Universidad Central "Marta Abreu" de Las Villas

Facultad de Matemática, Física y Computación

Ingeniería Informática

TRABAJO DE DIPLOMA

Título: "Desarrollo del módulo *PIM-PSM* versión 4.0 de la herramienta *jMDA*"

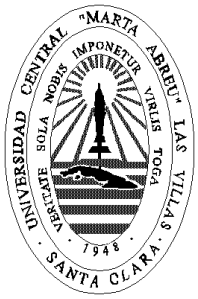
Autor: Liosbel Díaz Lorenzo

Tutor: Dr. Rosendo Moreno Rodríguez.

Ing. Patricia Airela Abreu Fong.

Santa Clara

2017



El que suscribe Liosbel Díaz Lorenzo , hago constar que el trabajo titulado "Desarrollo del módulo PIM-PSM versión 4.0 de la herramienta jMDA" fue realizado en la Universidad Central "Marta Abreu" de Las Villas como parte de la culminación de los estudios de la especialidad de Ingeniería Informática, autorizando a que el mismo sea utilizado por la institución, para los fines que estime conveniente, tanto de forma parcial como total y que además no podrá ser presentado en eventos ni publicado sin la autorización de la Universidad.

_____

Firma del Autor

Los abajo firmantes, certificamos que el presente trabajo ha sido realizado según acuerdos de la dirección de nuestro centro y el mismo cumple con los requisitos que debe tener un trabajo de esta envergadura referido a la temática señalada.

_____	_____
Firma del Tutor: Dr. Rosendo Moreno Rodríguez	Firma del Jefe del Laboratorio Dr. Carlos Pérez Risquet

DEDICATORIA

A mi madre Misleidys

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

A mi padre Leonel

A mi padre quien me enseñó a valorar los resultados de un gran esfuerzo, a conocer el precio de tener una gota de sudor en la frente, por ser "amigo" y ejemplo.

A mi novia Patricia

Por estar siempre a mi lado en todo momento, por ser un ejemplo de superación, y sobre todas las cosas por su amor y paciencia durante estos 5 años.

AGRADECIMIENTOS

Quiero agradecer a aquellas personas que compartieron sus conocimientos conmigo para hacer posible la conclusión de esta tesis. Especialmente agradezco a mi tutor el Dr. Rosendo Moreno Rodríguez y mi tutora la Ing. Patricia Airela Abreu Fong por su asesoría y apoyo incondicional.

A mi hermanita Isis Beatriz, que siempre estuvo pendiente de la tesis, fue mi probadora oficial y que me dio fuerzas con su alegría en los momentos más complicados de la tesis.

A mi suegrita, Laura, por ser todo lo contrario a lo que esa expresión con que la nominé representa en el sentido común. Gracias por su cariño, comprensión, apoyo y paciencia. Por quererme y cuidarme como su hijo.

A todos mis amigos especialmente a Denismay, Alejandro Marínez, Alejandro Alemán, Cesar Luis, Luis Yoel, Javier Antonio, Lázaro Zunda , José Alejandro , Asiel Benitez por sus críticas constructivas durante el proceso de implementación de la herramienta y por estar siempre en los buenos y malos momentos, a todos en general muchas gracias.

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)*, 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)*, 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 4.0 del módulo PIM-PSM de la herramienta "jMDA", constituye un perfeccionamiento de las anteriores, centrado en la implementación de una interfaz gráfica superior que posibilite la edición iterativa de los diagramas *UML* de clases, casos de uso, actividades, estado y secuencia al nivel de "*Platform Independent Model"* (*PIM*); 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)*. 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.

Palabras Claves:

Ingeniería del Software, Arquitectura Dirigida por Modelos.

Abstract

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

The Object Management Group (OMG) is a computer consortium aimed at enhancing the development of object-oriented applications. Since its inception, it has given special attention to the problem of interoperability and software integration, defining numerous specifications and standards. In 2001, OMG established the framework "Model Driven Architect" (MDA), as an architecture for the development of applications. In this paradigm of software development, called Model Engineering or Model Based Development, the models guide the whole process.

Version 4.0 of the PIM-PSM module of the "jMDA" tool is an improvement on the previous ones, focused on the implementation of a superior graphical interface that enables the iterative editing of the UML diagrams of classes, use cases, activities, state and sequence at the level of Independent Platform Model (PIM); as well as the computer-aided transformation of such diagrams and the creation of Artifact and Deployment diagrams at the Platform Specific Model (PSM) level. The development of the research not only focuses on the implementation of the solution, but also on the generation of the 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.

Key Works:

Software Engineering, Model Driven Architecture.

Índice

Índice de Figuras

Índice de Tablas

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)

Es por eso que el *OMG* 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 Estudios 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.

Dada la complejidad de esta tarea posteriormente se decide pasar al desarrollo de módulos independientes, cada uno de los cuales haría énfasis en la edición de un modelo específico y su transformación al siguiente; como consecuencia de esto el módulo PIM-PSM ha sido el más tratado y del mismo se desarrolla la versión 2.0 en la que se mejora la edición de los diagramas que existían anteriormente, se incorporaron

los diagramas de estado y de secuencia en el modelo *PIM* y se añade la capacidad de transformar los diagramas de clases al modelo *PSM.*

Más adelante en la versión 3.0 del módulo PIM-PSM se incorporaron mejoras en la interfaz de gráfica de modo que fuera más fácil la utilización, también se le incorpora la opción de exportar los diagramas de clases a una versión del estándar de *"XML Metadata Interchange" (XMI por sus siglas en inglés)* y se insertan nuevas funcionalidades que posibilitan la creación de nuevos elementos gráficos a los diagramas existentes en las anteriores versiones.

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

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

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 3.0 en el módulo *PIM-PSM* presenta insuficiencias en la representación de diagramas *UML*, en la interacción del usuario con la misma desde el punto de vista gráfico y conceptual, así como problemas en la transformación asistida del *PIM* al *PSM* con uso de *Java*.

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

Implementar una nueva versión del módulo *PIM-PSM* de la herramienta *CASE jMDA* que soporte una mejor representación gráfica de los diagramas *UML*, 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.*

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 4.0 de jMDA PIM-PSM, así como el estado actual de esta herramienta.
2. Diseñar la versión 4.0 de de jMDA PIM-PSM fundamentando las reglas de transformación a implementar.
3. Implementar la versión 4.0 de jMDA PIM-PSM, lo que incluye la creación de un ambiente gráfico adecuado para la edición iterativa de diferentes diagramas *UML* tanto en ambiente *PIM* como *PSM*, así como la lógica de transformaciones propuesta para pasar de un modelo al otro.

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?

La investigación que se desarrolla se sustenta en la siguiente hipótesis:

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.

Viabilidad:

El desarrollo del presente trabajo es viable por varios aspectos. Primero existe una mayor comprensión de la teoría de la Arquitectura Dirigida por Modelos que al inicio de la investigación. Se cuenta con las experiencias positivas y negativas de versiones anteriores, que ayudan a la obtención de un producto más funcional. Existen nuevas herramientas gráficas en código abierto que son posibles de aplicar para la obtención de un resultado totalmente operativo. Se cuenta con el equipamiento y los *softwares* básicos necesarios, así como el conocimiento alcanzado durante los estudios de Ingeniería Informática por parte del autor. Todo esto hace posible el cumplimiento de los objetivos trazados.

Alcance:

El presente trabajo está circunscrito al desarrollo del módulo PIM-PSM de la herramienta jMDA que comenzó su implementación como parte de un Proyecto Nacional para la obtención de un CASE de factura nacional que apoye la creación de aplicaciones del tipo Sistemas de Información, con uso intensivo de datos presumiblemente a partir de Bases de Datos con múltiples tablas sobre diferentes Sistemas Gestores de Base de Datos (SGBD). También estas aplicaciones se caracterizan por una Interfaz Gráfica de Usuarios con uso de ventanas de captación, modificación, y otras operaciones sobre esos datos. Por ello el módulo PIM-PSM que se propone permite el diseño de varios diagramas *UML* en el nivel PIM, su transformación a PSM y la edición en este nivel de esos y otros diagramas que aparecen en este último nivel. Todo ello enfocado a la gestación de Sistemas de Información.

Estructura de la Tesis:

Para una mejor comprensión del presente documento, la estructura del contenido queda contemplada de la siguiente forma:

Capítulo 1. "Análisis del marco teórico conceptual sobre el paradigma MDA": Comprende los elementos conceptuales que sustentan el problema científico y los objetivos. Se hace un estudio donde se tienen en cuenta definiciones, conceptos, estándares relacionados con el paradigma *MDA*, documentando además los conceptos fundamentales de UML. Se realiza un análisis de herramientas que se adhieren al paradigma *MDA.* Es efectuado una evaluación crítica de las versiones anteriores utilizando los criterios definidos por (García, 2004). Se fundamenta la selección de tecnologías, herramientas y metodología con las cuales se desarrolla el sistema.

Capítulo 2. "Análisis y diseño del Módulo PIM-PSM 4.0 de la herramienta CASE jMDA": Comprende el análisis y diseño de la herramienta CASE jMDA PIM-PSM v4.0 que se propone como solución de este trabajo. Se describen además los casos de uso del sistema significativos, así como los requisitos funcionales y no funcionales de la aplicación.

Capítulo 3. "Descripción de la propuesta de solución para la implementación del módulo PIM-PSM 4.0 de la herramienta CASE jMDA": Comprende la descripción del proceso de implementación, abarcando desde los diagramas de la arquitectura del sistema hasta los diagramas de secuencia de los casos de uso significativos. Contempla además el tratamiento de errores utilizado en el sistema, el diagrama de componentes.

Capítulo 4. "Diseño de pruebas y análisis de factibilidad del módulo PIM-PSM 4.0 de la herramienta CASE jMDA": Comprende el análisis de factibilidad aplicando el método de "estimación basada en casos", luego de realizar los cálculos pertinentes se ejecuta un análisis de los resultados obtenidos. Se diseñan los casos de prueba y posteriormente se somete el sistema a estos.

CAPÍTULO 1

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

La Arquitectura Dirigida por Modelos, *"Model Dirven Architecture"* o simplemente (*MDA)* por sus siglas en inglés, es una especificación promovida por el *OMG* en el año 2001 integrando diferentes estándares y especificaciones definidos por la misma organización

con el fin de proveer una solución a los problemas relacionados con los cambios en los modelos de negocio, la tecnología y la adaptación de los sistemas de información a los mismos (Vara, de Castro, Cáceres, & Marcos, 2017). 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.
Suele escucharse decir 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*.

Para lograr un mejor entendimiento se detallan algunos conceptos primarios.

- **Arquitectura:** La arquitectura de un sistema es la especificación de las partes del mismo, las interacciones entre ellas y sus normas de interacción haciendo uso de las conexiones especificadas (Albeiros, 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 una especificación de ese sistema y su entorno para desempeñar un determinado objetivo. 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:** Consiste en el análisis, construcción y desarrollo de esquemas, reglas, restricciones, modelos y teorías que se utilizan para el modelado de clases predefinidas de problemas. (Albeiros, 2009)
- **Dirigido por 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)

1.1.1 Aspectos generales y fundamentos de *MDA:*

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:

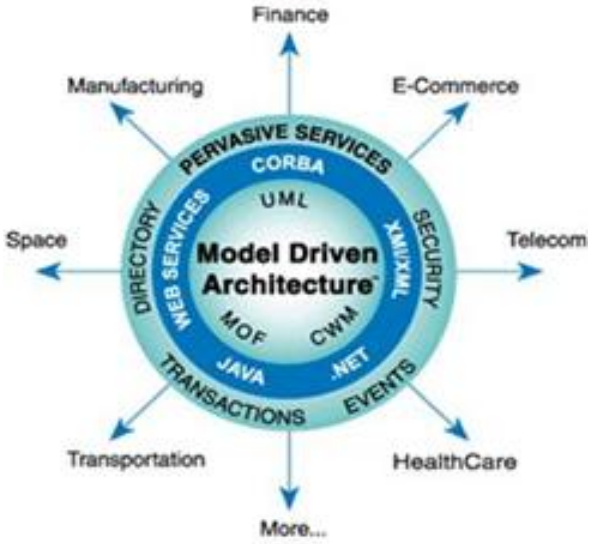


Figura 1: Arquitectura dirigidas por modelos de OMG.

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 modelación:** está representada por el lenguaje de modelación *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

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

el entorno de un almacén de datos, tales como, objetos relacionales, no relacionales, entre otros. (Quintero, 2003),(OMG, 2003)

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.	**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 los pros y los contras 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 (Carrillo, 2006) 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 tienen que presentar los modelos. Lo que significa que un modelo es independiente de las facilidades o características que implementan las plataformas, de cualquier tipo. (Carrillo, 2006)

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 (Mora, 2006):

- **Transformaciones de tipos *Model Type Mapping:*** según la especificación, "una transformación de tipos especifica una transformación para 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

Lenguaje Unificado de Modelado ("*Unified Modeling Language*"; *UML por sus siglas en inglés)* es un lenguaje gráfico para el desarrollo de sistemas que permite modelar la arquitectura, los objetos, las interacciones entre objetos, datos y aspectos del ciclo de vida de una aplicación, así como otros aspectos más relacionados con el diseño de componentes incluyendo su construcción y despliegue.(Mora, 2006; UML-OMG, 2012)

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 colaboración. 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 cada uno de ellos:

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 Objetos**: Muestra un conjunto de objetos y sus relaciones. Representan instantáneas de instancias de los elementos encontrados en los diagramas de clases. Cubren la vista de diseño estática o la vista de procesos

estática de un sistema como lo hacen los diagramas de clases, pero desde la perspectiva de casos reales o prototípicos.

- **Diagrama de Artefactos**: Muestra los constituyentes físicos de un sistema en la computadora. Incluyen archivos, bases de datos y colecciones físicas similares a los bits. Se usan generalmente en conjunción con diagramas de despliegue. Presentan además las clases y componentes que ellos implementan.
- **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 Estructuras Compuestas**: Se emplea para visualizar de manera gráfica las partes que definen la estructura interna de un clasificador. En el marco de una clase, este diagrama permite ampliar un diagrama de clases donde se muestran los diferentes atributos (partes), y las clases, a partir de las cuales se definen los atributos, indicando principalmente las asociaciones de agregación o de composición de la clase a la que se le elabora el diagrama.
- **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.
- **Diagramas de Interacción:** Son un subgrupo dentro de los diagramas de comportamiento especializados en el flujo de control y en el flujo de datos entre los elementos del sistema modelado. Este subgrupo incluye a:
  - **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.
  - **Diagrama de Comunicación** (antes Diagrama de colaboración en UML 1.0) Es un diagrama de interacción que resalta la organización estructural de los objetos que envían y reciben mensajes.
  - **Diagrama de Tiempos:** Se emplea para mostrar las interacciones donde el propósito fundamental consiste en razonar sobre la ocurrencia de eventos en el tiempo que provocan el cambio de estados de un elemento estructural (clase, componente, etc.). Existen 2 tipos: Notación Concisa y Notación Robusta.
  - **Diagrama General de Interacciones**: Se emplea fundamentalmente para representar las interacciones, a través de diagramas o fragmentos de diagramas de secuencias, entre los actores y el sistema como una gran caja negra, y de diagramas de actividades en los que aparecen dichos fragmentos. O sea, es un híbrido entre diagramas de secuencia y de actividades.

1.3.3 Perfiles UML

Un perfil UML se define como un conjunto de estereotipos, restricciones y valores etiquetados. Estos conceptos forman parte del mecanismo de extensión de UML. Se explican a continuación cada uno de ellos:(García, 2004)

- Mediante estereotipos se puede crear nuevos tipos de elementos de modelado basado en elementos ya existentes en el metamodelo de *UML*. Por lo tanto, un estereotipo será un nuevo tipo de elemento de modelado que extiende la semántica del modelado. Los estereotipos están definidos por un nombre y por una serie de elementos del metamodelado sobre los que puede asociarse. Gráficamente un estereotipo se representa con marcas como: *«nombre-estereotipo»*. Opcionalmente el elemento estereotipado se puede dibujar con un nuevo icono asociado al estereotipo.
- Las restricciones imponen condiciones que deben cumplir determinados elementos del modelo para que este esté "bien formado", según un dominio de aplicación específico. Gráficamente, una restricción se representa como una cadena de caracteres conectada a él por una relación de dependencia. Una restricción generalmente se define mediante una expresión en OCL.
- Un valor etiquetado es una extensión de las propiedades de un elemento de UML, permitiendo añadir nueva información en la especificación del elemento. Gráficamente un valor etiquetado se representa como una cadena de caracteres entre llaves asociada al nombre del elemento. La cadena incluye un nombre (etiqueta), un separador (=), y un valor (el de la etiqueta).

En la Figura 2 se puede ver un ejemplo sencillo de la representación gráfica de estos tres elementos en UML.

Figura 2: Ejemplo de estereotipo, restricción y valor etiquetado en UML.

Fuente: Tomado de (García, 2004)

Para obtener un perfil se tiene que especializar un subconjunto de *UML* a través de estereotipos, restricciones y valores etiquetados. Como resultado de crear un perfil *UML* se obtiene una variante de *UML* para un propósito específico, que se puede usar para completar un modelo con detalles específicos de una tecnología o plataforma determinada, o lo que es lo mismo, puede ser usado como lenguaje para definir *PSMs*.

Los perfiles *UML* permiten:

- Disponer de una terminología y vocabulario propio de un dominio de aplicación o de una plataforma de implementación correcta.
- Definir una nueva notación para símbolos ya existentes, más acorde con el dominio de aplicación.
- Añadir cierta semántica que no existe o no aparece determinada de forma precisa en el metamodelo.
- Añadir restricciones a las existentes en el metamodelo, restringiendo su forma de utilización.
- Añadir información que puede ser útil a la hora de transformar el modelo a otros modelos o a código.

Actualmente existen perfiles para COBRA, *Java*, EJB o C++, lo que permite construir *PSMs* específicos para estas tecnologías.

1.4 Meta Object Facility

El *OMG* ha creado el estándar *"Meta Object Facility"* (*MOF)* por sus siglas en inglés, que extiende *UML* para que este sea aplicado en el modelado de diferentes sistemas de información. El estándar *MOF* define diversos metamodelos, esencialmente abstrayendo la forma y la estructura que describe los metamodelos. *MOF* es un ejemplo de un meta metamodelo o un modelo del metamodelo. Define los elementos esenciales: sintaxis y estructuras de metamodelos que se utilizan para construir modelos de sistemas. Además, proporciona un modelo común para los metamodelos de *CWM* y *UML*.(Mora, 2006).

El gran potencial de *MOF* reside en que permite interoperar entre metamodelos muy diferentes que representan dominios diversos. Las aplicaciones que utilizan *MOF* no tienen por qué conocer las interfaces del dominio específico de algunos de sus modelos, pero pueden leer y actualizarlos utilizando las operaciones genéricas y reflexivas ofrecidas en las interfaces. La semántica de *MOF* define generalmente servicios para el repositorio de metadatos, para así permitir la construcción, la localización, la actualización. En concreto una implementación *MOF* debe proporcionar herramientas

para la autorización y la publicación de metadatos contra un repositorio *MOF*.(Mora, 2006)

1.4.1 Estándar de intercambio de metadatos para *MOF*

XML Metada Interchange (XMI) es un estándar que permite expresar en forma de fichero *XML* cualquier modelo (o meta-modelo) definido en *MOF*. Esta facilidad para la serialización o disposición en forma de flujo de datos (o meta-datos) ofrece un formato adecuado para intercambiar entre diferentes herramientas aquella información cuya semántica ha sido expresado en *MOF* .(Mora, 2006)

La especificación *XMI* incluye básicamente los siguientes elementos según (Mora, 2006):

- Un conjunto de reglas de producción sobre definición de tipo de documentos (*DTD*) *XML*, para transformar metamodelos basados en *MOF* en *XML DTD* (*Document Type Definitions*).
- Un conjunto de reglas de producción sobre documentos *XML*, para la codificación y decodificación de metadatos basados en *MOF*, es decir, para representar los metamodelos en *XMI*.
- Principios de diseño para los *DTD's* y flujos *XML* basados en *XMI*.
- *DTD* específicos del *UML* y del *MOF*.

Los metamodelos *MOF* se convierten en *DTD* y los modelos se convierten en documentos *XML* que son consistentes con su *DTD* correspondiente. *XMI* resuelve muchos de los problemas encontrados cuando se intenta utilizar un lenguaje basado en etiquetas para representar objetos y sus asociaciones, y además el hecho de que *XMI* esté basado en *XML* significa que tanto los metadatos (etiquetas) y las instancias que describen (elementos) se pueden agrupar en el mismo documento, permitiendo a las aplicaciones entender rápidamente las instancias por medio de los metadatos.

Muchas de las herramientas *CASE* [6] soportan *XMI* y el estándar para importar y exportar el formato. *XMI* no sólo se puede utilizar como un formato de intercambio *UML*, sino que puede ser utilizado para cualquier formato descrito por medio de un metamodelo *MOF*. Las herramientas compatibles con MOF permiten definir metamodelos o importarlos, por ejemplo, de repositorios y herramientas *CASE*, y empezar a editar o modificar la información del modelo, por ejemplo, instancias concretas de los servicios de negocios.

[6] Herramientas CASE: Conjunto de programas y ayudas que dan asistencia a los analistas, ingenieros de software y desarrolladores, durante todos los pasos del ciclo de vida de desarrollo de un software.

Una vez hecho esto la información del modelo podría ser intercambiada con otra herramienta compatible con *MOF* por medio de *XMI*.

UML, *MOF* y *XMI* son tres tecnologías clave para el desarrollo de software bajo el enfoque de *MDA*. Utilizadas de forma conjunta proporcionan grandes ventajas que hacen que los modelados sean más claros y fácilmente mantenibles. Estas tecnologías definen una forma estándar de almacenar e intercambiar modelos, bien sean de negocio o de diseño. Las herramientas que implementan estos estándares permiten automatizar y estandarizar numerosos procesos del desarrollo que facilitan muchas tareas, que antes eran manuales o que se realizaban de forma automática por medio de alguna característica propietaria de la herramienta, que en muchos casos hacía imposible el intercambio con otras herramientas del mercado.

1.4.2 Estándar de transformación de modelos para *MOF*

"Query, Views and Transformation" (QVT) por sus siglas en inglés, es un estándar que define el modo en que se llevan a cabo las transformaciones entre modelos cuyos lenguajes han sido definidos usando *MOF*. Consta de tres partes:

- Un lenguaje para crear vistas de un modelo.
- Un lenguaje para realizar consultas sobre modelos.
- Un lenguaje para escribir definiciones de transformaciones.

La última parte es la más relevante para *MDA*, pues actualmente no existe un modo estándar de definir transformaciones entre modelos. (Bernardo Quintero & Anaya de Páez, 2007)

La propuesta distingue entre dos tipos de transformaciones:

- **Relaciones:** Especificaciones de transformaciones multidireccionales. No son ejecutables en el sentido de que son incapaces de crear o modificar un modelo. Permiten comprobar la consistencia entre dos o más modelos relacionados. Se utilizan normalmente en la especificación del desarrollo de un sistema o para comprobar la validez de un *mapping*.
- ***Mappings*:** Implementaciones de transformaciones. A diferencia de las relaciones, los *mapping* son unidireccionales y pueden devolver valores. Un *mapping* puede refinar una o varias relaciones, en cuyo caso el *mapping* debe ser consistente con las relaciones que refina.

La propuesta también incluye un lenguaje estándar para definir relaciones y *mapping*, llamado *"Model Transformation Language" (MLT). MLT* utiliza el *pattern matching* como

uno de sus factores clave para permitir definir transformaciones potentes. La idea esencial del *pattern matching* es permitir expresar brevemente restricciones complejas sobre un tipo de dato de entrada, los datos que cumplen el patrón se seleccionan y se retornan al invocador.

1.5 Lenguaje para la definición de transformaciones

Object Constraint Language (OCL) por sus siglas en inglés, es un lenguaje de especificación con el que podemos escribir expresiones sobre modelos. Por ejemplo, el cuerpo de una operación de consulta, invariantes, precondiciones y postcondiciones. De esta manera, se puede definir modelos más precisos y completos. Algunos de los usos de *OCL* son:

- Especificar valores iniciales de atributos.
- Definir el cuerpo de operaciones de consulta.
- Establecer condiciones de guardia *Statecharts*.
- Especificar las reglas de derivación para atributos o asociaciones.
- Expresar restricciones sobre clases o atributos.

Además de dar más precisión a los modelos, *OCL* puede usarse de manera muy efectiva en la definición de transformaciones: una primera expresión en *OCL* especifica los elementos del modelo origen, y una segunda expresión describe los elementos en el modelo destino de la transformación. Las condiciones necesarias para aplicar una transformación también pueden expresarse usando *OCL*.

1.6 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*.

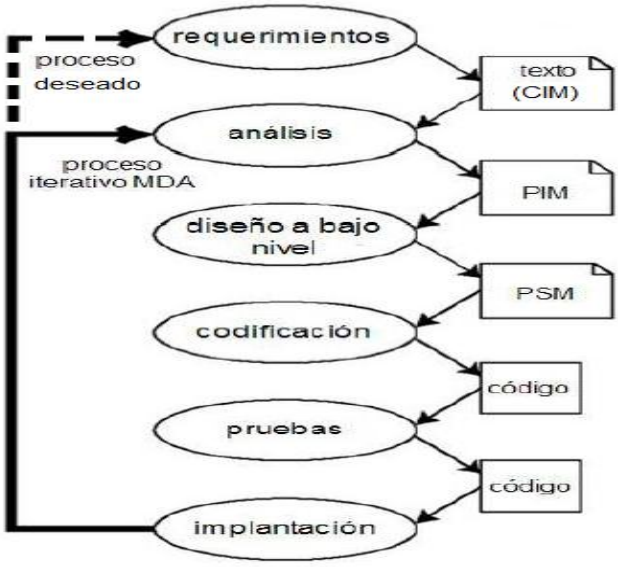


Figura 3: Etapas del desarrollo de software con MDA.
Fuente: Tomado de (Pons, 2010)

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):*** En *CIM* 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. (Pons, 2010)

2. Modelo Independiente de la Plataforma *(PIM):* Es una visión de un sistema en la que no se incluye la plataforma donde será implementado, y se busca ese grado de independencia con el fin de ser apto para usarlo en diferentes plataformas posteriormente y surgen como resultado de análisis y diseño. Tal modelo tiene cierto nivel de abstracción que no cambia; sin importar la plataforma que sea elegida para su implementación.(Pons, 2010). 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)*: Es un modelo de más bajo nivel de abstracción que el PIM que describe el sistema de acuerdo con una tecnología de implementación determinada. Su generación es automatizada y toma como punto de partida el PIM al que se le realizan transformaciones para generar uno o varios *PSMs* correspondientes a tecnologías de implementación, proporcionando una independencia entre la capa de negocio y la tecnología empleada.(Pons, 2010)

1.7 Análisis de herramientas CASE con soporte MDA

1.7.1 Herramientas *MDA* Propietarias

1.7.1.1 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.

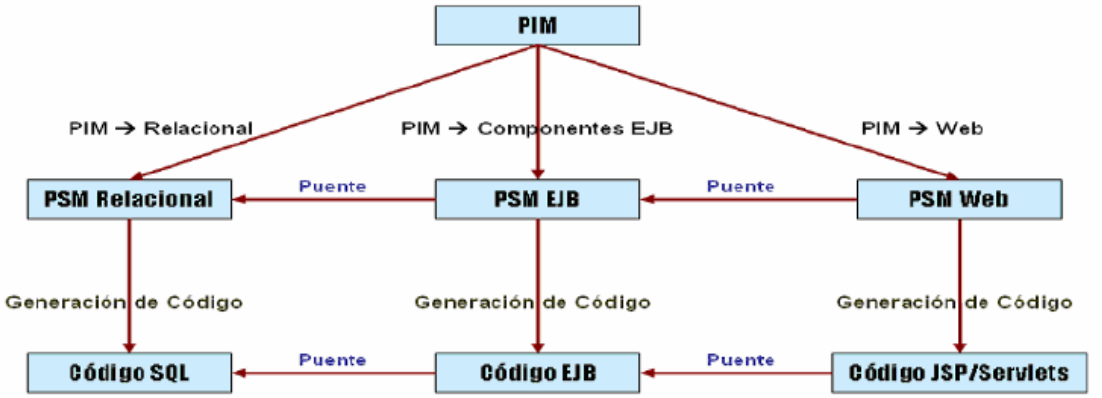


Figura 4: Conexión entre los modelos OptimalJ.

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.

1.7.1.2 ArcStyler

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*).

1.7.2 Herramientas *MDA* de código abierto

1.7.2.1 AndroMDA

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.

1.7.2.2 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.

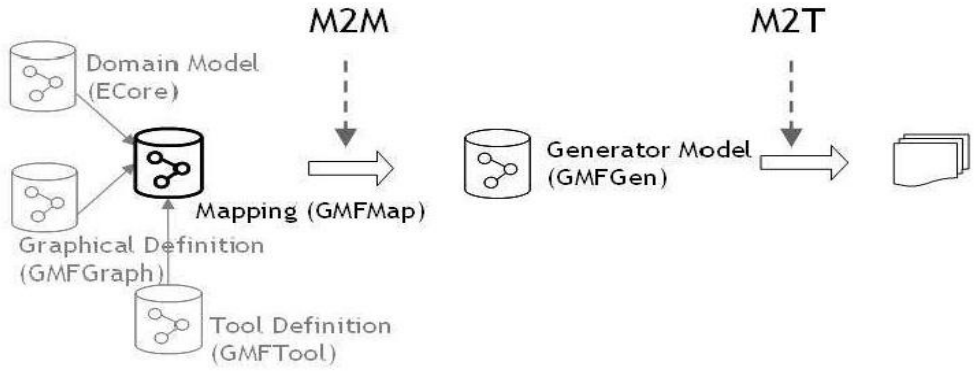


Figura 5: Arquitectura para el desarrollo basado en GMF.

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.

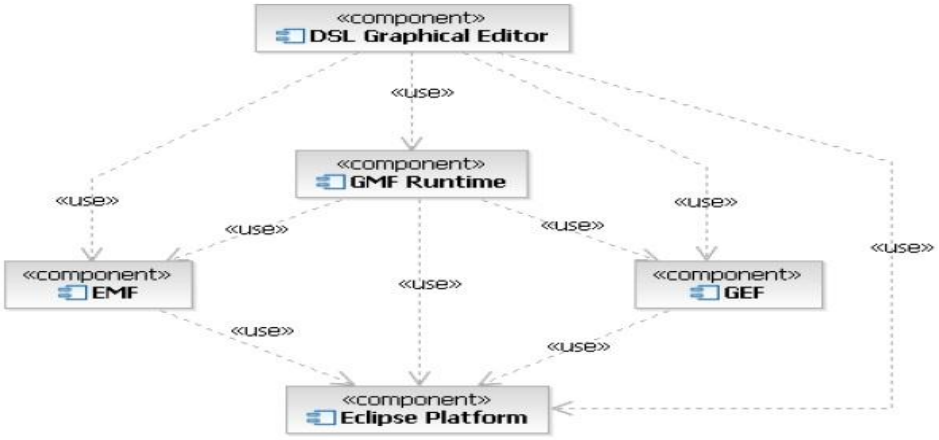


Figura 6: Dependencias de la plataforma GMF.

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.

1.7.3 Comparación entre 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.

1.8 Análisis de la herramienta *CASE jMDA* v1.0, v2.0 y v3.0

Para el desarrollo de la versión 4.0 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
P01	2	3	4
P02	2	3	4
P03	0	0	0
P04	0	0	0
P05	0	0	0
P06	0	0	0
P07	0	0	0
P08	2	3	4
P09	0	0	0
P10	0	0	0
P11	0	0	0
P12	0	0	0
P13	2	3	4
P14	1	1	1
P15	0	0	0
P16	1	1	1
Total	**10**	**14**	**18**

Como se puede observar en la Tabla 5, la herramienta jMDA fue evolucionando continuamente en las diferentes versiones. No obstante, los resultados evidencian que la estrategia de implementación utilizada en las tres versiones no ha permitido tener grandes avances, por lo que resulta imprescindible buscar otras alternativas de

implementación que posibiliten mejores resultados y que mantenga los elementos positivos de las versiones anteriores.

1.9 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)

1.10 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).

1.11 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 y 3.0 de la herramienta *CASE jMDA* presentan limitantes en las teorías de diseño e implementación utilizadas, elemento que impide su reutilización en la presente investigación. Es por ello que se decide crear una nueva herramienta que satisfaga realmente los objetivos de esta investigación.
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 y diseño del Módulo PIM-PSM 4.0 de la herramienta CASE jMDA"*

En este capítulo se realiza un abordaje de los principales elementos relacionados con las fases de análisis y diseño 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.

2.1 Diagramas del análisis y el diseño del módulo *PIM-PSM v4.0* de la herramienta *CASE jMDA*

Es bueno resaltar nuevamente que el presente trabajo tiene como alcance el desarrollo de un módulo de generación asistida por computadora del análisis y diseño de sistemas de información con el uso de diagramas de UML.

2.1.1 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.

2.1.2 Requisitos no funcionales del sistema

RNF 1: Interfaz del sistema: Una de las fortalezas de esta versión 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.

2.2 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 tres casos de uso principales "Modelar PIM", "Modelar PSM" y "Realizar transformación al modelo PSM"; los dos primeros extienden casos de uso correspondientes a cada uno de los diagramas contemplados en el sistema y que no necesariamente tienen todos que ser modelados para cada aplicación a desarrollar.

Sobre el tercer 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 herramienta.

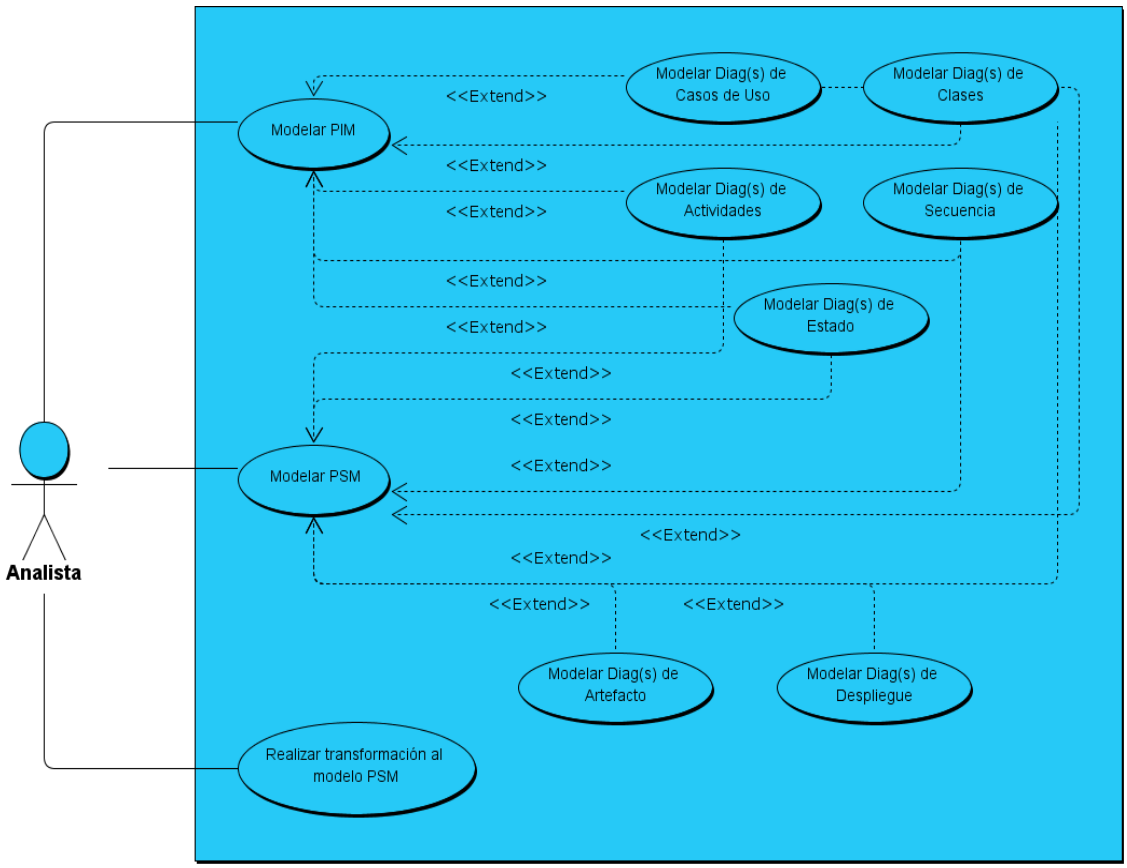


Figura 7: Diagrama de caso de uso del sistema que describe las principales funcionalidades de la herramienta jMDA PIM-PSM 4.0. Fuente: Elaboración propia.

2.2.1 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	

2.2.2 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: Modelar PIM

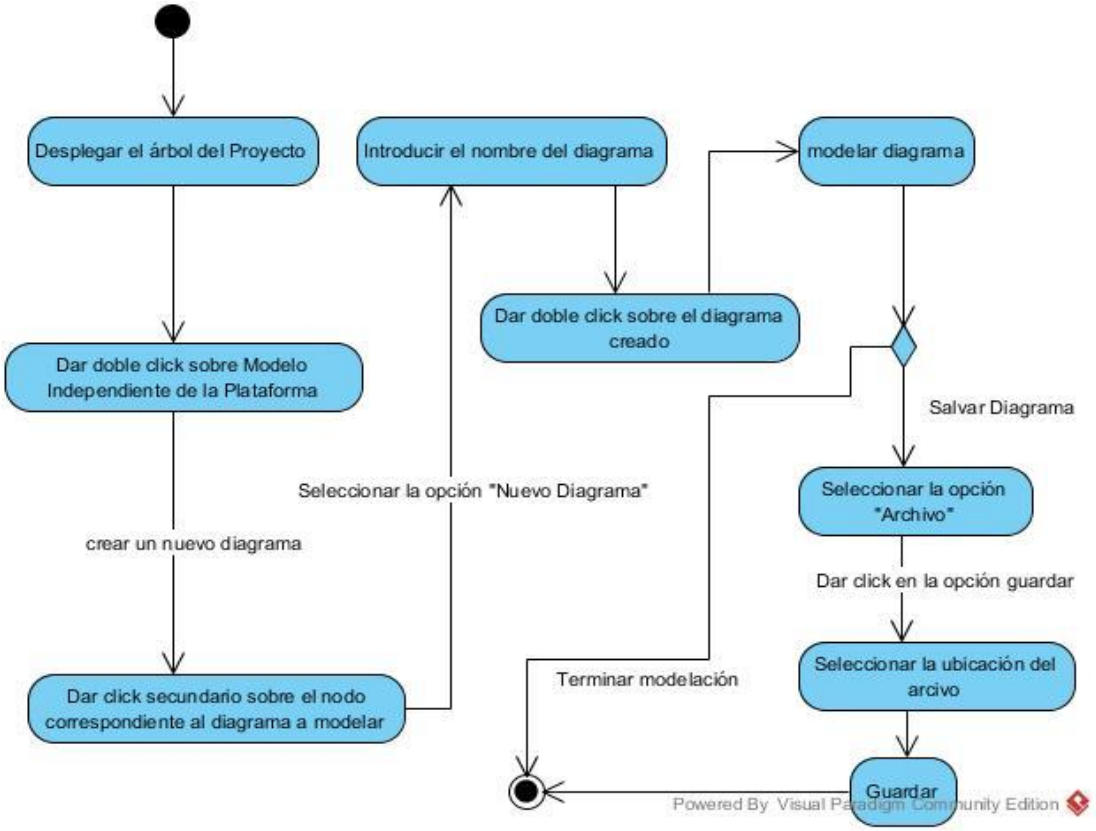


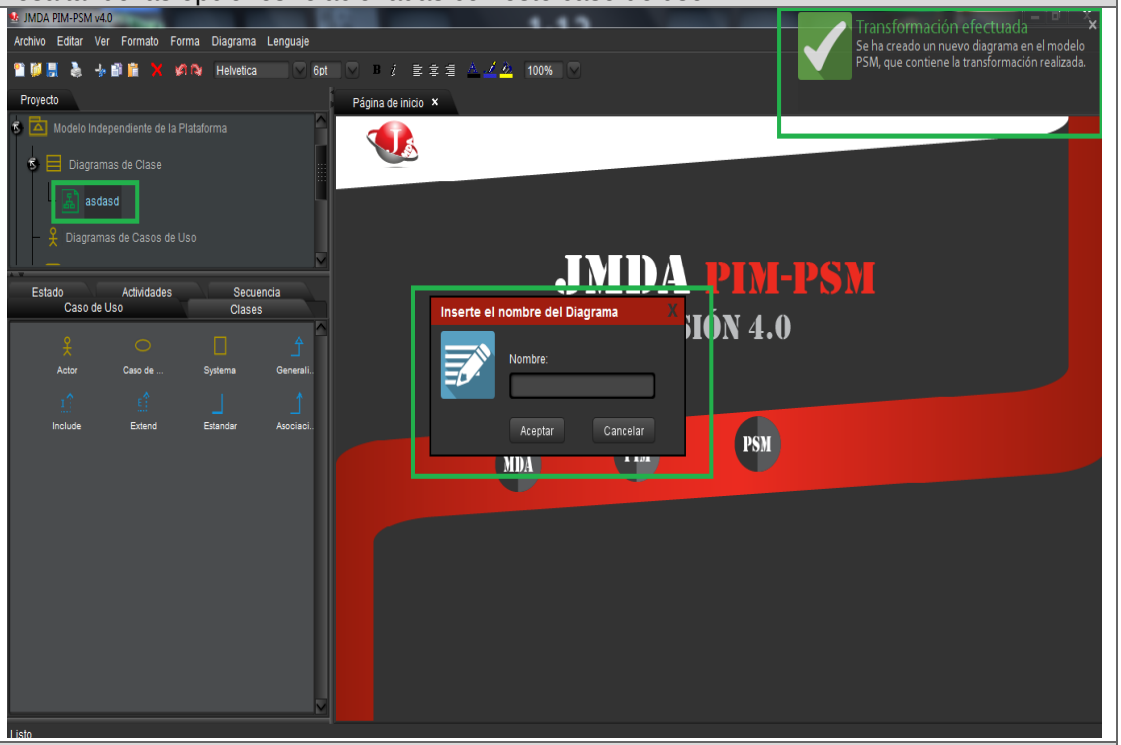
Figura 8 Diagrama de actividades para la modelación del PIM.

Fuente: Elaboración propia

La Figura 8 describe mediante un diagrama de actividades el caso de uso “Modelar PIM”, se hace alusión a las acciones que debe ejecutar el usuario para la creación de un nuevo diagrama sin tener en cuenta que tipo de diagrama que se crea. Luego de haber creado el diagrama el analista haciendo uso de los componentes correspondientes a este, realiza la modelación.

Tabla 7: Descripción del Caso de Uso del Sistema “Modelar PIM”.

Fuente: Elaboración propia.

Caso de uso del sistema	**Modelar PIM**
Actor	Analista
Propósito	Realizar el modelado de los diagramas correspondientes al modelo independiente de la plataforma.
Resumen	Inicia cuando el analista crea un nuevo diagrama en el árbol del proyecto, específicamente en los nodos correspondientes al modelo *(PIM)*. Luego de crear el diagrama con el nombre deseado el analista debe dar doble *click* sobre este nodo para cargar la hoja de trabajo en la pantalla.
Responsabilidades	Crear una hoja de trabajo en el nodo seleccionado en el modelo *(PIM)* 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 nodo correspondiente al modelo (PIM).	2. El sistema le muestra un menú con la opción “Nuevo Diagrama”.
3. El analista selecciona esta opción.	4. El sistema le muestra una ventana para que introduzca el nombre del diagrama.
5. El analista introduce el nombre del diagrama.	6. El sistema verifica que el nombre no coincide con ninguno de los diagramas anteriormente creados y crea el diagrama en el nodo correspondiente.
	7. El sistema muestra una notificación al analista de que el diagrama ha sido creado correctamente.
Flujo alternativo de eventos	
	6.1. El sistema encuentra una coincidencia con el nombre del diagrama.
	6.2. Muestra una notificación que explica el error provocado.
	6.3. Muestra una ventana para que introduzca un nombre nuevamente.
6.4. El analista introduce el nombre del diagrama.	6. El sistema verifica que el nombre no coincide con ninguno de los diagramas anteriormente creados y crea el diagrama en el nodo correspondiente.
	7. El sistema muestra una notificación al analista de que el diagrama ha sido creado correctamente.

Caso de uso: Modelar PSM

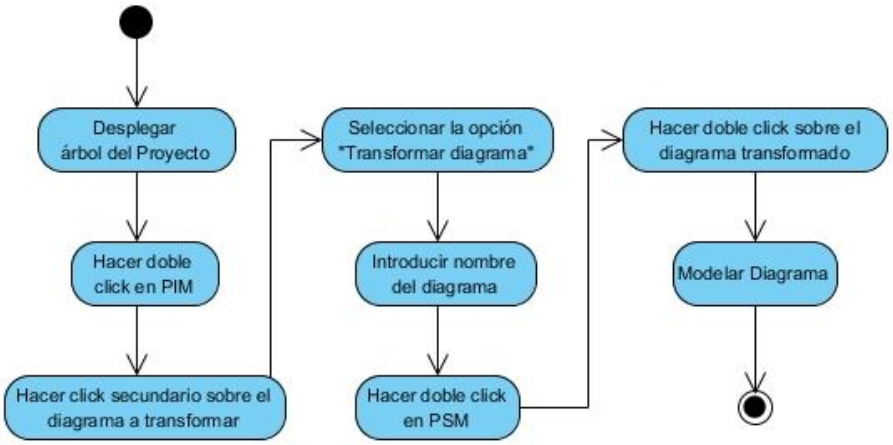


Figura 9: Diagrama de Actividades para la modelación del PSM. Fuente: Elaboración propia

La Figura 9 describe mediante un diagrama de actividades el caso de uso “Modelar PSM”, debido a que los modelos del *PSM* son el resultado de ejecutar una transformación a un diagrama del modelo *PIM*; este diagrama incluye el diagrama de la Figura 8 e incorpora las actividades a desarrollar para convertir el diagrama creado al modelo PSM.

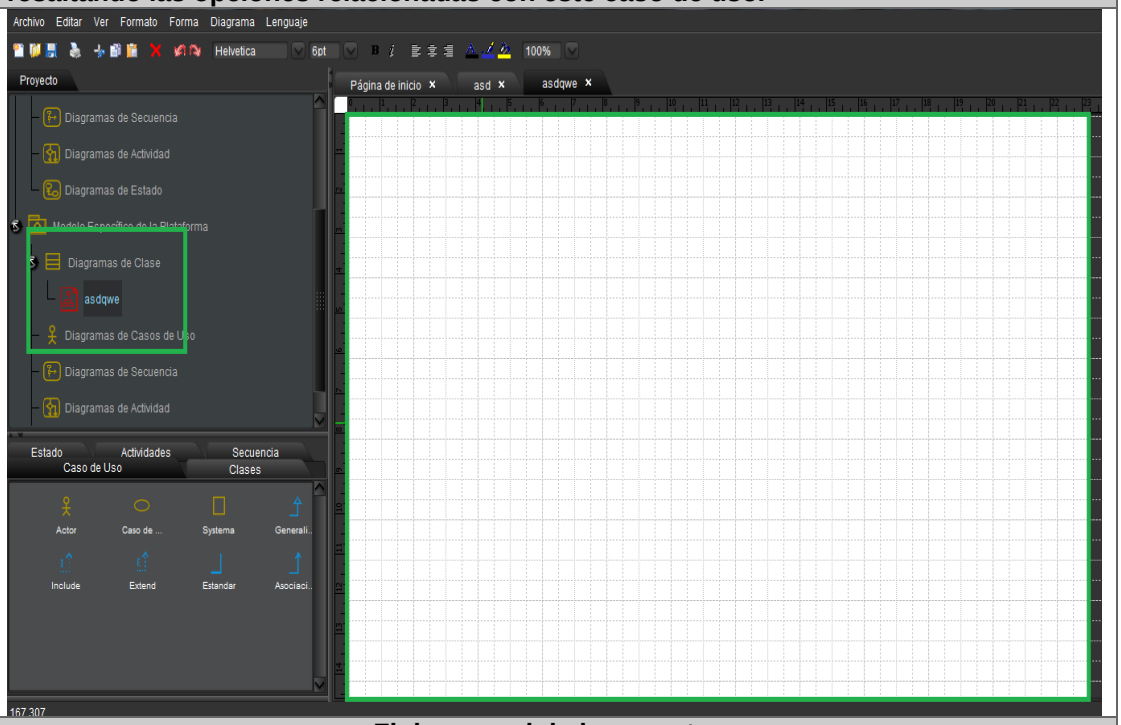
Tabla 8: Descripción del Caso de Uso del Sistema “Modelar PSM”.

Fuente: Elaboración propia.

Caso de uso del sistema	Modelar PSM
Actor	Analista

Propósito	Realizar la modelación de los diagramas correspondientes al modelo específico de la plataforma.
Resumen	Inicia cuando el analista crea un nuevo diagrama en el árbol del proyecto, específicamente en los nodos correspondientes al modelo *(PIM)*. Luego de crear el diagrama con el nombre deseado el analista debe dar doble *click* sobre este nodo para cargar la hoja de trabajo en la pantalla y realizar la modelación apoyándose en los componentes correspondientes al diagrama creado. Posteriormente el analista deberá transformar el diagrama al modelo *PSM,* y es entonces es cuando se puede comenzar a modelar en el *PSM*
Responsabilidades	Crear una hoja de trabajo en el nodo correspondiente en el modelo *(PSM)* y modelar un diagrama en este.
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 nodo correspondiente al modelo (PIM).	2. El sistema le muestra un menú con la opción “Nuevo Diagrama”.
3. El analista selecciona esta opción.	4. El sistema le muestra una ventana para que introduzca el nombre del diagrama.
5. El analista introduce el nombre del diagrama.	6. El sistema verifica que el nombre no coincide con ninguno de los diagramas anteriormente creados y crea el diagrama en el nodo correspondiente.
	7. El sistema muestra una notificación anunciando que el diagrama se ha creado correctamente
8. El analista hace doble click sobre el diagrama creado en el árbol del proyecto.	9. El sistema muestra la hoja de trabajo correspondiente al diagrama creado.

10. El analista realiza la modelación deseada.	11. El sistema permite modelar al analista haciendo uso de los componentes correspondientes al diagrama seleccionado.
12. El analista hace *click* secundario sobre el diagrama modelado.	13. El sistema muestra la opción “Transformar Diagrama”
14. El analista selecciona esta opción, haciendo *click* primario sobre ella.	15. El sistema muestra una ventana para que el analista introduzca el nombre del diagrama transformado al *PSM*
16. El analista introduce el nombre del diagrama	17. El sistema transforma el diagrama y crea un nodo en el árbol del proyecto correspondiente a este.
	18. El sistema muestra una notificación de la transformación correcta del diagrama
19. El analista hace doble *click* sobre el diagrama transformado.	20. El sistema carga la hoja de trabajo correspondiente al diagrama seleccionado.
Flujo alternativo de eventos 1	
	6.1. El sistema encuentra una coincidencia con el nombre del diagrama.
	6.2. Muestra una notificación que explica el error provocado.
	6.3. Muestra una ventana para que introduzca un nombre nuevamente.
6.4. El analista introduce el nombre del diagrama.	6. El sistema crea el diagrama correspondiente.
	7. El sistema muestra una notificación al analista de que el diagrama ha sido creado correctamente.
Flujo alternativo de eventos 2	
	17.1. El sistema encuentra una coincidencia con el nombre del diagrama.
	17.2. Muestra una notificación que explica el error provocado.
	17.3. Muestra una ventana para que introduzca un nombre nuevamente.
17.4. El analista introduce el nombre del diagrama.	17. El sistema transforma el diagrama y crea un nodo en el árbol del proyecto correspondiente a este.
19. El analista hace doble *click* sobre el diagrama transformado.	18. El sistema muestra una notificación de la transformación correcta del diagrama
	20. El sistema carga la hoja de trabajo correspondiente al diagrama seleccionado.

Caso de uso: Realizar Transformación

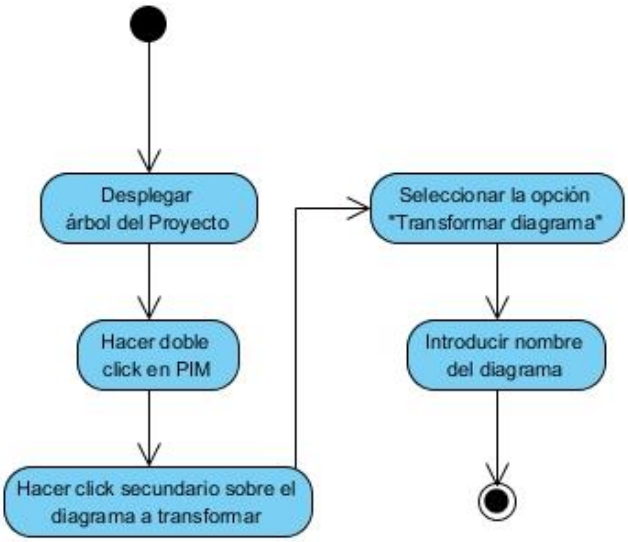


Figura 10: Diagrama de actividades caso de uso "Realizar transformación".

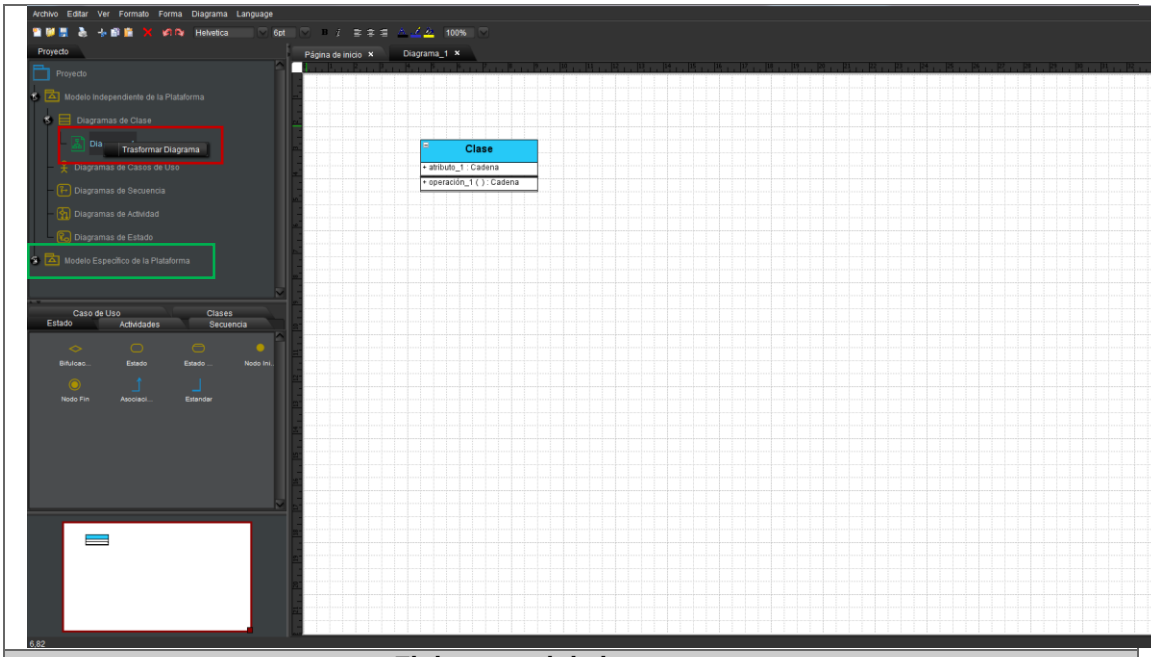
Fuente: Elaboración propia

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 9 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.

2.3 Diagrama de paquetes de la aplicación.

La figura 11 muestra el diagrama de paquetes de la herramienta jMDA PIM-PSM v4.0. 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.

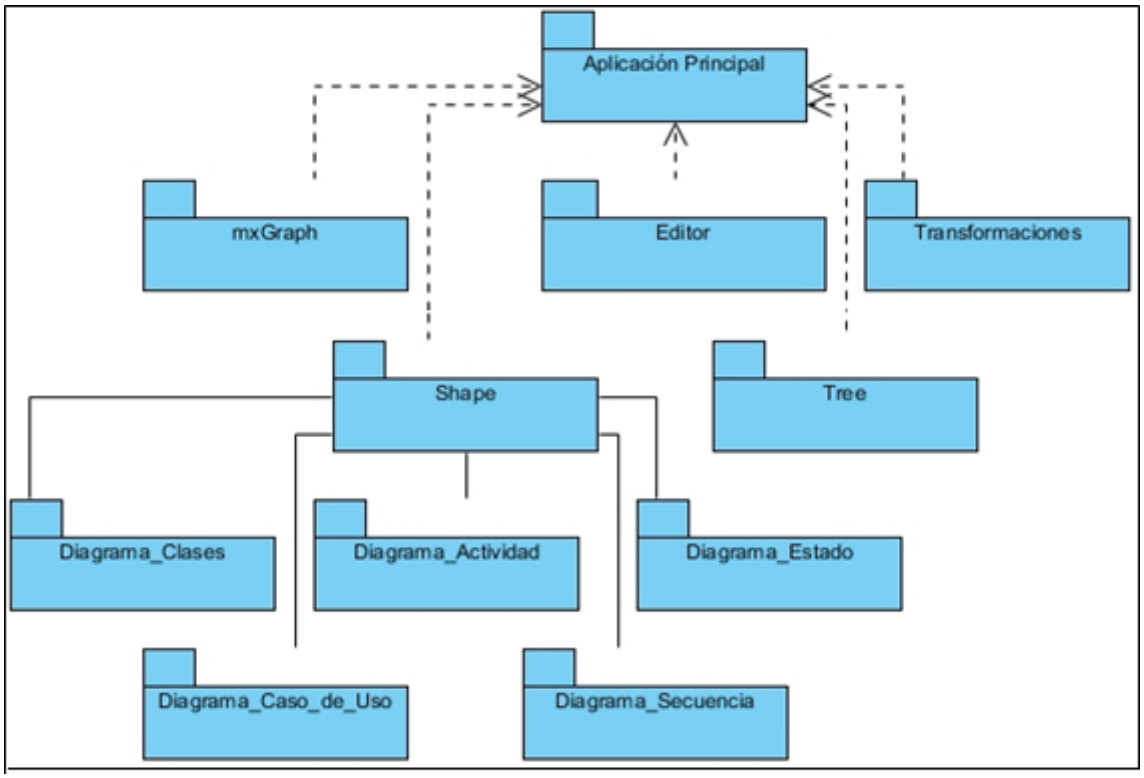


Figura 11: Diagrama de paquetes de la herramienta jMDA PIM-PSM v4.0.

Fuente: Elaboración propia

2.4 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:

- **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:

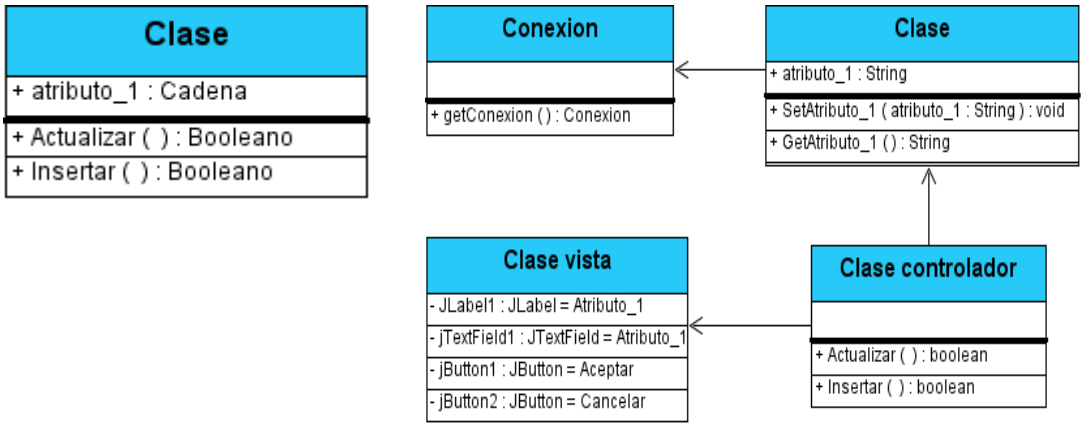


Figura 12: Ejemplo de transformación

Fuente 1: Elaboración propia

En la figura 12 muestra que las transformaciones se realizan para la plataforma java específicamente, por lo que los tipos de los atributos y métodos son cambiados en función de esto.

- **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 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.

- **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.

- **Regla 6:** Se crean diagramas de actividades que describen las principales operaciones que se realizan para gestionar la información de las clases del modelo PIM.

Estos diagramas de actividades describen operaciones como actualizar, eliminar e insertar un registro en la base de datos y son asociados una clase en el modelo *PSM*.

Como se puede apreciar en las reglas anteriormente planteadas, se utiliza una combinación de los dos enfoques de transformaciones: por tipos y por instancias.

2.4.1 Algoritmo de transformación utilizado

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.

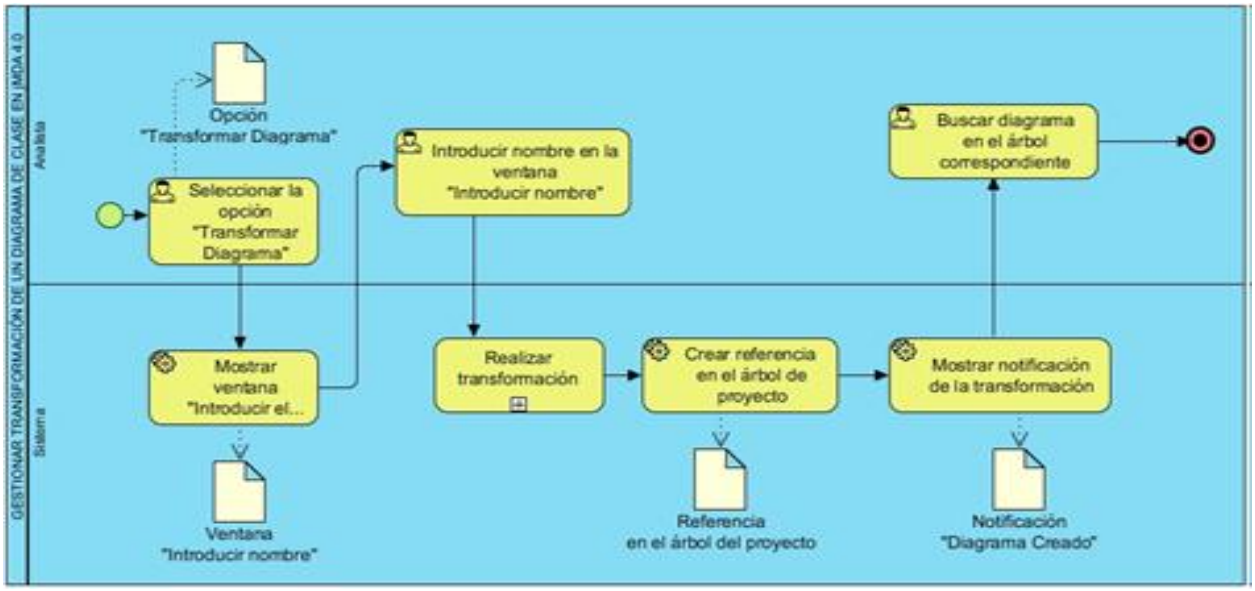


Figura 13: Definición del algoritmo de transformación de diagramas de clases utilizando la notación BPMN. Fuente: Elaboración propia

Como se puede apreciar en la Figura 13 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, estas acciones se describen detalladamente en la Figura 14. Luego de realizar la transformación el sistema se encarga de crear una referencia hacia el diagrama transformado de modo tal que el usuario pueda abrirlo dirigiéndose al nodo correspondiente a los diagrama de clases en el modelo *PSM*.

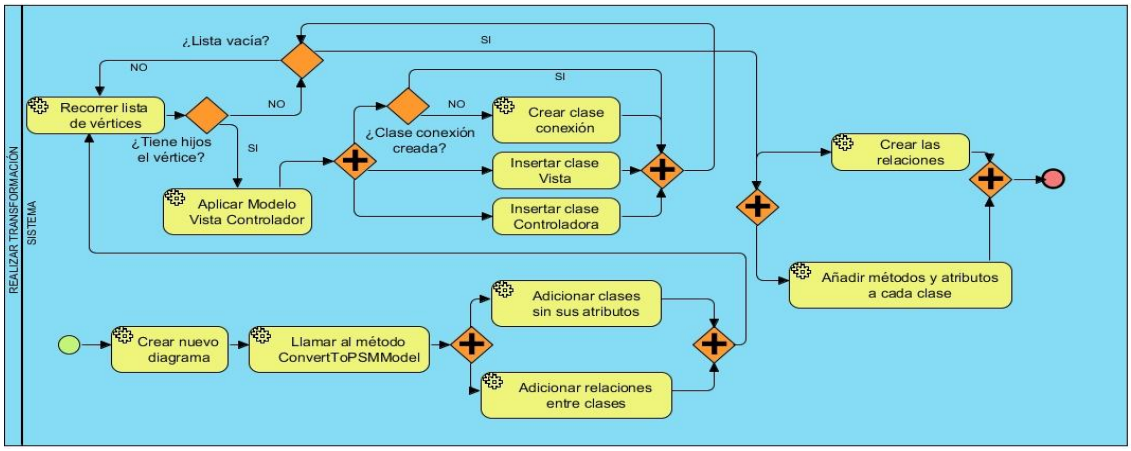


Figura 14: Acciones ejecutadas por el sistema para realizar una transformación modeladas en BPMN. Fuente: Elaboración propia

La Figura 14 describe el proceso “Realizar transformación” modelado en la figura 12. Como se puede apreciar en la imagen para realizar las transformaciones se aplica el patrón de diseño MVC, y en consonancia con esto se crean las clases pertinentes. Es muy importante destacar que el algoritmo aplica el modelo vista controlador si el vértice que se va a transformar contiene atributos u operaciones, pues en caso contrario no tiene lógica aplicarlo.

2.5 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.
3. Se describió el diagrama de paquetes que estructura el sistema, y se comenta cada uno de sus componentes para lograr un mejor entendimiento.
4. Se definió un grupo de reglas tomadas en cuenta a la hora de realizar las transformaciones del modelo PIM al modelo *PSM*.
5. Se diseñó el algoritmo de transformación que se utilizará en la conversión del diagrama de clases del modelo *PIM* al modelo *PSM*.

CAPÍTULO 3

Capítulo 3: *"Descripción de la propuesta de solución para la implementación del módulo PIM-PSM 4.0 de la herramienta CASE jMDA."*

En el presente capítulo 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.

3.1 Descripción del proceso de implementación.

El módulo *PIM-PSM* v4.0 de la herramienta *CASE jMDA* desarrollado en la presente tesis tiene en común con las primeras versiones 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 cambiar la estrategia de implementación utilizada en las versiones anteriores, debido a que el diseño utilizado para las versiones anteriores no permite cumplimentar los objetivos trazados al iniciar el desarrollo de la versión 4.0. Se utiliza la idea de separar en paquetes cada una de las formas correspondientes a los diagramas ya que esto permite que las modificaciones se realicen de forma modular.

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.

3.1.1 Arquitectura del sistema.

Para su comprensión el diagrama de clases, se divide por paquetes y en forma de metamodelo debido a la gran cantidad de atributos y métodos que poseen las clases de este sistema.

3.1.1.1 Diagrama de clases del paquete *Tree*

En este paquete la clase principal se denomina "*createTree*", pues tiene la responsabilidad de crear el árbol del proyecto y añadir los menús correspondientes a cada uno de sus nodos.

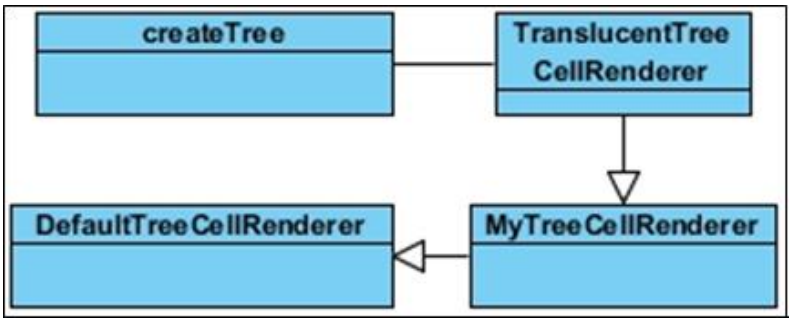


Figura 15: Diagrama de clases del paquete "Tree".

Fuente: Elaboración propia

En la Figura 15 se muestra el diagrama de clases del paquete "*Tree*", resulta importante destacar que además de manejar la lógica de cada uno de los nodos del árbol la clase "createTree" se encarga a través de la clase "*TranslucentTreeCellRender*" de manejar la visualización de cada uno de sus nodos; pues esta varía en dependencia de la clase a la que represente. Esto no sería posible sin el paquete "*model*", que no se presenta en el diagrama por cuestiones de espacio, pero que contiene el modelo de cada uno de los nodos creados en el árbol del proyecto.

3.1.1.2 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.

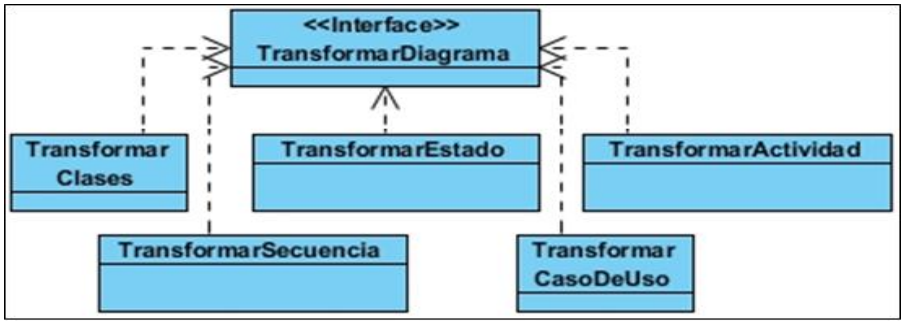


Figura 16: Diagrama de clases del paquete "Transformaciones". Fuente: Elaboración propia

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.

3.1.1.3 Diagrama de clases del paquete *Editor*

En este paquete existen varias clases importantes para la aplicación, una de ellas es la clase “*GraphEditor*”, la cual tiene la responsabilidad de configurar y ubicar en la aplicación todos los componentes que la integran. Así como manejar las acciones sobre el árbol del proyecto que se encargan de crear una nueva pestaña en el área de modelación.

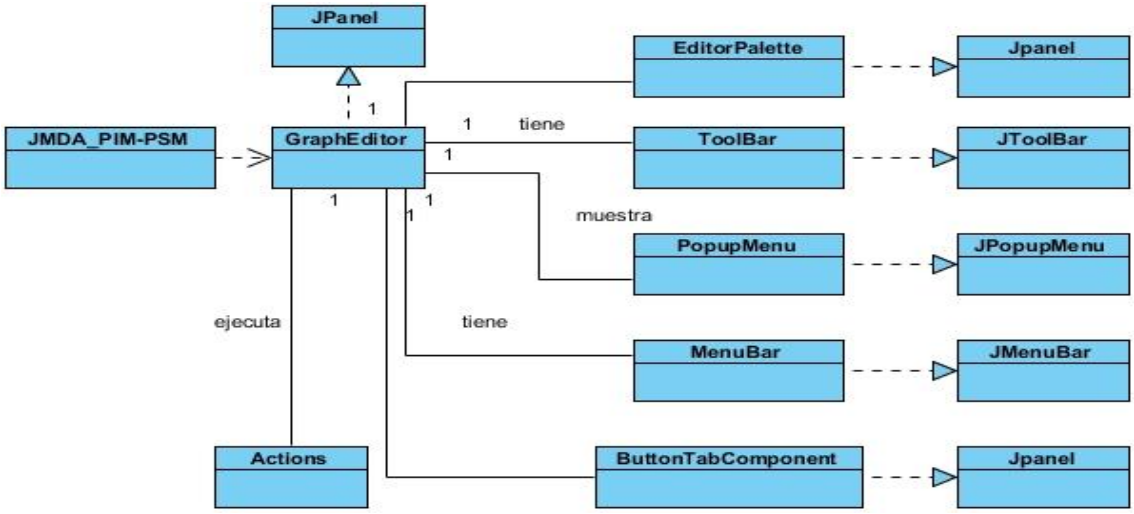


Figura 17: Diagrama de clases del paquete "Editor".

Fuente: Elaboración propia

En la Figura 17 se muestra el diagrama de clases del paquete “Editor”, la clase de mayor importancia es “*GraphEditor*” que se encarga de inicializar cada uno de los componentes que integran la aplicación y de ubicarlos en esta. Otra clase de vital importancia es la clase “*Actions*” que contiene la lógica de cada una de las opciones que brinda el sistema.

3.2 Diagrama de secuencia de los casos de uso significativos

A continuación se representan los casos de uso más significativos mediante diagramas de secuencia, en los que se describe la interacción del analista con las diferentes interfaces y las clases responsables de llevar a cabo las operaciones.

3.2.1 Diagramas de secuencia del caso de uso "Modelar *PIM*"

El siguiente diagrama de secuencia describe las acciones del analista y la respuesta del sistema cuando se inicia el caso de uso modelar *PIM*.

Creación de un nuevo diagrama:

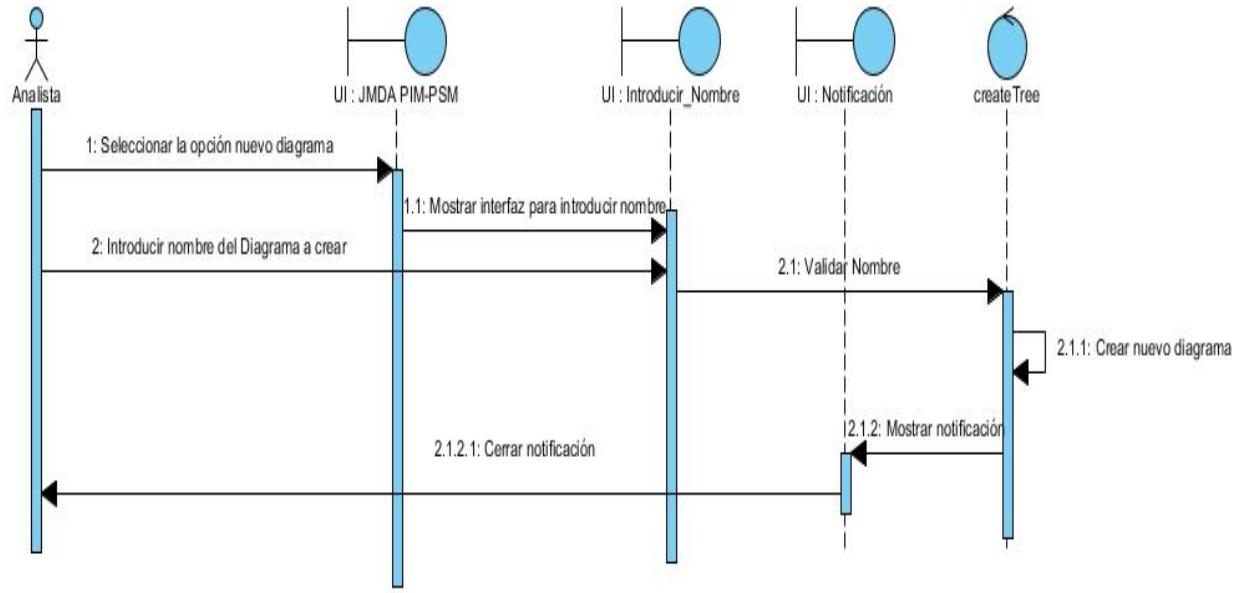


Figura 18: Diagrama de secuencia que describe el escenario de creación de un nuevo diagrama.

Fuente: Elaboración propia

En la Figura 18 se muestra el diagrama de secuencia que describe el proceso de creación de un nuevo diagrama en el modelo *PIM*. En este diagrama se muestran de forma clara y concisa las interfaces con las que interactúa el usuario, así como la clase encargada de crear el diagrama en el modelo *PIM.*

Abrir especificación de una clase:

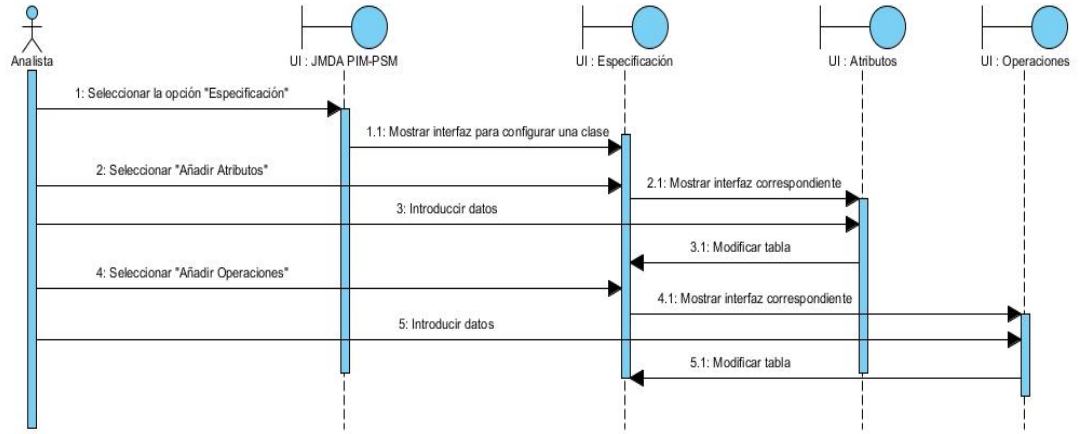


Figura 19: Diagrama de secuencia que describe el proceso de modificar los datos de una clase determinada. Fuente: Elaboración propia

En la Figura 19 se muestra mediante un diagrama de secuencias las interfaces con las que interactúa el analista cuando decide “Abrir Especificación” de una clase determinada, opción que se mostrará al hacer *click* secundario sobre una clase. Estas interfaces de una forma sencilla permiten al usuario añadir los atributos y métodos pertenecientes a la clase en cuestión.

3.2.2 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”.

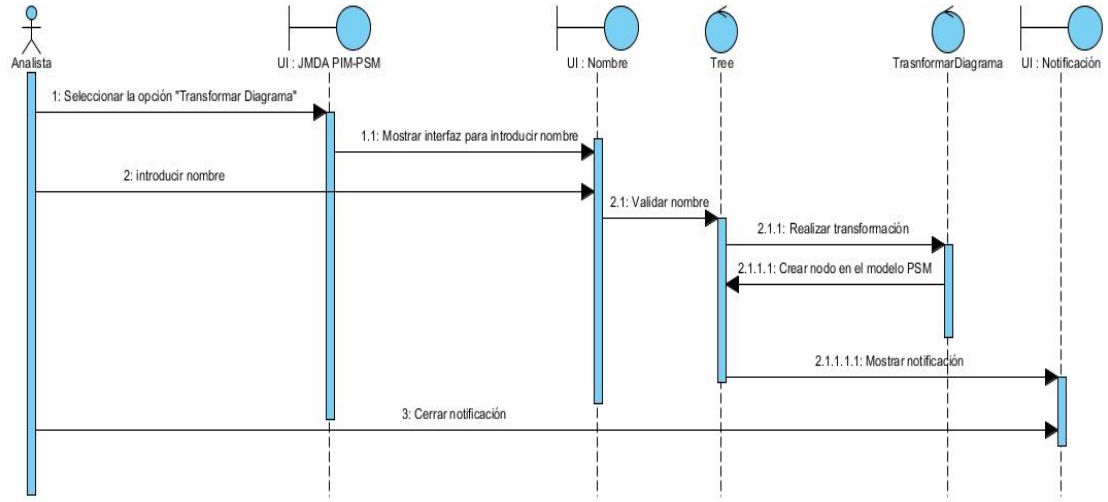


Figura 20: Diagrama de secuencia que describe la opción "Transformar diagrama". Fuente: Elaboración propia.

En la Figura 20 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.

3.3 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 esta versión de la herramienta se presta especial atención a este aspecto. En consonancia con esto se crea un área de notificaciones donde se mostrará al usuario la información asociada a la validación de los datos y a los posibles errores que surjan en la aplicación.

En la Figura 21 se muestra el ejemplo de una notificación de error, ocasionada por insertar un nombre de un diagrama que ya existía en el árbol del proyecto.

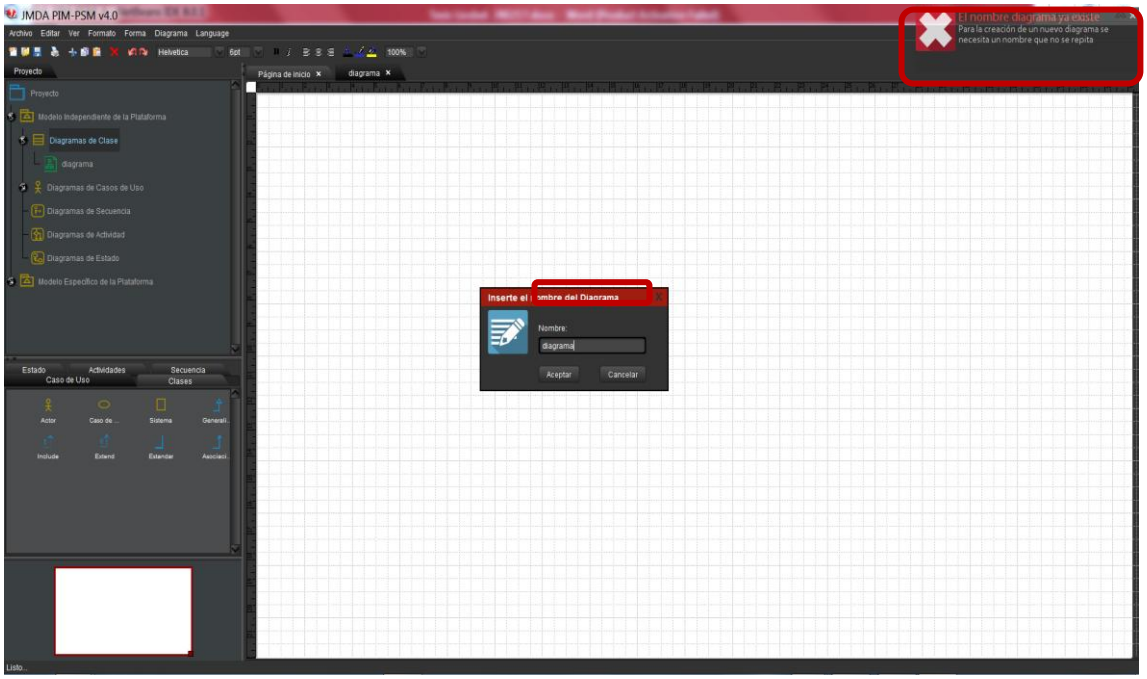


Figura 21: Notificación de error. Fuente: Elaboración propia

Como se puede ver en la Figura 21 los errores y demás notificaciones son mostradas en la parte superior derecha de la pantalla, estas notificaciones tienen la particularidad de que si no se cierran por el botón "X" que tiene, luego de pasados 6 segundos se ocultan automáticamente; permitiendo que el usuario tenga tiempo de leerlas pero que no tenga que estar constantemente cerrando notificaciones lo que entorpece la tarea de modelado.

3.4 Diagrama de componentes

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

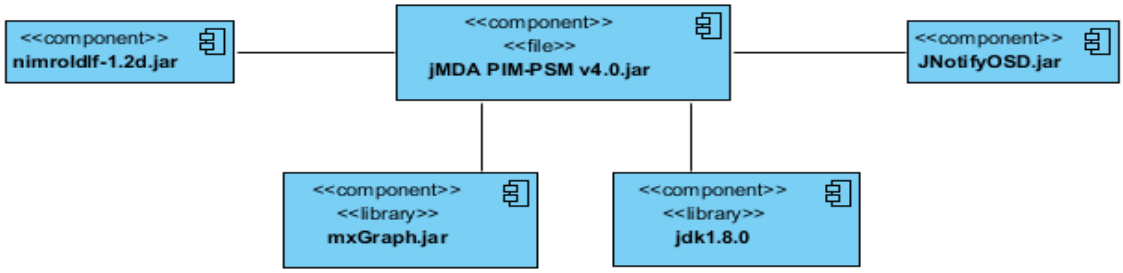


Figura 22: Diagrama de componentes de la herramienta CASE JMDA PIM-PSM 4.0.

Fuente: Elaboración propia

Como se puede observar en la Figura 22 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.*

3.5 Conclusiones del capítulo

1. Se realizó el diseño y descripción de los diagramas de clases; separándolos según el paquete para lograr una mejor comprensión.
2. Se describieron los casos de uso significativos de la aplicación por medio de diagramas de secuencia.
3. Se analizó el procedimiento requerido para la realización del tratamiento de errores.
4. Se diseñó el diagrama de componentes que define la arquitectura del sistema.

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) .

4.1.1 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

4.1.2 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 9: 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 9 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:

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

$$UAW = 1 \times 0 + 2 \times 0 + 3 \times 1 = 3$$

4.1.3 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.

Tabla 10: 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 10 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

4.1.4 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

4.1.5 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 11: Factores de complejidad técnica. Fuente: Adaptado de (Fernández 2012)

Número factor	Descripción	Peso	Valor	Factor
T1	Sistema Distribuido	2	2	4
T2	Tiempo de respuesta	1	5	5
T3	Eficiencia por el usuario	1	5	5
T4	Proceso interno complejo	1	5	5
T5	Reusabilidad	1	5	5
T6	Facilidad de instalación	0.5	5	2.5
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	3	3
T12	Acceso directo a terceras partes	1	2	2
T13	Facilidades especiales de entrenamiento a usuarios finales	1	3	3
Total Factor				50

En la Tabla 11 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* 50

TCF = 1.1

4.1.6 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:

Tabla 12: 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 12 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

4.1.7 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)

4.1.8 Estimación del esfuerzo del proyecto

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

Tabla 13: Distribución del esfuerzo. Fuente: Elaboración propia.

Actividad	*Porcentaje*
Análisis	*15.00%*
Diseño	*15.00%*
Programación	*50.00%*
Pruebas	*10.00%*
Sobrecarga(otras actividades)	*10.00%*

Con la distribución mostrada en la Tabla 13 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.

Tabla 14: Distribución del esfuerzo en el proyecto. Fuente: Elaboración propia.

Actividad	*Porcentaje*	*Horas / hombre*
Análisis	*15.00%*	100.716
Diseño	*15.00%*	100.716
Programación	*50.00%*	335.72
Pruebas	*10.00%*	67.144
Sobrecarga(otras actividades)	*10.00%*	67.144
Total		671.44

4.1.9 Cálculo del esfuerzo total

Ecuación:	Donde:
Etotal=∑ actividades	**ETotal**: esfuerzo total
Etotal= 671.44 horas/hombres	

4.1.10 Cálculo del tiempo de desarrollo

Ecuación:	Donde:
TDesarrollo=ETotal/CHTotal/CHTrabajo TDesarrollo=671.44 /1/8 TDesarrollo= 84 días aproximadamente	**TDesarrollo**: tiempo de desarrollo total en horas **CHTotal**: cantidad total de hombres **CHTrabajo**: cantidad de horas de trabajo diario

4.1.11 Cálculo del costo

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

4.2 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 15 se presenta se presentan un conjunto de precondiciones que deben cumplirse para la ejecución de los escenarios de prueba que se proponen.

Tabla 15: 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.

4.2.1 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.

Tabla 16: 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*

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

Tabla 17: 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.

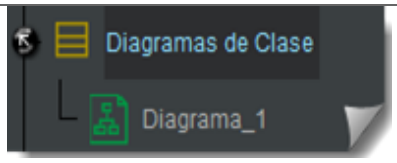
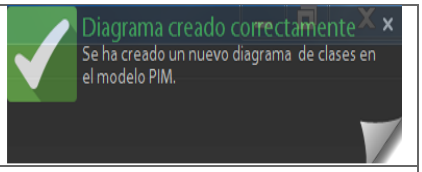
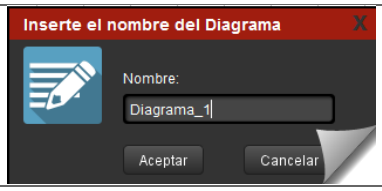
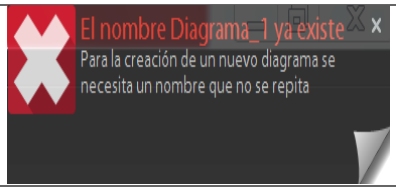
	 
Observaciones	No se encontró ningún problema en la prueba realizada.

Tabla 18: 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.

4.2.2.2 Escenario 2: Gestión de formas.

Tabla 19: Escenario para la adición de una forma en un área de trabajo caso 1. Fuente: Elaboración propia

Escenario 2 Caso 1	
Nombre del proceso	Adición de una forma a un área de trabajo
Descripción	El analista selecciona una forma y con el *click* primario sostenido arrastra dicha forma hasta el área de trabajo seleccionada, al soltar el *click* la forma se dibuja en esa posición.
Escenario	El analista hace *click* primario sostenido sobre una forma de la paleta de componentes de los diagramas de caso de uso.
Entradas	No existen entradas de datos
Pre-condiciones	Existe un diagrama de caso de uso en el árbol del y su área de trabajo se encuentra cargada en la pantalla.

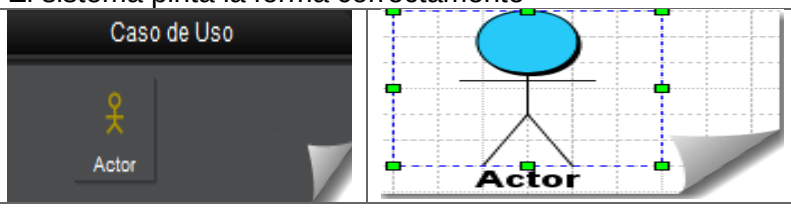
Procedimiento	1- El analista selecciona la forma a dibujar. 2- El analista hace *click* primario sostenido sobre la forma. 3- El sistema muestra la imagen de la forma seleccionada. 4- El analista deja de presionar el *click* en el lugar donde desea pintarla. 5- El sistema pinta la forma en el área de trabajo.
Resultado	El sistema pinta la forma correctamente 
Observaciones	No se encontró ningún problema en la prueba realizada.

Tabla 20: Escenario para la adición de una forma en un área de trabajo caso 2.

Fuente: Elaboración propia.

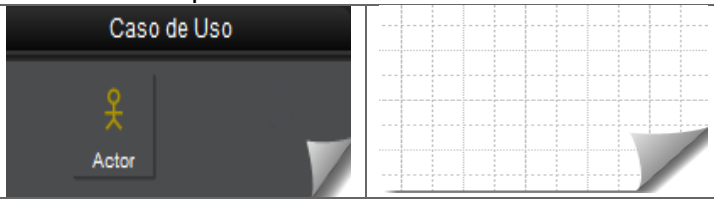
Escenario 2 Caso 2	
Nombre del proceso	Adición de una forma a un área de trabajo
Descripción	El analista selecciona una forma y con el *click* primario sostenido arrastra dicha forma hasta el área de trabajo seleccionada, al soltar el *click* la forma se dibuja en esa posición.
Escenario	El analista hace *click* primario sostenido sobre una forma de la paleta de componentes de los diagramas de caso de uso.
Entradas	No existen entradas de datos
Pre-condiciones	Existe un diagrama de caso de uso en el árbol del y su área de trabajo se encuentra cargada en la pantalla.
Procedimiento	1- El analista selecciona la forma a dibujar. 2- El analista hace *click* primario sostenido sobre la forma. 3- El sistema muestra la imagen de la forma seleccionada. 4- El analista deja de presionar el *click* fuera del área de trabajo. 5- El sistema no pinta la forma seleccionada.
Resultado	El sistema no pinta la forma 
Observaciones	No se encontró ningún problema en la prueba realizada.

Tabla 21: Escenario para la modificación del color de fondo de una forma.

Fuente: Elaboración propia.

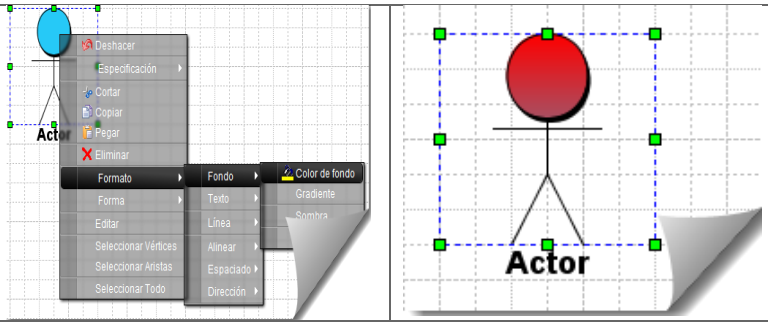
Escenario 2 Caso 3	
Nombre del proceso	Modificación del color de fondo de una forma.
Descripción	El analista hace *click* derecho sobre sobre una forma, y a través del menú, Formato/Fondo/Color de Fondo, cambia el color de fondo de la forma.
Escenario	El analista hace *click* secundario sobre una forma dibujada en el área de trabajo.
Entradas	No existen entradas de datos
Pre-condiciones	Existe un área de trabajo con una forma añadida.
Procedimiento	1- El analista hace *click* secundario sobre la forma. 2- El sistema muestra un menú con las opciones habilitadas. 3- El analista accede al menú Color de Fondo a través de la ruta Formato/Fondo/Color de Fondo. 4- El sistema muestra un *JColorChooser*. 5- El analista selecciona un color y presiona el botón aceptar.
Resultado	El sistema cambia el color de fondo de la forma. 
Observaciones	No se encontró ningún problema en la prueba realizada.

Tabla 22: Escenario para la modificación del tipo de letra de una forma.

Fuente: Elaboración propia.

Escenario 2 Caso 4	
Nombre del proceso	Modificación del tipo de letra de una forma.
Descripción	El analista hace *click* primario sobre una forma, y a través de la barra de herramientas selecciona la fuente que desea ponerle a la forma.
Escenario	Escenario para la modificación del tipo de letra de una forma determinada.
Entradas	No existen entradas de datos
Pre-condiciones	Existe un área de trabajo con una forma añadida.
Procedimiento	1- El analista hace *click* primario sobre la forma. 2- El analista selecciona en la barra de herramientas el tipo de letra que desea. 3- El sistema actualiza el tipo de letra de la forma.
	El sistema cambia el tipo de letra correctamente.

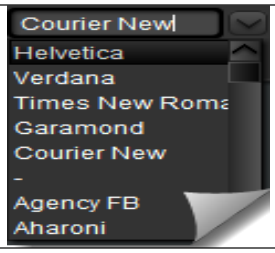
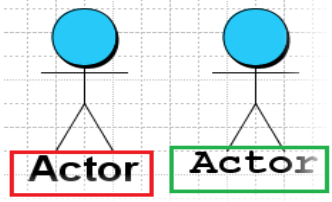
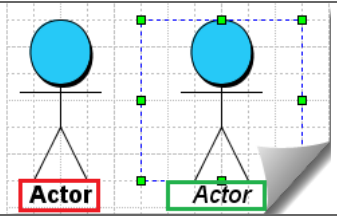
Resultado		
Observaciones	No se encontró ningún problema en la prueba realizada.	

Tabla 23: Escenario para la modificación del estilo de letra de una forma.

Fuente: Elaboración propia

Escenario 2 Caso 5	
Nombre del proceso	Modificación del estilo de letra de una forma.
Descripción	El analista hace *click* primario sobre una forma, y a través de la barra de herramientas selecciona el estilo de letra que desea ponerle a la forma.
Escenario	Escenario para la modificación del estilo de letra de una forma determinada.
Entradas	No existen entradas de datos
Pre-condiciones	Existe un área de trabajo con una forma añadida.
Procedimiento	1- El analista hace *click* primario sobre la forma. 2- El analista selecciona en la barra de herramientas el estilo de letra que desea. 3- El sistema actualiza el tipo de letra de la forma.
Resultado	El sistema cambia el tipo de letra correctamente.  
Observaciones	No se encontró ningún problema en la prueba realizada.

4.2.2.3 Escenario 3: Especificación de una clase en el modelo *PIM*

Tabla 24: 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

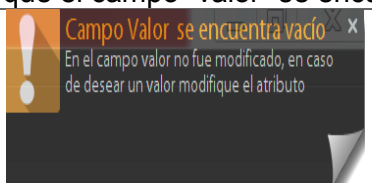
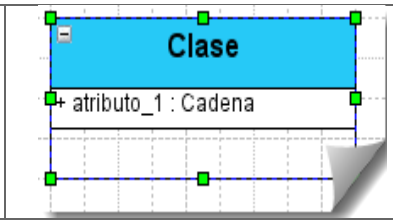
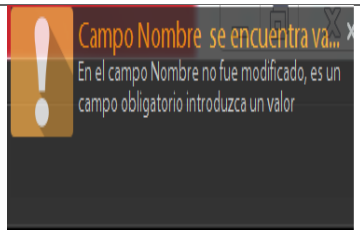
	Valor: -
Pre-condiciones	Existe un área de trabajo con una forma de tipo clase añadida.
Procedimiento	1- El analista hace *click* secundario sobre la forma. 2- El analista selecciona la opción “Abrir especificación” a través del menú “Especificación”. 3- El analista en la paleta correspondiente a los 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 25: 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.

4.2.2.4 Pruebas del sistema

Como se explicaba en los capítulos anteriores la creación de un nuevo diagrama en la versión 4.0 de la herramienta JMDA es un proceso sencillo; primeramente el analista debe seleccionar el nodo en el árbol del proyecto correspondiente a al diagrama que desea modelar, luego hacer click secundario y seleccionar la opción “nuevo diagrama” e introducir el nombre del diagrama creado.

Luego de la creación de un nuevo diagrama lo que sigue es la modelación de este, a continuación se muestran ejemplos de la modelación de los diferentes diagramas implementados en la versión 4.0 del módulo PIM-PSM de la herramienta JMDA.

Diagrama de Casos de Uso

El diagrama de casos de uso fue implementado nuevamente, logrando una mejor representación de sus componentes y facilitando la diagramación del usuario. En la Figura 23 se muestra un diagrama de caso de uso que describe un sistema para controlar la información asociada a los estudiantes, profesores y facultades de una universidad.

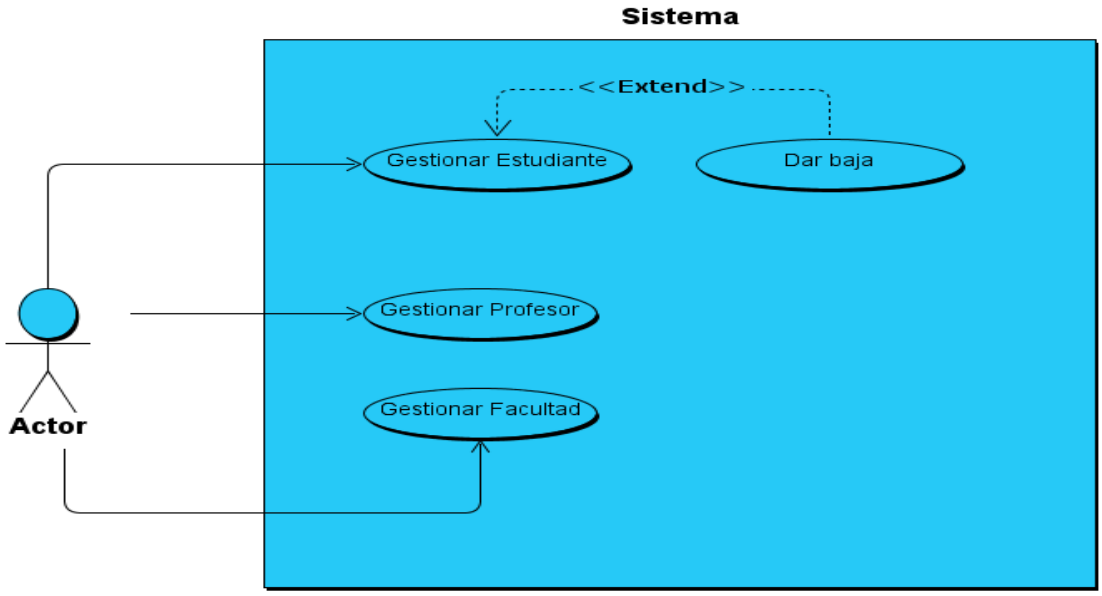


Figura 23: Diagrama de caso de uso modelado en la herramienta prototípica JMDA

Fuente: Elaboración propia

Diagrama de clases modelo PIM

El diagrama de clases fue implementado nuevamente, manteniendo los logros alcanzados en las anteriores versiones e incorporándole nuevas funcionalidades. En esta nueva versión se realiza la transformación del modelo PIM al modelo PSM aplicando el patrón de diseño MVC para generar el diagrama de clases en el modelo PSM en función de la plataforma java específicamente.

En la Figura 24 se muestra un diagrama de clases del modelo PIM donde se puede apreciar algunos de los componentes implementados en esta versión. Este diagrama describe mediante un ejemplo sencillo el proceso de diagramación del diagrama de clases.

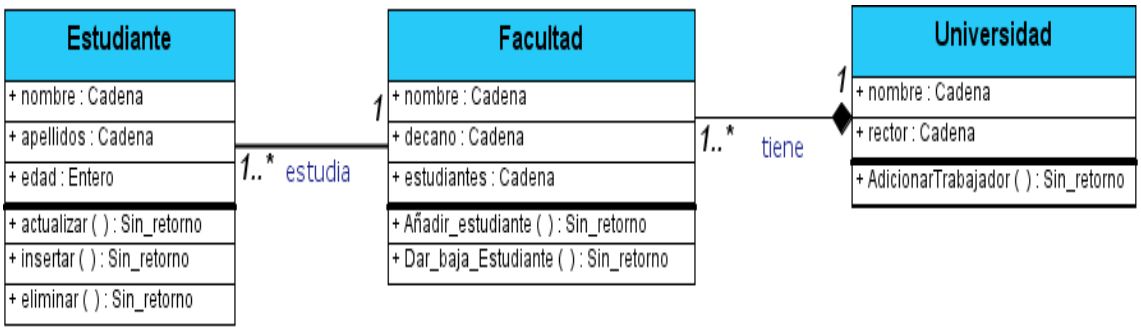


Figura 24: Diagrama de clases modelado en la herramienta prototípica JMDA

Fuente: Elaboración propia

Diagrama de clases modelo PSM

En la Figura 25 se muestra el diagrama obtenido al realizar una transformación al diagrama de la Figura 24, como se puede ver se aplica el patrón de diseño MVC y se crean los métodos *set* y *get* de cada uno de los atributos de las clases del modelo PIM. Las operaciones presentes en cada una de estas clases son transferidas a su controlador correspondiente y son declarados los componentes básicos que contendrá la vista de cada una de las clases.

Como esta transformación se realiza específicamente para la plataforma java, son cambiados los tipos genéricos que tenían los atributos y operaciones en el modelo PIM, por sus correspondientes en el la plataforma java en el modelo PSM.

Es importante destacar que para los atributos que desde el modelo PIM tiene un valor definido, no se crean componentes en la vista asociada a la clase que lo contiene.

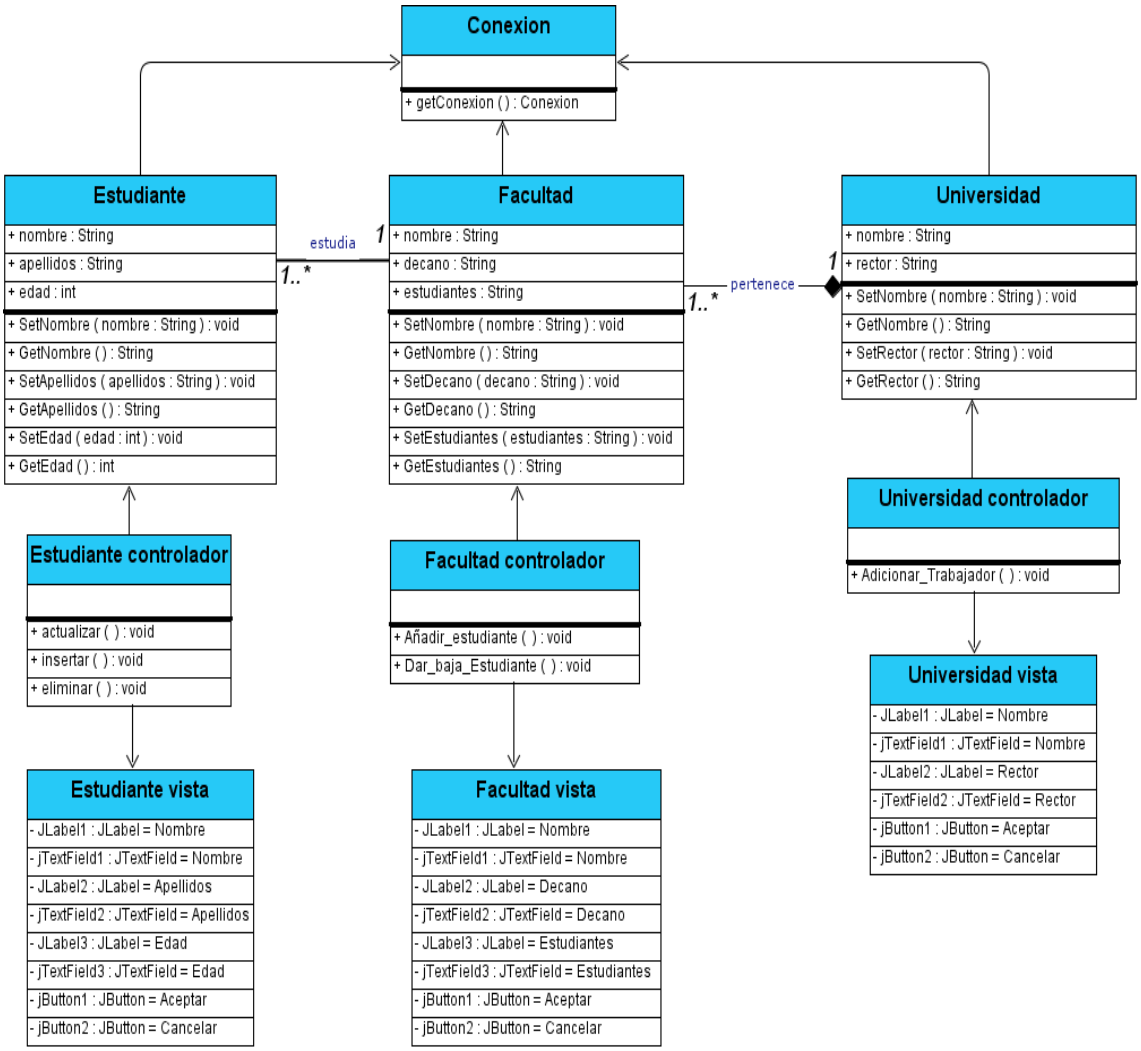


Figura 25: Diagrama de clases transformado en la herramienta prototípica JMDA Fuente: Elaboración propia

En los anexos aparecen los diagramas de actividades creados para cada una de las clases que tienen los métodos actualizar, insertar o eliminar un registro en la base de datos.

4.3 Conclusiones del capítulo

1. Se realizó un estudio de factibilidad donde se estimaron los principales costos asociados al desarrollo de la versión 4.0 de la herramienta jMDA.
2. Se diseñaron tres escenarios de prueba para la validación funcional de la herramienta; identificándose nueve casos de prueba específicos.
3. Las pruebas realizadas al *software* demostraron en un 100 % el correcto funcionamiento de las funcionalidades implementadas en la aplicación.
4. Se desarrolló un ejemplo de uso de la herramienta, comprobando su aplicabilidad.

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 y que las versiones anteriores de la herramienta *CASE* JMDA presentan limitantes en las teorías de diseño e implementación utilizadas.
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* siguiendo las tendencias en la bibliografía y se modeló el algoritmo de transformación empleando la notación BPMN.
3. Se diseñó el diagrama de paquetes, los diagramas de casos de uso significativos fueron ilustrados mediante diagramas de secuencia y fue descrito como se llevó a cabo el tratamiento de errores en la herramienta *CASE* JMDA.
4. Se realizó un estudio de factibilidad donde se estimaron los principales costos asociados al proyecto; se diseñaron y aplicaron tres escenarios de prueba con sus respectivos casos; las cuales demostraron en un 100 % el correcto funcionamiento de las funcionalidades implementadas en la aplicación.
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 4.0 de la herramienta jMDA en la docencia de pregrado y postgrado de la UCLV.
2. Divulgar la herramienta obtenida en las empresas de desarrollo del software cubanas para que sea utilizada o probada.
3. Definir un estándar *XMI* que permita la estandarización de los diagramas modelados para su intercambio.
4. Diseñar la arquitectura general de la herramienta *CASE jMDA* que permita obtener una vista macro de los componentes que se han obtenido y que faltan por obtener.
5. Continuar el perfeccionamiento del módulo obtenido en próximas versiones que permita:
    a. Implementar más reglas de transformación del propio diagrama de clases y de los otros que se pueden modelar en el PIM para la propia plataforma Java.
    b. Implementar otras plataformas como C# y Phyton.
    c. Mejorar la trazabilidad implementada entre los diagramas del modelo *PIM* y el modelo *PSM, permitiendo lograr la reconstrucción*.

REFERENCIAS BIBLIOGRÁFICAS

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).

Pereira, C. T. (2008). Formalización de Refractorings en el contexto de MDA.

Pérez Armayor, D. (2014). *Technology combinations decision model for supply chains information systems integration*. Oldenburg, Germany: Shaker Verlag.

Pérez Armayor, D., Abreu Fong, P. A., Hernández Lantigua, D., León Alen, E. O., & Díaz Batista, J. A. (2016). El empleo de Temix para evaluación de tecnologías de la información en planes de Sistemas de Información. Retrieved from

Pérez, D. (2015). Modelo de Referencia de Escenarios de Gestión y Tecnologías de la Información aplicado en un Grupo Farmacéutico.

Plante, F. (2006). Introducing the GMF Rumtime.

Pons, C. (2010). Desarrollo de software dirigido por modelos.

Pressman, R. S. (2001). Ingeniería de Software. Un enfoque práctico.*, 5*.

Quintero, R. (2003). Aplicación de MDA al desarrollo de aplicaciones web en OOWS.

Sáez, P. A. F. (2009). *Un Análisis Crítico Sobre la Aproximación Model Driven Architecture* Universidad Complutense de Madrid, Madrid.

Sommerville, I. (2002). Ingeniería de Software 6ta ed.

T. Gardner, L. Y. (2006). A closer look at model-driven development and other industry initiatives,. Retrieved from

UML-OMG. (2012). OMG Unified Modeling Language.

Vara, J., de Castro, V., Cáceres, P., & Marcos, E. (2017). Arquitectura de MIDAS-CASE: una herramienta para el desarrollo de SIW basada en MDA.

ANEXOS

Diagrama de actividades modelo PSM

Paralelamente a la transformación de un diagrama de clases se crean diagramas de actividades que describen las operaciones básicas que se realizan sobre una determinada clase. Lo descrito anteriormente sucede cuando en una clase en el modelo PIM está presente la declaración de las operaciones actualizar, eliminar o insertar.

En las siguientes figuras se muestran los diagramas de actividades asociados a la clase "Estudiante" descrita en la figura 23.

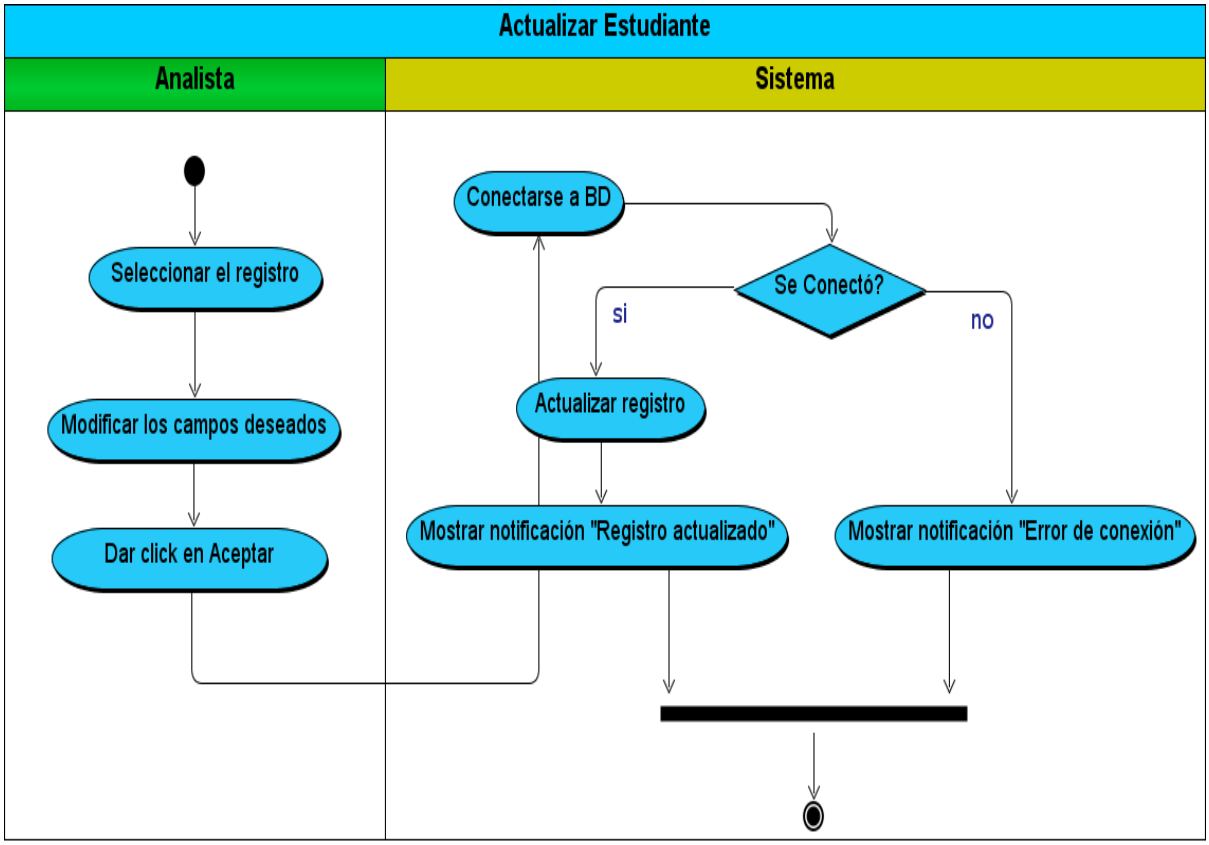


Figura 26: Diagrama de actividades asociado a la clase Estudiante que describe la operación actualizar estudiante.

Fuente: Elaboración propia

La figura 25 muestra el diagrama de actividades generado automáticamente por el sistema que describe el proceso de actualización de los datos de un estudiante.

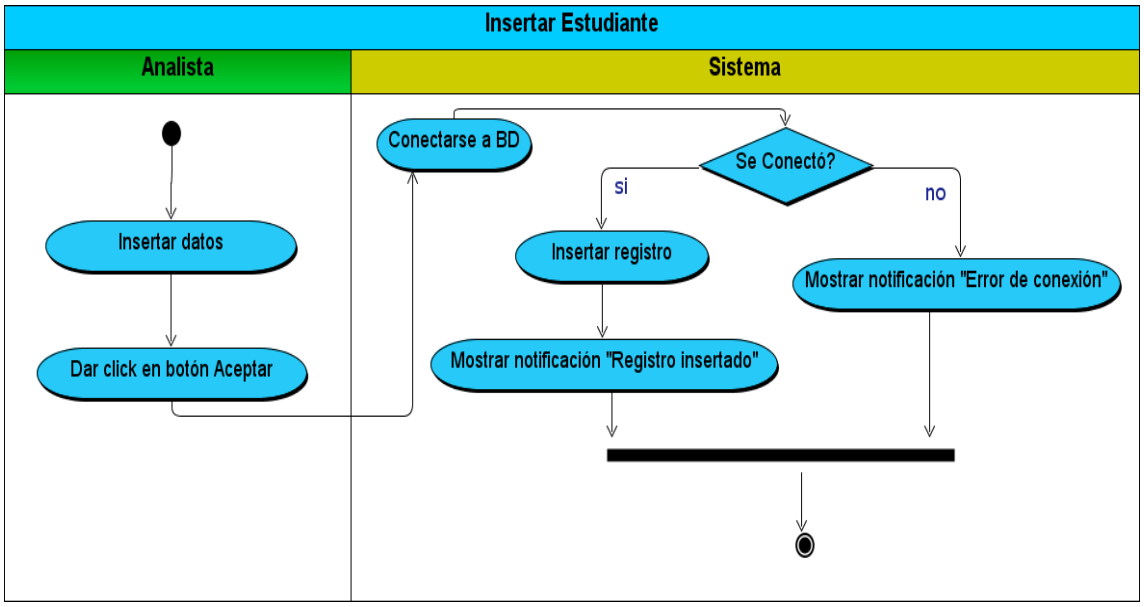


Figura 27: Diagrama de actividades asociado a la clase Estudiante que describe la operación insertar estudiante.

Fuente: Elaboración propia

La figura 26 muestra el diagrama de actividades generado automáticamente por el sistema que describe el proceso de eliminación de los datos de un estudiante.

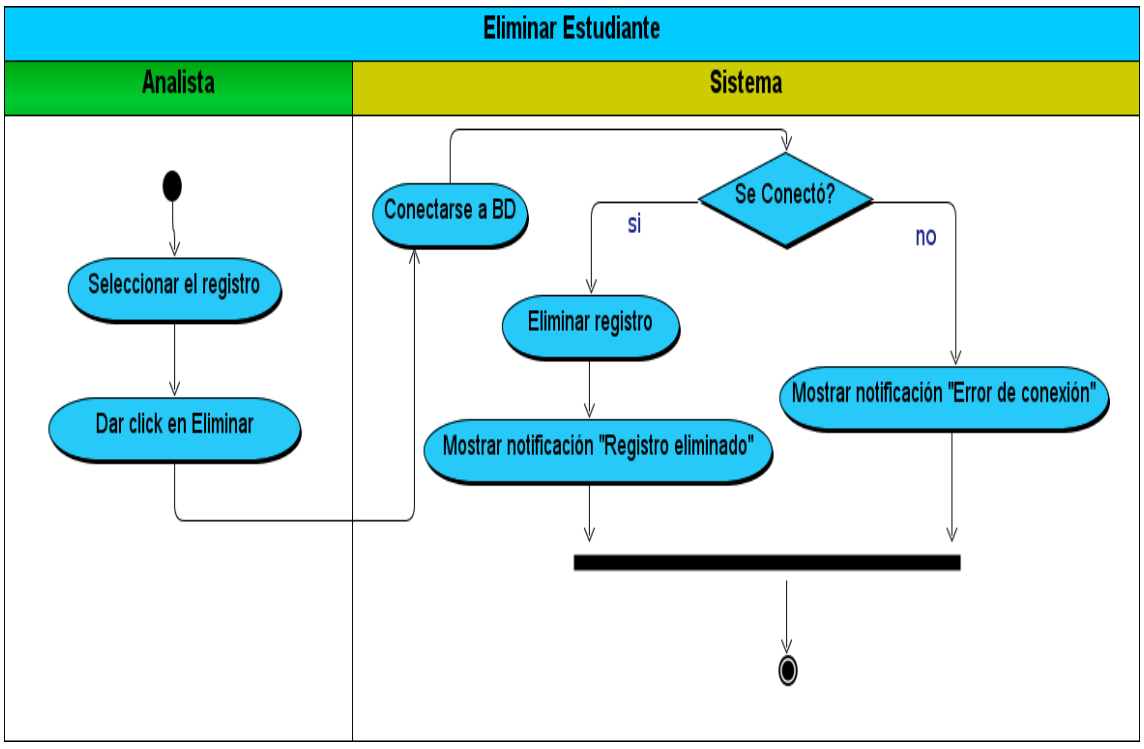


Figura 28: Diagrama de actividades asociado a la clase Estudiante que describe la operación insertar estudiante.

Fuente: Elaboración propia

La figura 27 muestra el diagrama de actividades generado automáticamente por el sistema que describe el proceso de inserción de los datos de un estudiante.