**SIMPOSIO ¨TRANSFERENCIA DE CONOCIMIENTO EN TECNOLOGÍAS DE LA INFORMACIÓN¨**

**Módulo PSM-CODIGO versión 4.0 de la herramienta JMDA**

***PSM-CODE module version 4.0 of the JMDA tool***

**Dr. Rosendo Moreno Rodríguez1, Ing. David Ernesto Becerra Cabera 2**

1-Universidad Central “Marta Abreu” de las Villas, Cuba. E-mail: rosendo@uclv.edu.cu

2- Universidad Central “Marta Abreu” de las Villas, Cuba

**Resumen:**

La Industria Nacional del Software necesita de herramientas que permitan incrementar la productividad, portabilidad, mantenibilidad y calidad en la producción de este esencial producto en el mundo actual. Estas herramientas se conocen como Ingeniería del Software Asistida por Computadoras (CASE) y existen tanto propietarias como de código abierto. Pero lo mejor fuese tenerlas propias, para que incidan en el desarrollo de Cuba.

El Grupo de Dirección de la Orientación a Objetos (OMG) estandarizó el marco de trabajo Arquitectura Dirigida por Modelos (MDA) que fundamenta la existencia de cuatro etapas en el desarrollo del software basadas en el modelado del análisis y diseño fundamentalmente con Lenguaje Unificado de Modelado (UML), así como las transformaciones automáticas entre cada una y la siguiente. Las herramientas consideradas de este tipo tampoco son soberanas, y ni siquiera completas.

En el Centro de Investigaciones en Informática de la UCLV se fundamentó un Proyecto de Investigación que tenía como tarea esencial crear una herramienta propia que implementara dicha arquitectura (jMDA) dividida en las 3 transformaciones esenciales. El presente trabajo se corresponde a la cuarta versión de la transformación del Modelo Específico de Plataforma (PSM) a código fuente. En esta versión se hizo énfasis en tener un ambiente gráfico adecuado para crear o importar diagramas UML de los modelos específicos de la plataforma, y establecer principios adecuados de generación del código fuente que en este caso incluye varios lenguajes de programación destino: Java, C# y Phyton, así como la conversión a un SQL estándar de para clases tipo tablas de una Base de Datos.

***Abstract:***

*The National Software Industry needs tools to increase productivity, portability, maintainability and quality in the production of this essential product in today's world. These tools are known as Computer Assisted Software Engineering (CASE) and exist both proprietary and open source. But the best is to have their own, so that they influence the development of Cuba.*

*The Object Orientation Management Group (OMG) has standardized the Model-Driven Architecture (MDA) framework that supports the existence of four stages in software development based on modeling analysis and design primarily with Unified Modeling Language, like the automatic transformations between each one and the follows. The tools considered of this type are also not sovereign, nor even complete.*

*In the Research Center of Computer Science of the UCLV was based an Investigation Project that had as essential task to create an own tool that would implement said architecture (jMDA) divided in the 3 essential transformations. The present work corresponds to the fourth version of the transformation of the Specific Model of Platform to a source code. In this version, emphasis was placed on having a suitable graphical environment for create or import UML diagrams of the platform-specific models, and establish proper principles of source code generation, which in this case includes several target programming languages: Java, C # and Python, as well as the conversion to a standard SQL for table-type classes of a Database.*

**Palabras Clave:** Ingeniería del Software; Arquitectura Dirigida por Modelos.

***Keywords:*** Software Engineering; Model Driven Architecture.

**1. Introducción**

El modelado es una parte central de todas las actividades que conducen a la producción de buen software, ya que a través del modelado se consigue visualizar, especificar (estructura y comportamiento de un artefacto), proporcionar plantillas que sean una guía en la construcción del sistema y documentación de las decisiones adoptadas por los desarrolladores. El UML (“Unified Modeling Language”, Lenguaje Unificado de Modelado) es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad (Cueva, 1999); está respaldado por el OMG (“Object Management Group”). Los modelos proveen abstracciones de un sistema físico que permiten a los ingenieros analizar el sistema ignorando detalles complejos mientras se enfocan en las partes más relevantes, como puede ser la lógica del negocio. MDA es una arquitectura que proporciona un conjunto de guías para estructurar especificaciones expresadas como modelos.

Uno de los principales objetivos de MDA es separar el diseño de la arquitectura y de las tecnologías de construcción, facilitando que el diseño y la arquitectura puedan ser alterados independientemente. El diseño alberga los requerimientos funcionales (casos de uso) mientras que la arquitectura proporciona la infraestructura a través de la cual se hacen efectivos requerimientos no funcionales como la escalabilidad, fiabilidad o rendimiento. Usando la metodología MDA, la funcionalidad del sistema será definida en primer lugar como un modelo independiente de la plataforma (Platform-Independent Model o PIM) a través de un lenguaje específico para el dominio del que se trate. Dado un modelo de definición de la plataforma (Platform Definition Model o PDM) correspondiente a CORBA, .NET, web, etcétera, el modelo PIM puede traducirse entonces a uno o más modelos específicos de la plataforma (Platform-Specific Models o PSM) para la implementación correspondiente, usando diferentes lenguajes específicos del dominio, o lenguajes de propósito general como Java, C#, Python, etc.

La última de las transformaciones incluidas en este marco de trabajo es la conversión de los diagramas UML logrados en PSM al lenguaje de programación previamente determinado como definitivo para el software en construcción. Esta es la parte o módulo que se logró extender con este trabajo. El mismo es parte integrante de una herramienta más amplia que incluye los módulos CIM (Modelo Independiente de la Computación)-PIM y el PIM-PSM, que se han desarrollado antes como parte de la tarea de investigación.

El objetivo de este trabajo fue Implementar la versión 4.0 del módulo PSM-Código de la herramienta JMDA, mejorando los servicios que brinda las versiones anteriores y haciendo énfasis en las prestaciones gráficas adecuadas, así como la lógica de la transformación a código fuente en los lenguajes Java, C#, Python y SQL.

**2. Metodología**

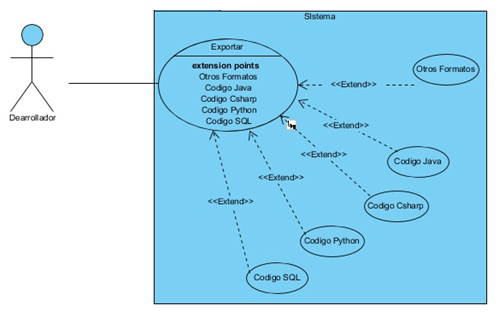
El presente trabajo es una investigación aplicada. Se usaron métodos y técnicas de la Ingeniería del Software como el modelado UML y la programación en Java, para el desarrollo de la herramienta.

**3. Resultados y discusión**

**3.1. Modelado UML.**

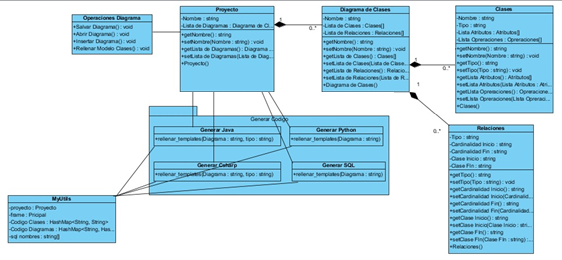
Se desarrolló el modelado, usando específicamente los diagramas de Casos de Uso, Clases y Actividades fundamentales. Solo se mostrarán los 2 primeros.

En este software se define a un único Actor, al Desarrollador del software el cual tiene la tarea de diseñar e implementar las clases necesarias para la generación automática en cada lenguaje.



**Figura 1 Casos de Uso del Módulo PSM-Código**

En la siguiente imagen se muestran las clases más importantes de la herramienta implementada, estas permiten la creación, manejo y almacenamiento de los PSM.



**Figura 2 Diagrama de Clases**

**3.2. Herramientas de implementación.**

Para el desarrollo de la versión 4.0 de la herramienta JMDA PSM-Código se utilizó el entorno de desarrollo integrado libre NetBeans, hecho principalmente para el lenguaje de programación java, que fue el utilizado en la implementación. Se utilizaron las bibliotecas JGraphX, JDK 8u91 y ANTLR.

**3.3. Algoritmo de transformación.**

A los Modelos del Sistema de las fases de definición y análisis deben aplicárseles una serie de reglas de transformación para incorporarle los elementos propios de la plataforma específica.

A las reglas ya implementadas en la versión anterior del software se le añadieron las diferentes implementaciones de cada regla para la plataforma especifica.

* R1: Todo componente de un Diagrama de Clase de tipo mxConstants.CLASE y que no sea Abstracta será convertido en una clase.

public class nombreclase {} ---- para los lenguajes Java y Csharp

class nombreclase(Object): ---- para el lenguaje Python

* R2: Todo componente de un Diagrama de Clase de tipo mxConstants.CLASE y sea Abstracta será convertido en una clase.

public abstract class nombreclase {} ---- para los lenguajes Java y Csharp

from abc import ABCMeta, abstractmethod

class nombreclase(Object): ---- para el lenguaje Python

\_metaclass\_=ABCMeta\_

* R3: Todo componente de un Diagrama de Clase de tipo mxConstants.INTERFAZ será convertido en una interfaz.

public interface nombreinterfaz{} ---- para los lenguajes Java y Csharp

from abc import ABCMeta, abstractmethod

class nombreclase(Object): ---- para el lenguaje Python

“””Interface”””

\_metaclass\_=ABCMeta\_

* R4: Todo componente de un Diagrama de Clase de tipo mxConstants.ATRIBUTO será convertido en un atributo de la clase en la que se encuentra contenido.

<visibilidad> <tipo de dato> nombreatrib <value> ---- para los lenguajes Java y Csharp

self.<visibilidad><nombre> = <value> ---- para el lenguaje Python

"""@AttributeType <tipo>"""

* R5: Todo componente de un Diagrama de Clase de tipo mxConstants.OPERACION y que no sea Abstracta será convertido en una operación.

<visibilidad> <tipo de retorno> nombreoper(<parámetros>){} ---- para los lenguajes Java y Csharp

def <visibilidad> nombreoper(<parámetros>): ---- para el lenguaje Python

* R6: Todo componente de un Diagrama de Clase de tipo mxConstants.OPERACION y que sea Abstracta será convertido en una operación de la clase en la que se encuentra contenido.

public abstract <tipo de dato> nombreoper(<parámetros>); --- para el lenguaje Java

<visibilidad> abstract <tipo de dato> nombreoper(<parámetros>);

--- para el lenguaje Csharp

@abstractmethod

def <visibilidad> nombreoper(<parámetros>): ---- para el lenguaje Python

* R7: Todo componente de un Diagrama de Clase de tipo mxConstants.ASOCIACION, mxConstants.AGREGACION y mxConstants.COMPOSICION serán convertidos en un atributo de tipo Clase origen en la Clase destino, y viceversa, de la forma.

<visibilidad> <objetoClaseDestino> nombreObjeto; >){} ---- para los lenguajes Java y Csharp

nombreObjeto = []

# @AssociationType <objetoClaseDestino>[] ---- para el lenguaje Python

# @AssociationMultiplicity

* R8: Todo componente de un Diagrama de Clase de tipo mxConstants.GENERALIZACION convertirá sus dos componentes asociados de tipo mxConstants.CLASE en una relación de generalización.

public class nombreclase extends nombreClasePadre{} --- para el lenguaje Java

public class nombreclase: <nombreClasePadre1>,………., <nombreClasePadreN> --- para el lenguaje Csharp

import <nombreClasePadre1>

import <nombreClasePadreN>

class nombreclase(<nombreClasePadre1>,………., <nombreClasePadreN>): ---- para el lenguaje Python

Para los lenguajes Csharp y Python se implementó las características de herencia múltiple que ambos soportan.

* R9: Todo componente de un Diagrama de Clase de tipo mxConstants. IMPLEMENT implicara importar en el lenguaje la clase origen en la clase destino.

public class nombreclase implements nombreInterfaz{} --- para el lenguaje Java

public class nombreclase: <Interfaz1>,………., <InterfazN> --- para lenguaje Csharp

import < Interfaz1>

import < InterfazN>

class nombreclase(<Interfaz1>,………., < InterfazN>): ---- para el lenguaje Python

* R10: Todo componente del Diagrama de Clases de tipo mxCell.Steriotype si es Struct y asociado a una mxConstants.CLASE pasa a ser una Estructura del lenguaje Csharp.

public struct nombreStructura{}

* R11: Todo componente del Diagrama de Clases de tipo mxCell.Steriotype si es Enum y asociado a una mxConstants.CLASE pasa a ser un Enumeradodel lenguaje Csharp.

public enum nombreEnumerado{}

* R12: En el código generado en el lenguaje Csharp se le adicionará un namespace del mismo nombre del diagrama al que pertenece la clase.

namespace nombreDiagrama{}

* R13: Todo componente del Diagrama de Clases de tipo mxCell.Steriotype si es Table y asociado a una mxConstants.CLASE pasara a ser un script .sql . Cada atributo asociado, a ese componente pasa a ser una columna a insertar de dicha tabla.

CREATE TABLE nombreTabla

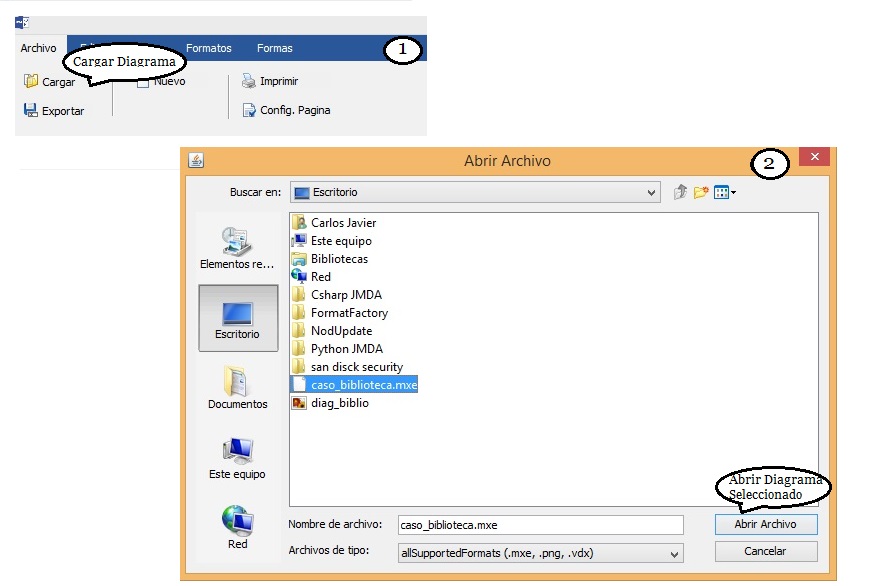
“nombreAtributo” <tipo>

**3.4. Breve descripción de la herramienta.**

La herramienta implementada tiene como objetivo la generación de código en Csharp, Python y SQL, además de optimizar la generación en código Java de la versión anterior de la herramienta a partir de un PSM, utilizando la tecnología Stringtemplate que contiene la biblioteca de java ANTLR. La misma muestra un ambiente de trabajo sencillo, lo cual facilita que el usuario tenga un buen desenvolvimiento en la ejecución del programa. Esto brinda la posibilidad de completar esta tarea en un corto período de tiempo.

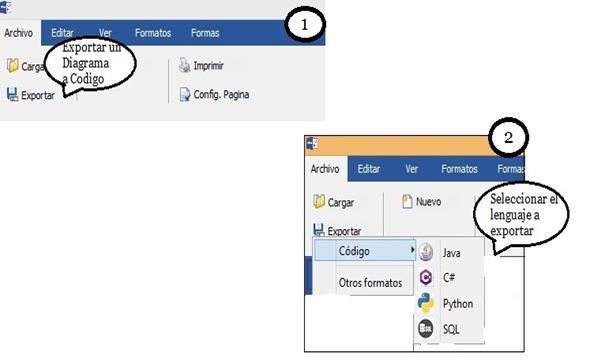
El módulo PSM-Código en su versión 4.0, al iniciarse tiene un menú con varias opciones (solo se presentan algunas):

* **Cargar**: Despliega una ventana para importar los Diagramas guardados anteriormente en los formatos XME, PNG o VDX.



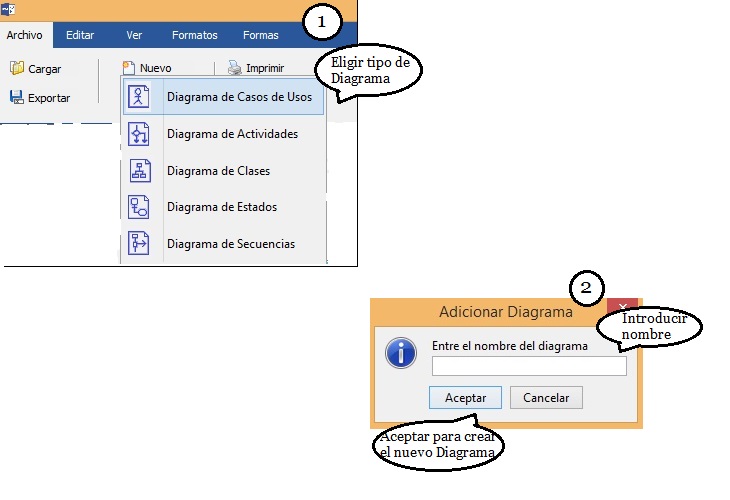
**Figura 3 Menú Cargar Diagrama**

* **Exportar:** Despliega un menú de dos opciones, la primera para exportar el código generado en los diferentes tipos de lenguaje de programación que ofrece la herramienta, esta característica solo está disponible para los Diagramas de Clase y la segunda opción permite exportar los diagramas en diferentes formatos MXE, PNG, SVG los cuales son los que la herramienta emplea para importar los diagramas guardados, además existen otros formatos como bmp, wbmp, html, gif, etc.



**Figura 3 Menú Exportar a Código**

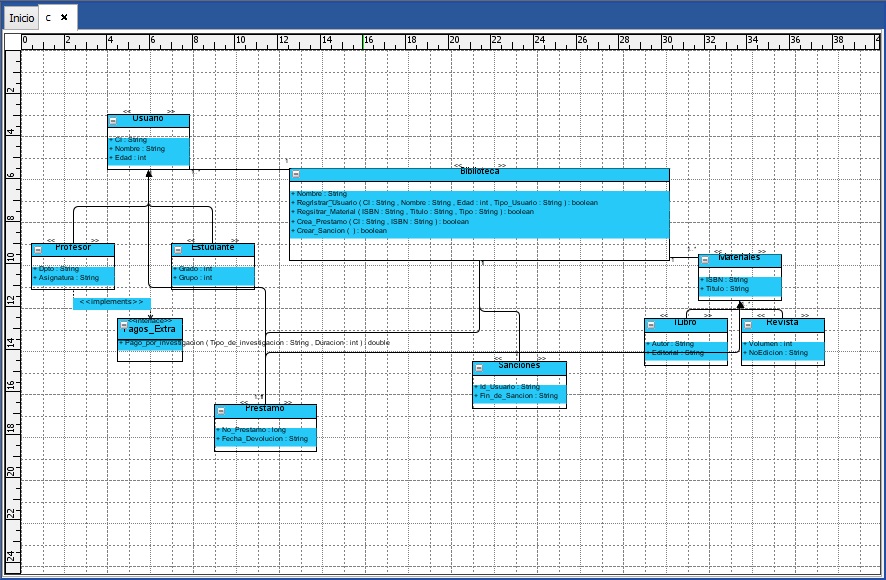
* **Nuevo:** Permite crear un nuevo diagrama según el tipo especificado.



**Figura 4 Menú Crear Diagrama**

* **Panel de Trabajo**

En este panel es donde el usuario realiza y diseña los diferentes tipos de diagrama.



**Figura 5 Panel de Trabajo**

**4. Conclusiones**

1. Se desarrolló un nuevo módulo gráfico a la herramienta que permite la edición directa o a través de la importación a un ambiente PSM, de los diagramas UML esenciales para la generación automática de código fuente, con las funcionalidades adecuadas y normales en este tipo de herramientas, al que se le incorporaron los formatos de salida (JPG, PNG, MXE) logrando exportar los distintos diagramas creados con una excelente calidad. Además, contiene de forma adicional otros formatos de salida propios de la biblioteca utilizada para la confección de los diagramas para su conveniente uso.
2. Se logró implementar el módulo PSM-Código versión 4 de la herramienta jMDA, que permite a los desarrolladores diseñar los diagramas de casos de uso, clases, estado, actividades y secuencia del UML en un Modelo Especifico de la Plataforma, también permite importar esos mismos diagramas desde el módulo PIM-PSM versión 5 (desarrollado en otro trabajo); así como generar el código fuente, en los lenguajes Java, C#, Phyton y SQL.

**5. Referencias bibliográficas**

1. Durán Muñoz, F., Troya Castilla, J., & Vallecillo Moreno, A. (2013). Desarrollo de software dirigido por modelos. Journal of Chemical Information and Modeling, 53(9), 1689–1699.
2. Giandini, R. (2007). Un Marco Formal para Transformaciones en la Ingeniería de Software Conducida por Modelos.
3. Pilone, B. D., & Pitman, N. (2005). UML 2.0 in a Nutshell. In Journal of Enhanced Heat Transfer (Vol. 12, Issue June). https://doi.org/10.1615/JEnhHeatTransf.v12.i4.80
4. Scholl, P. M., Reiterer, P. H., & Mansmann, A. S. (n.d.). Model-Driven Architecture ( MDA ) and Data Warehouse Design Eduard Schibrowski. Development.