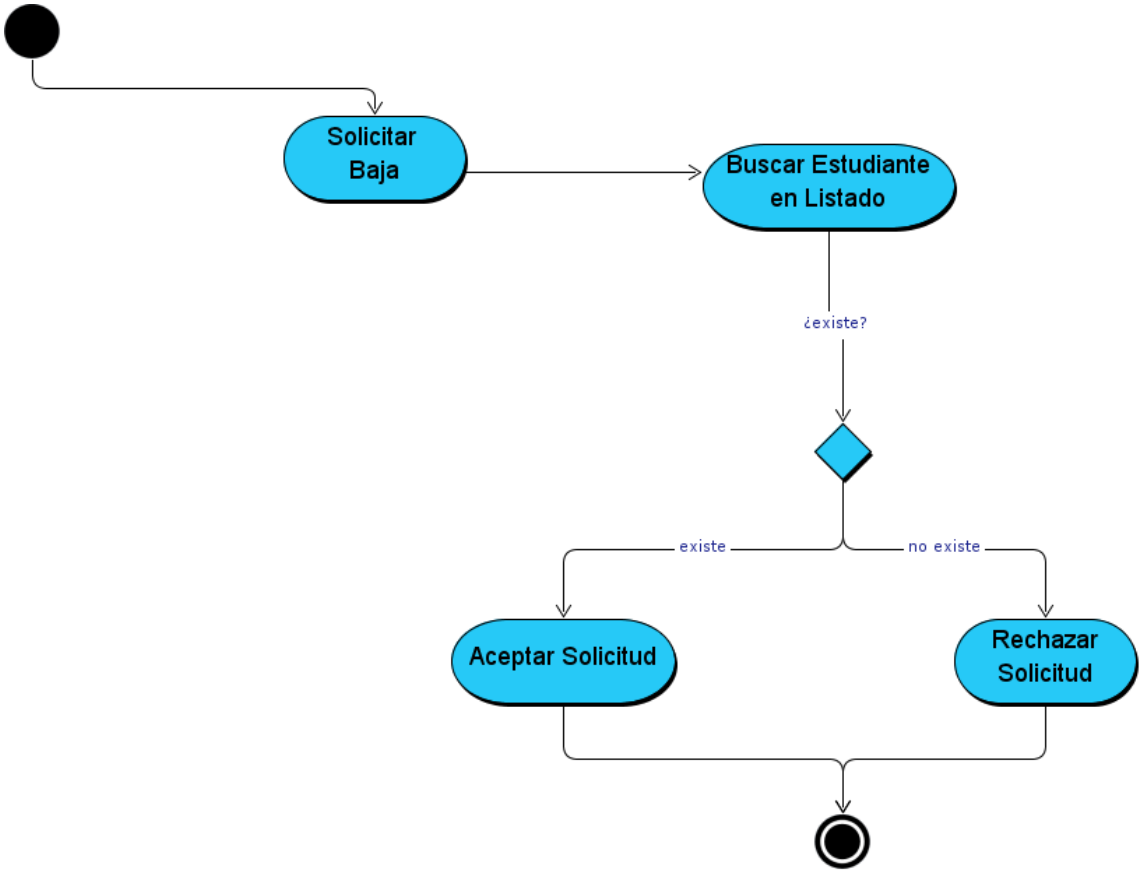


Figura 20: Diagrama de Estado PSM

Fuente: Elaboración propia

3.18 Conclusiones del Capítulo

1. Se brindó una descripción de uso del software para el mejor entendimiento de la aplicación y todas sus funcionalidades.
2. Se pusieron ejemplos de todos los diagramas y su modelado en el PIM.
3. Se pusieron ejemplos de todos los diagramas transformados por el software al modelo PSM.

CAPÍTULO 4

Capítulo 4: "Diseño de pruebas y análisis de factibilidad del módulo PIM-PSM 4.0 de la herramienta CASE jMDA."

En el presente capítulo se realiza la estimación del tiempo de desarrollo y las pruebas de caja negra, para comprobar que el sistema satisface los requisitos planteados. Se realiza un diseño detallado de los posibles escenarios de prueba y los principales resultados producto de su aplicación.

4.1 Estimación basada en casos de uso

El método de estimación de proyectos de software fue desarrollado en 1993 por Gustav Karner de *Rational Software* y está basado en una metodología orientada a objetos, dándole el nombre de "estimación de esfuerzos con casos de uso". Surge como una mejora al método de puntos de función, pero basando las estimaciones en el modelo de casos de uso, producto del análisis de requerimientos. Según su autor, la funcionalidad vista por el usuario (modelo de casos de uso) es la base para estimar el tamaño del software (Fernández 2012) .

4.2 Cálculo de puntos de casos de uso sin ajustar

Ecuación:	**Donde:**
UUCP=UAW+UUCW	**UUCP**: Puntos de Casos de uso sin ajustar
UUCP=3+25=28	**UAW**: Factor de peso de los actores sin ajustar
	UUCW: Factor de peso de los casos de uso sin ajustar

4.3 Factor de peso de los actores sin ajustar (UAW)

El factor de peso de los actores sin ajustar está dado por la complejidad de los actores con los que tendrá que interactuar el sistema. Este puntaje se calcula determinando si cada actor es una persona u otro sistema, a la forma en la que este interactúa con el caso de uso, y la cantidad de actores de cada tipo. Los criterios a tener en cuenta para su cálculo se describen en la Tabla 9.

Tabla 8: Factor de peso de Actores sin Ajustar. Fuente: Adaptado de (Fernández 2012)

Tipo de actor	Descripción	Factor de peso	Número de Actores	*Resultado*
Simple	Otro sistema que interactúa con el sistema a desarrollar mediante una interfaz de programación(*API*, *Aplication Programming Interface*)	1	0	*0*
Promedio	Otro sistema que interactúa con el sistema a desarrollar mediante un protocolo o una interfaz basada en texto.	2	0	*0*
Complejo	Una persona que interactúa con el sistema mediante una interfaz gráfica.	3	1	*3*
Total				*3*

Como se puede apreciar en la Tabla 8 en el sistema existe un solo actor, y como este interactúa con la interfaz de la herramienta, es de tipo complejo por lo que el resultado obtenido para el factor de peso de los actores sin ajustar es 3.

Entonces:

$$\textbf{UAW} = \sum (\text{Actor i} * \text{Factor de Pesoi})$$

$$\text{UAW} = 1 \times 0 + 2 \times 0 + 3 \times 1 = 3$$

4.4 Factor de peso en los casos de uso sin ajustar (UUCW)

Este punto funciona similar al anterior, pero para determinar el nivel de complejidad se puede realizar mediante dos métodos: basado en transacciones o basado en clases de análisis. En el caso del método basado en transacciones, la complejidad de los Casos de Uso se establece teniendo en cuenta la cantidad de transacciones efectuadas en el mismo, donde una transacción es una secuencia de actividades completa, donde se efectúan todas las actividades de la secuencia o no es efectuada ninguna de estas.

Tabla9: Factor de peso de los casos de uso… Fuente: Adaptado de (Fernández 2012)

Tipo de actor	Descripción	Factor de peso	Número de CU	*Resultado*
Simple	El Caso de Uso contiene de 1 a 3 transacciones	5	1	*5*
Promedio	El Caso de Uso contiene de 4 a 7 transacciones	10	2	*20*
Complejo	El Caso de Uso contiene más de 8 transacciones	15	0	*0*
Total				*25*

Como se muestra en la Tabla 9 el sistema tiene tres caso de uso principales, de ellos dos tienen de cuatro a siete iteraciones y el caso de uso nombrado “Realizar

transformación" tiene de una a tres transacciones. Por lo que se puede concluir que el factor de peso de casos de uso sin ajustar toma un valor de veinticinco puntos.

Entonces:

UUCW = ∑ (Caso de Uso i* Factor de Pesoi)

UUCW= 5 x 1+ 10 x 2 + 15 x 0= 25

4.5 Cálculo de puntos de casos de uso ajustados

Para el cálculo de los Casos de Uso ajustado se utilizan las siglas UCP y se obtiene al multiplicar el UUCP el TCF y el EF quedando de la siguiente forma:

Ecuación	Donde
UCP=UUCP*TCF*EF	**UCP:** Puntos de Casos de Uso ajustado
UCP=28*1.1*0.545 = 16.786	**UUCP:** Puntos de Casos de Uso sin ajustar
	TCF: Factor de complejidad técnica
	EF: Factor de Ambiente

4.6 Factor de complejidad técnica (TCF)

Este coeficiente se calcula mediante la cuantificación de un conjunto de 13 factores que determinan la complejidad de los módulos del sistema. Cada uno de los factores se cuantifica con un valor de 0 a 5, donde 0 significa un aporte irrelevante y 5 un aporte muy importante. En la Tabla 11 son descritos.

Tabla 10: Factores de complejidad técnica. Fuente: Adaptado de (Fernández 2012)

Número factor	Descripción	Peso	Valor	Factor
T1	Sistema Distribuido	2	1	2
T2	Tiempo de respuesta	1	5	5
T3	Eficiencia por el usuario	1	5	5
T4	Proceso interno complejo	1	4	4
T5	Reusabilidad	1	4	4
T6	Facilidad de instalación	0.5	1	0.5
T7	Facilidad de uso	0.5	5	2.5
T8	Portabilidad	2	5	10
T9	Facilidad de cambio	1	4	4
T10	Concurrencia	1	2	2
T11	Objetivos especiales de seguridad	1	2	2
T12	Acceso directo a terceras partes	1	2	2
T13	Facilidades especiales de entrenamiento a usuarios finales	1	3	3
Total Factor				46

En la Tabla 10 se ponderan los diferentes factores de complejidad técnica lo que da como resultado que el valor toral de este factor sea cincuenta.

El cálculo del factor de complejidad técnica se realiza mediante la siguiente ecuación:

TCF = 0.6 + 0.01 * Σ (Peso i x Valor asignado i)

TCF = 0.6 + 0.01* 46

TCF = 1.06

4.7 Factor de ambiente (EF)

Los factores sobre los cuales se realiza la evaluación son 8 puntos, los que están relacionados con los conocimientos y habilidades del grupo de persona que se encuentran en el proyecto, lo que produce un gran impacto en las estimaciones de tiempo. Estos factores se muestran en la Tabla 12:

Tabla11: Factores de ambiente. Fuente: Adaptado de (Fernández 2012)

Número del factor	Descripción	Peso	Valor	Factor
E1	Familiaridad con el modelo del proyecto usado.	1.5	4	6
E2	Experiencia en la aplicación	0.5	5	2.5
E3	Experiencia en orientación a objetos.	1	5	5
E4	Capacidad del analista líder.	0.5	4	2
E5	Motivación.	1	5	5
E6	Estabilidad de los requerimientos.	2	4	8
E7	Personal media jornada.	-1	0	0
E8	Dificultad en lenguaje de programación.	-1	0	0
Total				*28.5*

En la tabla 11 se realiza la ponderación de los factores de ambiente, como se puede apreciar dos de los más importantes son la estabilidad de los requerimientos y la motivación, piezas claves en el desarrollo de cualquier aplicación. Luego de ponderar cada uno el resultado total obtenido de 28.5 puntos.

El cálculo del factor de ambiente se realiza mediante la siguiente ecuación:

EF = 1.4 – 0.03 * Σ (Peso i x Valor asignado i)

EF = 1.4 – 0.03 * 28.5

EF = 0.545

4.8 Esfuerzo horas-hombre (E)

Este cálculo se realiza con el fin de tener una aproximación del esfuerzo, pensando solo en el desarrollo según las funcionalidades de los Casos de Uso. Para el cálculo del mismo se utiliza la siguiente ecuación:

Ecuación	Donde
E=UCP*CF	**E:** Esfuerzo estimado en horas-hombre

E=16.786*20	**UCP:** Puntos de Casos de Uso ajustados
E= 335.72	**CF:** Factor de conversión (20 horas-hombre por defecto)

4.9 Estimación del esfuerzo del proyecto

En la Tabla 13 se destaca la distribución en porcentaje del esfuerzo total de desarrollo del proyecto:

Tabla12: Distribución del esfuerzo. Fuente: Elaboración propia.

Actividad	*Porcentaje*
Análisis	*10.00%*
Diseño	*10.00%*
Programación	*40.00%*
Pruebas	*20.00%*
Sobrecarga(otras actividades)	*20.00%*

Con la distribución mostrada en la Tabla 12 y tomando como entrada la estimación de tiempo calculada a partir de los Puntos de Casos de Uso, se pueden calcular las demás estimaciones para obtener la duración total del proyecto mostrados en la Tabla 14.

Tabla13: Distribución del esfuerzo en el proyecto. Fuente: Elaboración propia.

Actividad	*Porcentaje*	*Horas / hombre*
Análisis	*10.00%*	83.93
Diseño	*10.00%*	83.93
Programación	*40.00%*	335.72
Pruebas	*20.00%*	167.86
Sobrecarga(otras actividades)	*20.00%*	167.86
Total		839.3

4.10 Cálculo del esfuerzo total

Ecuación:	Donde:
Etotal=∑ actividades	**ETotal**: esfuerzo total
Etotal 839.3horas/hombres	

4.11 Cálculo del tiempo de desarrollo

Ecuación:	Donde:
TDesarrollo=ETotal/CHTotal/CHTrabajo	**TDesarrollo**: tiempo de desarrollo total en horas
TDesarrollo=839.3/1/8	**CHTotal**: cantidad total de hombres
TDesarrollo= 105 días aproximadamente	**CHTrabajo**: cantidad de horas de trabajo diario

4.12 Cálculo del costo

Ecuación: **CostoTotal**=ETotal*CHTotal*TH CostoTotal=839.3*1*4.0625 CostoTotal= 3409.656	Donde: **TH**: El salario promedio de 1 desarrollador es de $650 y por tanto la **TH** = 650 / 160 = 4.0625

Ecuación: **CostoTotal**=ETotal*CHTotal*TH CostoTotal=671.44 *1*4.0625 CostoTotal= $2727.725	Donde: **TH**: El salario promedio de 1 desarrollador es de $650 y por tanto la **TH** = 650 / 160 = 4.0625

4.13 Casos de Prueba

Con el objetivo de que el software tenga la calidad requerida y de verificar si los requerimientos del usuario se han completado en la fase de implementación; son realizadas un grupo de pruebas para la corrección de los posibles errores que pueda presentar el mismo. En el presente epígrafe se enfoca en el diseño y ejecución de las pruebas de caja negra. Este tipo de pruebas como se explica en la fundamentación teórica permite la validación de las funcionalidades correspondientes con los casos de uso significativos del sistema.

Condiciones de ejecución:

En la tabla 14 se presenta se presentan un conjunto de precondiciones que deben cumplirse para la ejecución de los escenarios de prueba que se proponen.

Tabla14: Precondiciones para la ejecución de las pruebas de caja negra. Fuente: Elaboración propia.

Precondiciones para la ejecución de las pruebas	
Tipo Precondición	**Precondición**
Precondición invalidante ☒	La aplicación debe ejecutarse en una computadora que tenga instalado la versión 1.8.0 o superior de la máquina virtual de java.
Precondición no invalidante ☐	El probador debe tener conocimientos de la aplicación que se prueba.

Como se puede observar en la tabla anteriormente presentada la segunda precondición aunque no es un invalidante mejoraría la ejecución de las pruebas. Las funcionalidades que sean analizadas deben probarse de manera lógica y coherente lo que aumenta la calidad en los resultados que se obtengan en la ejecución las pruebas funcionales.

4.14 Diseño de las pruebas que serán aplicadas

El actual epígrafe se orienta a la creación de escenarios de prueba que son usados como punto de partida para la fase de su ejecución. Para la organización de estos escenarios se emplea un esquema que se propone en (Carrillo, 2006). Este formato tiene la potencialidad de identificar cada uno de los elementos que van a ser probados en los escenarios seleccionados.

Tabla15: Esquema de un caso de prueba "n". Fuente: Tomado de (Carrillo, 2006)

Esquema de un caso de prueba n	
Nombre del proceso	*Proceso XXXXX*
Descripción	*Acción del proceso*
Escenario	*Acción del usuario*
Entradas	*Datos de entrada del proceso*
Pre-condiciones	*Requerimientos para probar el proceso*
Procedimiento	*Pasos para probar el proceso*
Resultado	*Qué se obtiene al probar el proceso*
Observaciones	*Observaciones generales de la prueba*

4.15 Escenario 1: Creación de un nuevo diagrama en el modelo independiente de la plataforma.

Tabla 16: Escenario para la creación de un nuevo diagrama Caso 1. Fuente: Elaboración propia

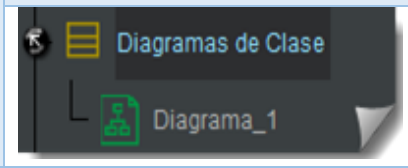
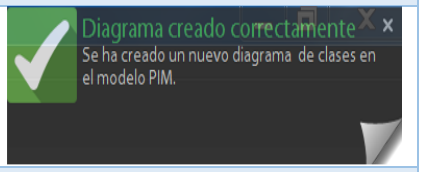
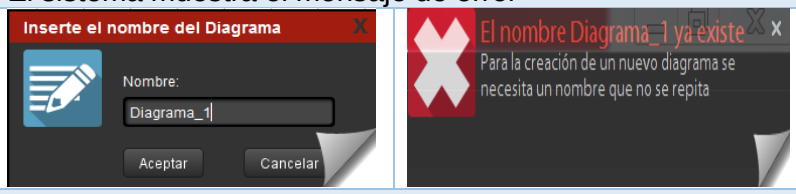
Escenario 1 Caso 1	
Nombre del proceso	Creación de un nuevo diagrama *PIM* Caso 1
Descripción	La creación de un nuevo diagrama provoca que se adicione un nuevo diagrama en el nodo seleccionado en el árbol del proyecto.
Escenario	El analista hace *click* derecho sobre un nodo del modelo *PIM*.
Entradas	Diagrama_1
Pre-condiciones	No existe ningún diagrama en el árbol del proyecto con el nombre “Diagrama_1”.
Procedimiento	1- El analista selecciona el nodo en el árbol del proyecto. 2- El analista hace *click* derecho sobre el nodo seleccionado. 3- El sistema muestra la ventana para añadir el nombre del nuevo diagrama. 4- El analista introduce el nombre. 5- El sistema muestra la notificación que indica que el diagrama se añadió correctamente
Resultado	La creación de un nuevo diagrama en el árbol del proyecto en el nodo seleccionado.  
Observaciones	No se encontró ningún problema en la prueba realizada.

Tabla 17: Escenario para la creación de un nuevo diagrama Caso 2. Fuente: Elaboración propia

Escenario 1 Caso 2	
Nombre del proceso	Creación de un nuevo diagrama *PIM* Caso 2
Descripción	La creación de un nuevo diagrama provoca que se adicione un nuevo diagrama en el nodo seleccionado en el árbol del proyecto.
Escenario	El analista hace *click* derecho sobre un nodo del modelo *PIM*.
Entradas	Diagrama_1

Pre-condiciones	Existe un diagrama en el árbol del proyecto con el nombre “Diagrama_1”.
Procedimiento	1- El analista selecciona el nodo en el árbol del proyecto. 2- El analista hace *click* derecho sobre el nodo seleccionado. 3- El sistema muestra la ventana para añadir el nombre del nuevo diagrama. 4- El analista introduce el nombre. 5- El sistema muestra la notificación que indica que el error provocado.
Resultado	El sistema muestra el mensaje de error 
Observaciones	No se encontró ningún problema en la prueba realizada.

4.16 Escenario 2: Realizar Transformación de un diagrama en el modelo independiente de la plataforma a el modelo específico de plataforma

Tabla 18: Escenario para la Transformación de un diagrama Caso 1. Fuente: Elaboración propia

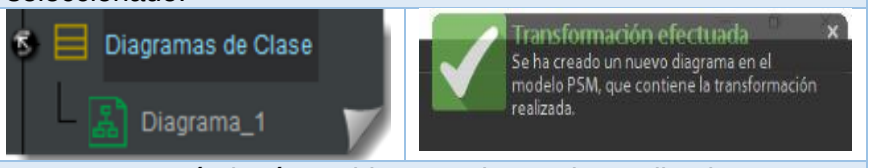
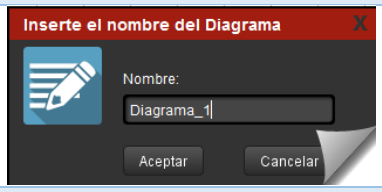
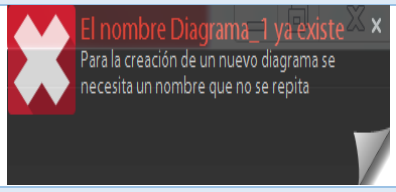
Escenario 2 Caso 1	
Nombre del proceso	Transformación de un diagrama *PIM a PSM* Caso 1
Descripción	La transformación de un diagrama provoca que se adicione un nuevo diagrama en el nodo correspondiente en el árbol del proyecto.
Escenario	El analista hace *click* derecho sobre un nodo del modelo *PIM*.
Entradas	Diagrama_1
Pre-condiciones	No existe ningún diagrama en el árbol del proyecto con el nombre “Diagrama_1”.
Procedimiento	1- El analista selecciona el nodo en el árbol del proyecto. 2- El analista hace *click* derecho sobre el nodo seleccionado. 3- El sistema muestra la ventana para añadir el nombre del nuevo diagrama. 4- El analista introduce el nombre. 5- El sistema muestra la notificación que indica que el diagrama se añadió correctamente
Resultado	La transformación de diagrama en el árbol del proyecto en el nodo seleccionado. 
Observaciones	No se encontró ningún problema en la prueba realizada.

Tabla 19: Escenario para la creación de un nuevo diagrama Caso 2. Fuente: Elaboración propia

Escenario 2 Caso 2	
Nombre del proceso	Transformación de un diagrama *PIM a PSM* Caso 1
Descripción	La transformación de un diagrama provoca que se adicione un nuevo diagrama en el nodo correspondiente en el árbol del proyecto.
Escenario	El analista hace *click* derecho sobre un nodo del modelo *PIM*.
Entradas	Diagrama_1
Pre-condiciones	Existe un diagrama en el árbol del proyecto con el nombre “Diagrama_1”.
Procedimiento	1- El analista selecciona el nodo en el árbol del proyecto. 2- El analista hace *click* derecho sobre el nodo seleccionado. 3- El sistema muestra la ventana para añadir el nombre del nuevo diagrama. 4- El analista introduce el nombre. 5- El sistema muestra la notificación que indica que el error provocado.
Resultado	El sistema muestra el mensaje de error  
Observaciones	No se encontró ningún problema en la prueba realizada.

4.17 Escenario 3: Especificación de una clase en el modelo PIM

Tabla 20: Escenario para la modificación de los datos de una clase Caso 1.

Fuente: Elaboración propia.

Escenario 3 Caso 1	
Nombre del proceso	Modificación de la especificación de una clase del modelo *PIM*.
Descripción	El analista hace *click* secundario sobre una forma, y a través del menú que le muestra el sistema selecciona la opción “Especificación”.
Escenario	Escenario para la modificación de los datos de una clase.
Entradas	Nombre : atributo_1 Tipo: Cadena Visibilidad: público Valor: -
Pre-condiciones	Existe un área de trabajo con una forma de tipo clase añadida.
Procedimiento	1- El analista hace *click* secundario sobre la forma. 2- El analista selecciona la opción “Abrir especificación” a través del menú “Especificación”. 3- El analista en la paleta correspondiente a los atributos presiona el botón añadir.

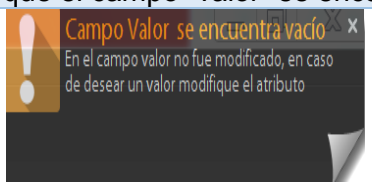
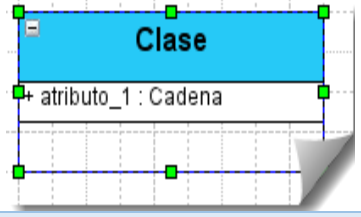
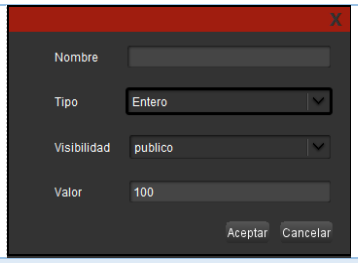
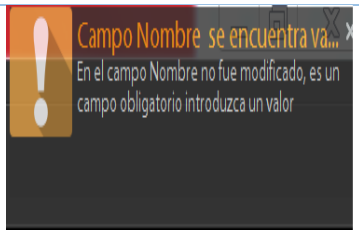
	4- El sistema muestra una ventana para la introducción de los datos. 5- El analista introduce los datos. 6- El sistema muestra una notificación indicando que el campo “Valor” se encuentra vacío. 7- El analista presiona el botón aplicar. 8- El analista presiona el botón aceptar. 9- El sistema actualiza la información.
Resultado	El sistema actualiza la información de la clase correctamente y muestra la notificación al usuario indicando que el campo “valor” se encuentra vacío.  
Observaciones	No se encontró ningún problema en la prueba realizada.

Tabla 21: Escenario para la modificación de los datos de una clase caso 2.

Fuente: Elaboración propia

Escenario 3 Caso 2	
Nombre del proceso	Modificación de la especificación de una clase del modelo *PIM*.
Descripción	El analista hace *click* secundario sobre una forma, y a través del menú que le muestra el sistema selecciona la opción “Especificación”.
Escenario	Escenario para la modificación de los datos de una clase.
Entradas	Nombre : - Tipo: Entero Visibilidad: público Valor: 100
Pre-condiciones	Existe un área de trabajo con una forma de tipo clase añadida.
Procedimiento	1- El analista hace *click* secundario sobre la forma. 2- El analista selecciona la opción “Abrir especificación” a través del menú “Especificación”. 3- El analista en la paleta correspondiente a los atributos presiona el botón añadir. 4- El sistema muestra una ventana para la introducción de los datos. 5- El analista introduce los datos. 6- El sistema muestra una notificación indicando que el campo “Nombre” se encuentra vacío y que es obligatorio. 7- El sistema espera que el analista introduzca un valor y presione el botón aceptar. 8- El analista presiona el botón aplicar. 9- El analista presiona el botón aceptar. 10- El sistema actualiza la información.
Resultado	El sistema informa al usuario de que tiene un campo obligatorio sin llenar.

		
Observaciones	No se encontró ningún problema en la prueba realizada.	

4.18 Conclusiones del capítulo

1. Se realizó un estudio de factibilidad donde se estimaron los principales costos asociados al desarrollo de la versión 5.0 de la herramienta jMDA.
2. Se diseñaron tres escenarios de prueba para la validación funcional de la herramienta.
3. Las pruebas realizadas al *software* demostraron en un 100 % el correcto funcionamiento de las funcionalidades implementadas en la aplicación.

CONCLUSIONES

1. Se analizaron los referentes teórico-metodológicos concluyendo que el paradigma *MDA* permite incrementar la productividad, interoperabilidad y calidad del software que se desarrolla; que no se ha identificado que un estándar XMI que permita la estandarización de los diagramas modelados para su intercambio.
2. Se definieron los requisitos funcionales y no funcionales del sistema, determinando la correspondencia entre los requisitos funcionales y los casos de uso del sistema, fueron diseñadas las reglas de transformación del modelo PIM al modelo PSM y se modeló el algoritmo de transformación utilizado en la notación BPMN.
3. Se realizó la ingeniería inversa de la versión 4.0 que permitió notar donde se encontraban los errores más destacables y se desarrolló la versión 5, lo que incluye varios diagramas de UML en análisis y diseño y su implementación en Java utilizando la librería mxgraph con el NetBeans IDE 8.0.1 y el JDK 8.
4. Se realizó un estudio de factibilidad donde se estimaron los principales costos asociados al proyecto.
5. Se desarrolló un ejemplo de uso del módulo implementado que incluye los diagramas correspondientes en el modelo PIM y los obtenidos en el modelo PSM a través de la transformación establecida.

RECOMENDACIONES

1. Impulsar el uso del módulo PIM-PSM versión 5.0 de la herramienta jMDA en la docencia de pregrado y postgrado de la UCLV.
2. Continuar el perfeccionamiento del módulo obtenido en próximas versiones que permitan:
    a. Implementar otras plataformas como C# y Phyton.
    b. Lograr establecer la trazabilidad entre los diagramas en los diferentes modelos.
    c. A partir de lo anterior tratar de lograr la reconstrucción.
3. Publicar los resultados obtenidos en eventos.

REFERENCIAS BIBLIOGRÁFICAS

Fernández, L. O. (2012). Estimación de software basada en puntos de casos de uso. Ejemplo práctico - Monografias.com.

PADILLA HERNÁNDEZ, J. M. (2010). Universidad politécnica de valencia. *Tesis Doctoral*. doi:10.4995/Thesis/10251/11227

Quintero, J. B., & Anaya, R. (2007). MDA y el papel de los modelos en el proceso de desarrollo de software. *Revista EIA*, (8), 131–146.

Quintero, R., Pelechano, V., Fons, J., & Pastor, O. (n.d.). Aplicación de MDA al Desarrollo de Aplicaciones Web en OOWS, 1–12.

Albeiros, C. (2009). Comparativo de herramientas MDA.

Bernardo, J., & Duitama, J. F. (2011). Reflexiones acerca de la adopción de enfoques centrados en modelos en el desarrollo de software.

Bernardo Quintero, J., & Anaya de Páez, R. (2007). Marco de Referencia para la Evaluación de Herramientas Basadas en MDA.

Bollati, V., & Vara, J. (2012). Análisis de herramientas MDA.

Bonillo, P. (2014). [Propuesta de un MArco de Arquitectura Empresarial para la Gestión de Tecnología y Sistemas de Información.].

Booch, G., Brown A., Rumbaugh, J. (2004). An MDA Manifesto. . Retrieved from

Calisoft. (2014). Libro de diagnóstico.

Carrillo, C. E. (2006). Arquitectura dirigida por modelos (MDA) y su aplicación en un caso de estudio.

Consuelo, M. (2010). MDA: Arquitectura dirigida por modelos.

Córdova, A. (2011). Sistema automatizada de búsqueda web de promociones de tickets aéreos y portal web para la agencia de viajes y turismo valle cía.

Franky, M. C. (2010). *MDA: Arquitectura Dirigida por Modelos*. Universidad Javeriana.

Gaitán Torres, L. C. (2012). Refactorización de Marcos Orientados a Objetos hacia Arquitecturas MVC

García, M. (2004). Un estudio comparativo de dos herramientas MDA: OptimalJ y ArcStyler.

Gulzar, N. (2002). Fast Track to Struts: What id Does and How. .

Lazo Alvarado, Y., Gómez Barroso, C., Mariño Zayas, Y., Bony Fernández, M. M., Agramonte Díaz, E., & Batista González, D. (2016). Proceso de aseguramiento de la calidad para un modelo de la calidad en Cuba.

Medicine, I. o. (2004). Patient Safety: Achieving a New Standard for Care. National Academy Press.

Mora, B. (2006). Definición de lenguajes de modelados MDA vs DSL.

Moreno, S. (2009). Análisis del Grafical Modeling Framework del Proyecto Eclipse Universidad Complutense de Madrid.

OMG, M. (2003). Guide Version 1.0. 1. *Object Management Group, 62*, 34.

(2009).