II CONVENCIÓN CIENTÍFICA INTERNACIONAL
"II CCI UCLV 2019"

DEL 23 AL 30 DE JUNIO DEL 2019.
CAYOS DE VILLA CLARA. CUBA.

II CONFERENCIA INTERNACIONAL DE PROCESAMIENTO DE LA INFORMACIÓN (CIPI 2019)

Módulo PIM-PSM versión 5.0 de la herramienta jMDA

Module PIM-PSM version 5.0 for jMDA tool

Dr. Rosendo Moreno Rodríguez[1], Ing. Liosbel Díaz Lorenzo [2], Ing. Patricia Airela Abreu Fong[3], Ing. Eduardo Ariel Orozco Álvarez[4]

1- Dr. Rosendo Moreno Rodríguez. Universidad Central "Marta Abreu" de Las Villas, Cuba. E-mail: rosendo@uclv.edu.cu

2- Ing. Liosbel Díaz Lorenzo. UCLV, Cuba. E-mail: ldlorenzo@uclv.cu

3- Ing. Patricia Airela Abreu Fong. UTHJAE. E-mail: paabreu@ind.cujae.edu.cu

4- Ing. Eduardo Ariel Orozco Álvarez. UCLV, Cuba. E-mail: eorozco@uclv.cu

Resumen:

El desarrollo de un software es un proceso complejo y difícil de gestionar en el que intervienen múltiples elementos. 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)*, ha prestado 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, los modelos guían todo el proceso. Varias herramientas se han desarrollado a partir de esta idea internacionalmente, pero muchas son de autor o privativas, y además no cubren el modelo completo.

En la UCLV se está desarrollando una herramienta jMDA que pretende cumplir con las tres transformaciones establecidas teóricamente, entre las cuatro fases. En este trabajo se

acomete la segunda etapa entre las fases del Modelo Independiente de Plataforma (PIM y Modelo Específico de Plataforma (PSM)

La versión 5.0 del módulo PIM-PSM de la herramienta "jMDA", constituye un perfeccionamiento de las anterior, ampliando 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.

Abstract:

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

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

Version 5.0 of the PIM-PSM module of the "jMDA" tool, is a refinement of the previous ones, focused on the implementation of the diagrams of use cases, activities, status and sequence at the level of "Platform Independent Model" (PIM); as well as the computer-assisted transformation of these diagrams, and the creation of artifact diagrams and deployment to the "Platform Specific Model" (PSM). The development of the research

does not focused only on the implementation of the solution, but also in the generation of documentation of the software development methodology used, as well as the use of the tool and the feasibility analysis associated with the development of the proposed solution.

Palabras Clave: Ingeniería del software; Arquitectura dirigida por modelos
Keywords: *Software engineering; Model driven architecture.*

1. Introducción

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.

El *Object Management Group (OMG)* en búsqueda de una alternativa para resolver estos problemas en el año 2001 establece el *framework* Arquitectura Dirigida por Modelos *(MDA),* 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, permitiendo que se obtengan beneficios importantes en aspectos fundamentales como son la productividad, la portabilidad, la interoperabilidad y el mantenimiento

A pesar de las ventajas de esta nueva filosofía de trabajo resulta imprescindible la existencia de herramientas que le den soporte. En este contexto el Centro de Investigaciones de Informática de la UCLV desde el año 2010 se propone desarrollar una herramienta que implemente el paradigma *MDA* por sus tres módulos de conversión de manera independiente. Este trabajo específicamente se refiere al módulo correspondiente a la transformación de diagramas UML en el Modelo Independiente de la Plataforma hacia el Modelo Específico de la Plataforma solo orientado a la programación Java.

La industria de software nacional contaría entonces con una herramienta profesional que supere las limitantes con que cuenta las existentes en la actualidad, ya sea el pago de una licencia en el caso de las propietarias o limitantes de orden técnico en las de código abierto. Por lo que significaría un aporte invaluable al proceso de desarrollo de software en Cuba la implementación de una herramienta con estas características.

2. Metodología

El presente trabajo es una investigación aplicada. Se usaron métodos y técnicas de la Ingeniería del Software como el modelado UML y la programación en Java, para el desarrollo de la herramienta.

3. Resultados y discusión

3.1. 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 1 se presenta el proceso de desarrollo de software siguiendo el paradigma MDA.

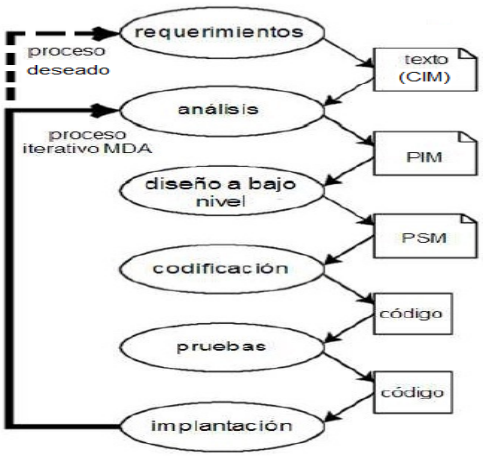

Figura 1: Etapas del desarrollo de software con MDA. Fuente: Tomado de (Pons, 2010)

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** (CIM): Surge en la fase inicial del proceso de desarrollo comprendiendo la modelación del negocio en su totalidad; donde se modelan los requisitos del sistema, se describe la situación en la que el sistema será utilizado y sirve de enlace entre los expertos en el dominio del problema y sus requisitos con los expertos en el diseño y construcción de software. Los requisitos pueden ser representados mediante diagramas de caso de uso, de actividad y de secuencia. Entre los tipos de requisitos más comunes obtenidos en CIM se encuentran los requisitos de usuario, los funcionales, los no funcionales y los organizacionales.
2. **Modelo Independiente de la Plataforma (PIM)**: Representa los modelos que describen una solución de software que no contiene detalles de la plataforma concreta donde será implementada la solución, de ahí su nombre de modelos independientes

de la plataforma. Estos modelos surgen como resultado del análisis y diseño. Tal modelo tiene cierto nivel de abstracción que no cambia; sin importar la plataforma que sea elegida para su implementación.

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

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

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

3.2. Herramientas de implementación.

Para el desarrollo de la versión 5.0 del módulo PIM-PSM de la herramienta jMDA se utilizó el entorno de desarrollo integrado libre NetBeans, hecho principalmente para el lenguaje de programación Java, que fue el utilizado en la implementación. 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, así como la biblioteca MxGraph, sobre la cual se desarrolló el ambiente gráfico que permite tanto crear, como modificar continuamente los diagramas UML tanto en el modelo PIM como en el PSM que se obtiene automáticamente. Además con esa biblioteca se logró la importante opción de guardar en un formato específico que posibilita la conectividad de este módulo con los otros dos desarrollados (módulo CIM-PIM y módulo PSM-Código).

3.3. Algoritmo de transformación.

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

Reglas mantenidas de la versión anterior.

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

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

Ejemplo:

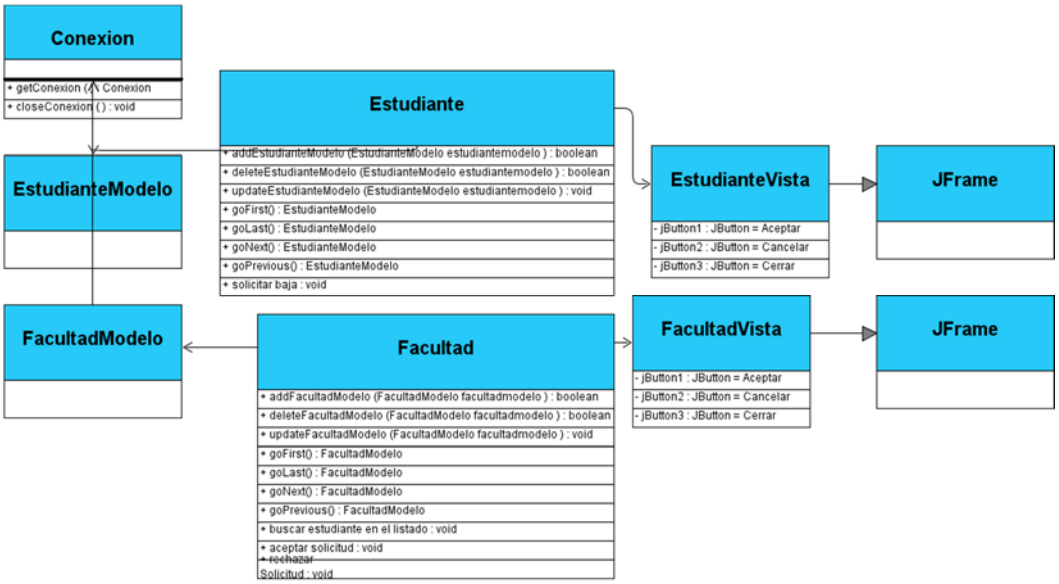

Figura2: Diagrama de clases en el modelo PSM

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

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

Si una clases del modelo *PIM* no contiene atributos no tiene sentido aplicar el patrón de diseño antes mencionado ya que no sería necesario tener ninguna vista para manejar la

información asociada a ella, es por eso que se decide aplicar este patrón a las clases que contengan atributos.

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

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

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

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

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

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

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

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

Reglas elaboradas para la nueva versión

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

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

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

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

Para realizar las trasformaciones de los diagrama de clases del modelo *PIM* al modelo *PSM* se diseña un algoritmo con alto grado de complejidad que se muestra en la figura que sigue.

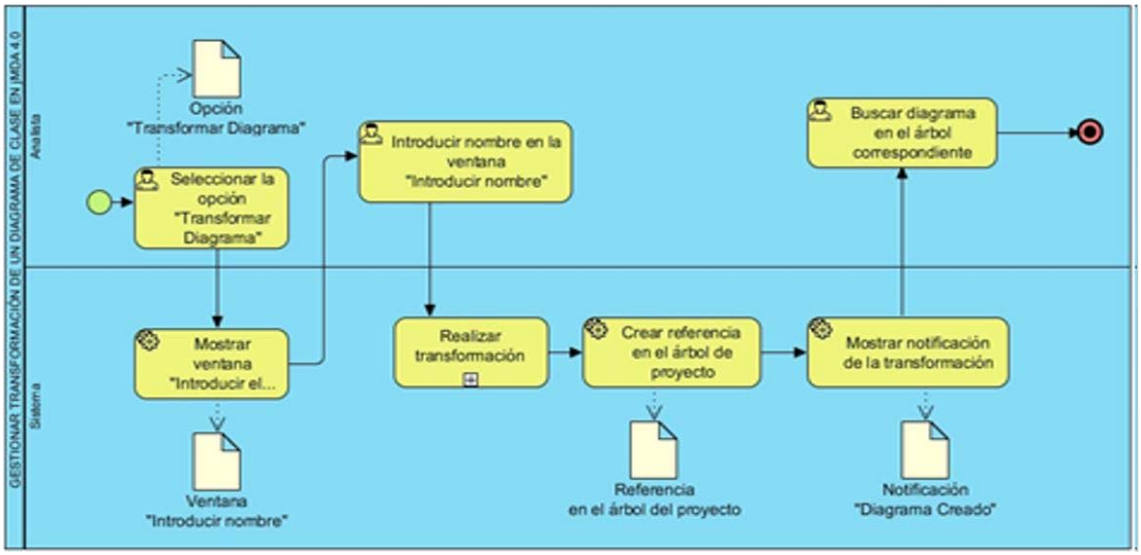

Figura3: Algoritmo de transformación de diagramas de clases utilizando la notación BPMN.

Como se puede apreciar en la Figura 3 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.

3.4. Breve descripción de la herramienta.

El módulo PIM-PSM en su versión 5.0, al iniciarse tiene esta vista inicial:

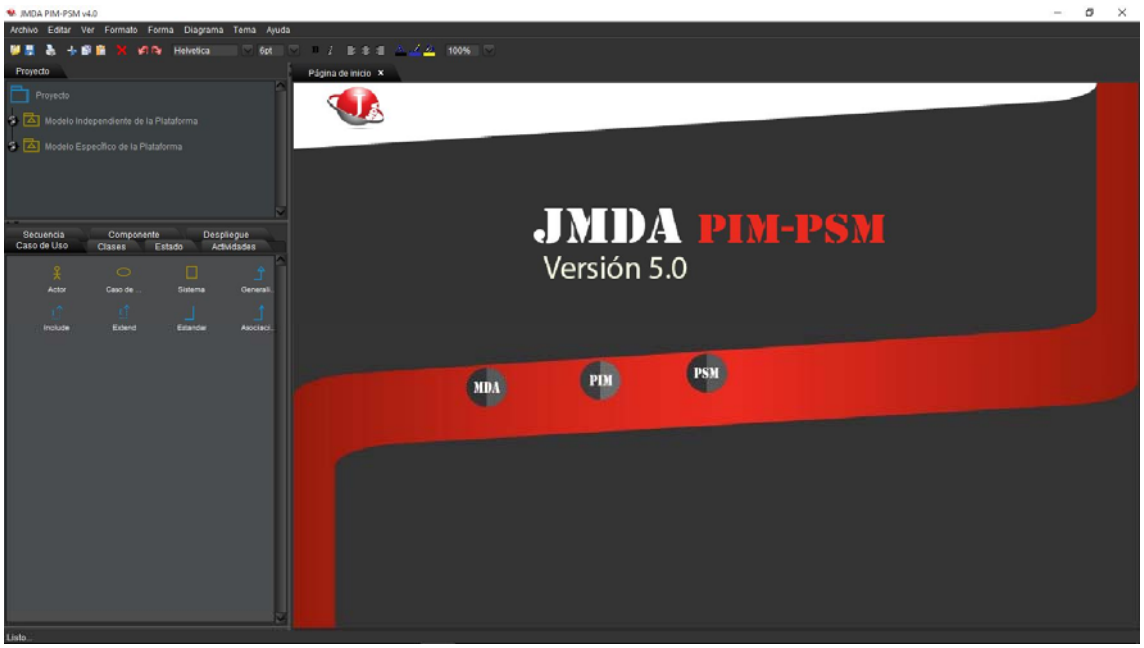

Figura 4. Imagen de inicio de la herramienta.

Como se puede apreciar se presenta un menú principal con las opciones Archivo, Editar, Ver, Formato, Forma, Diagrama, y Lenguaje). De acuerdo a la opción seleccionada aparecerán otras opciones correspondientes. Debajo en la parte izquierda está el panel de los Diagramas UML contemplados en esta versión (Casos de Uso, Actividades, Clases, Estados y Secuencias) para el PIM. Cuando se selecciona un diagrama entonces se amplía el panel con los componentes correspondientes.

El área de trabajo de la aplicación donde se puede destacar tres áreas principales que serían el árbol del proyecto que es donde se crean los nuevos diagramas y se pueden abrir otros que se hayan creado también está la paleta de componentes que son todos los componentes a utilizar para la elaboración de los diagramas, y la otra área importante es el área de trabajo donde se puede trabajar con los diagramas que estén en elaboración. A continuación, en la figura 5, se muestran varias áreas importantes de la aplicación.

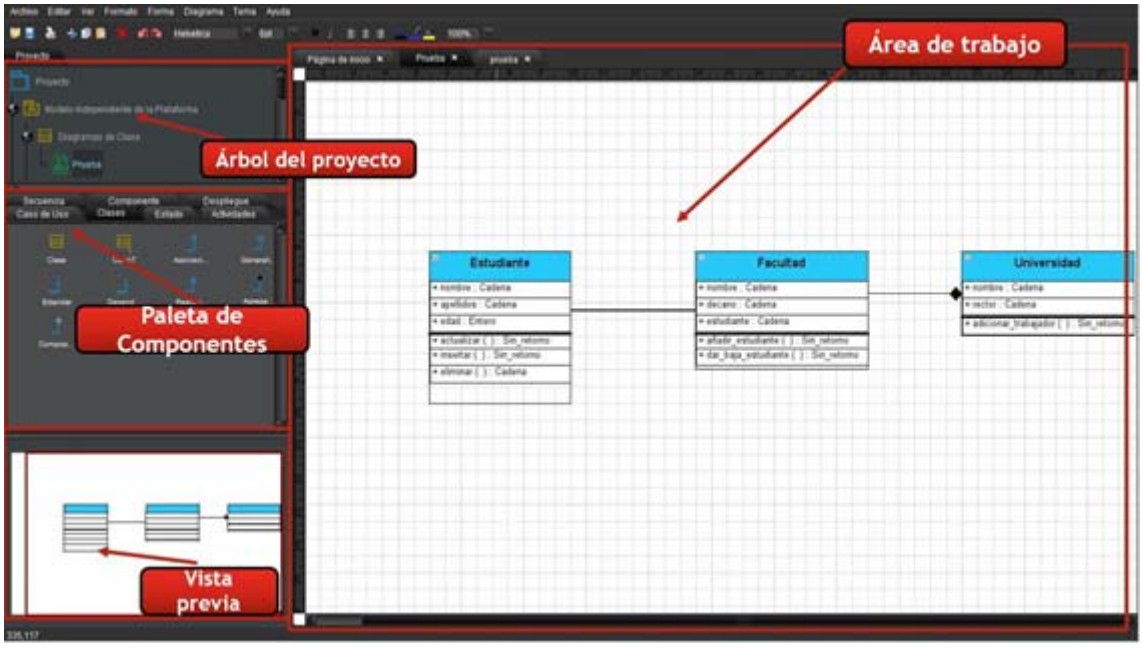

Figura 5. Ejemplo de inicio de diagrama de clases en la herramienta destacando áreas.

3.5. Comprobación de aplicabilidad.

Para comprobar la aplicabilidad de este módulo PIM-PSM en su versión 5 se desarrolló un ejemplo: 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.

Diagrama de Clases en PIM

En la Figura 6 se muestra un diagrama de clases reducido de un sistema para controlar la información asociada a los estudiantes, profesores y facultades de una universidad.

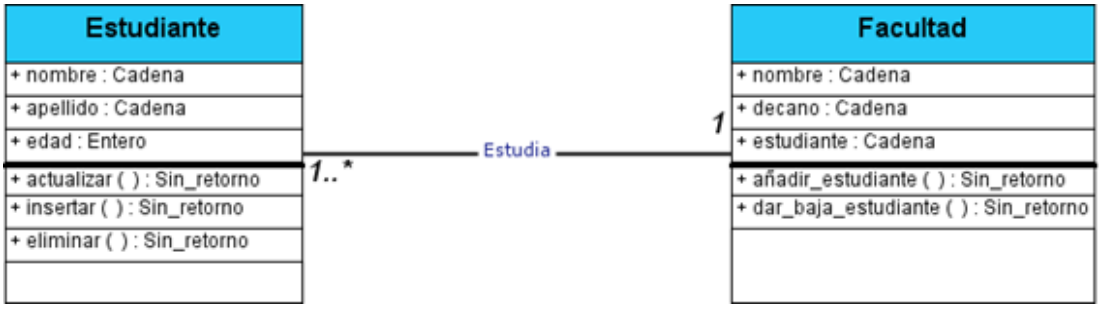

Figura 6: Diagrama de clases modelado en la herramienta prototípica jMDA. Nota: para mayor comprensión y nitidez solo se presenta el contenido del área de trabajo.

Diagrama de clases modelo PSM

En la Figura 7 se muestra el diagrama obtenido al realizar una transformación al diagrama de la Figura 6, 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 de 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.

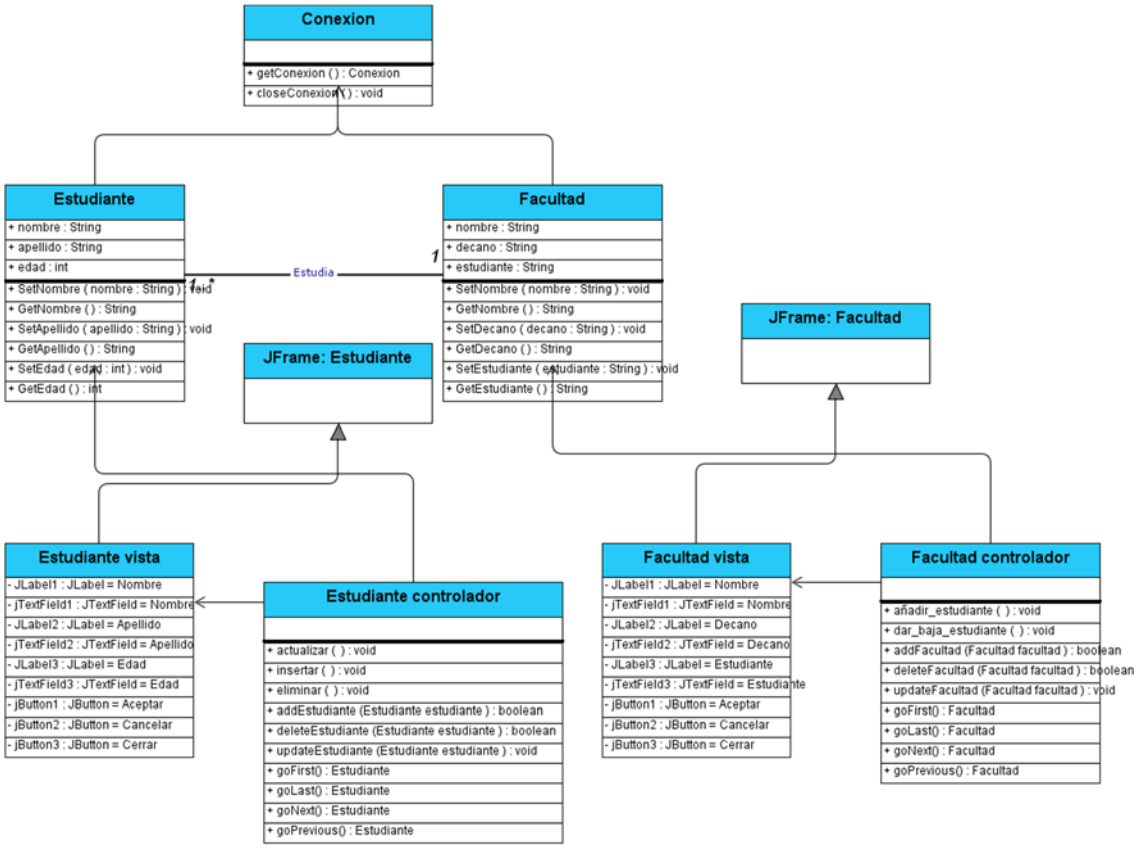

Figura 7: Diagrama de clases transformado en la herramienta prototípica jMDA.

4. Conclusiones

1. En base de un análisis crítico de la versión 4.0 se pudo notar donde se encontraban los errores más destacables y se desarrolló la versión 5, lo que incluye varios diagramas de UML en análisis y diseño y su implementación en Java utilizando la librería mxgraph con el NetBeans IDE 8.0.1 y el JDK 8.
2. 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, lo que permite validar su utilidad y corrección.

5. Referencias bibliográficas

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

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.].

Calisoft. (2014). Libro de diagnóstico.

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.

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

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