Universidad de las Ciencias Informáticas

Facultad 6

Título: Extensión de la herramienta “Visual Paradigm for UML” para el soporte al Desarrollo Dirigido por Modelos con Ext JS.

Trabajo de Diploma para optar por el título de

Ingeniero en Ciencias Informáticas

Autores:

Yenisleydis Ledesma Rodríguez.

Yunier Boza Roget.

Tutores:

Ing. Armando Robert Lobo.

Ing. Sergio García De La Puente.

Ing. Marleysi López Duque.

La Habana, junio del 2011

¿Qué será nuestro país dentro de 40, 50, 100 años? Todos nosotros nos preocupamos por ese país y quisiéramos tener el privilegio de que todo lo que hagamos sea para ese futuro, y que esta Cuba de hoy, que nosotros decimos que es el prototipo de la que soñó Martí al morir, siga siendo (…) si es posible una Cuba mejor que la de hoy y más revolucionaria que hoy. Y eso va a depender de ***nosotros*** *(…)* ***de todos*** *los revolucionarios.*

Fidel Castro Ruz

24 de febrero de 1998

DECLARACIÓN DE AUTORÍA

Declaración de autoría: Declaramos que somos autores de la presente tesis y reconocemos, a la Universidad de las Ciencias Informáticas, sus derechos patrimoniales sobre la misma con carácter exclusivo.

Para que así conste firmamos la presente a los ____ días del mes de _____ del año _____.

Yunier Boza Roget.

_____

Firma del autor

Yenisleydis Ledesma Rodríguez.

_____

Firma del autor

Ing. Armando Robert Lobo.

_____

Tutor

Ing. Sergio García De La Puente

_____

Tutor

Ing. Marleysi López Duque.

_____

Tutor

Datos de contacto

AUTORES:

Yenisleydis Ledesma Rodríguez.

Universidad de las Ciencias Informáticas,

Ciudad de la Habana, Cuba

E-mail: yledesmar@estudiantes.uci.cu.

Yunier Boza Roget

Universidad de las Ciencias Informáticas,

Ciudad de la Habana, Cuba

E-mail: yboza@estudiantes.uci.cu.

TUTORES:

Ing. Armando Robert Lobo.

Universidad de las Ciencias Informáticas,

Ciudad de la Habana, Cuba

E-mail: arobert@uci.cu.

Ing. Sergio García De La Puente.

Universidad de las Ciencias Informáticas,

Ciudad de la Habana, Cuba

E-mail: sgarcia@uci.cu.

Ing. Marleysi López Duque.

Universidad de las Ciencias Informáticas,

Ciudad de la Habana, Cuba

E-mail: mduque@uci.cu.

AGRADECIMIENTOS

A mi mamita linda por ser mi esperanza, por estar ahí en los momentos que mas la necesite, por ser mi paño de lagrimas en esos momentos en que la vida te juega una mala pasada, por ser mi fuente de inspiración y a la que le debo todo lo que hoy, si dios lo permite, lograr este triunfo que hemos esperado por cinco largos años.

A mis abues queridos, por ser un ejemplo a seguir y por brindarme todo el apoyo que se le puede ofrecer a una persona, los amo y adoro con la vida y ojala poder tenerlos muchos años más.

A mi papá que aunque no estuvo en los mejores tiempos de mi vida, lo supo compensar con su amor y apoyo en todo este tiempo.

A Camilo por criarme cuando niña y porque una vez más vuelve a entrar en mi vida con su apoyo incondicional.

A mis hermanos, y en especial a mi hermanito más pequeño Yeandris, al cual adoro y amo con la vida, por quererme y estar junto a mí por siempre.

A Lucia, Sonia, Nereida por ser mis tiitas, las más espectaculares de la tierra, por brindarme siempre todo su apoyo y comprensión, y por ser un ejemplo a seguir.

A mis primas por estar conmigo siempre, a Yenet y Yadira por su sentido del humor, a ti Yailo por ser como yo, desde pequeña vi mi reflejo en ti, las quiero con el alma, y a mi primo Yaniel por ser el negrito lindo de la familia, te quiero.

A mis amigos, en especial a ti miji por estar conmigo en los momentos que mas necesite de un familiar a mi lado, por ser la mejor amiga del mundo, si algo debo de agradecer a este dios es haberme dado la posibilidad de encontrar personas de tan buenos sentimientos como tú, además de darme la bendición de poderlos conocer a todos y compartir con ellos mi corazón durante este tiempo aquí en la UCI, esos que siempre han estado en todo momento ofreciéndome su apoyo incondicional y todo su amor: a ti miji nuevamente por ser como una hermana para mí, la hermana mayor que tanto necesite a mi lado, Liset, Estela, Maryin, Yelena, Delmis, Magnolia, Taimi Gongalez, Tahimi, Yeimi, Ana Lilian, Yaima, Enelis, Carla, Montano, Juan José, Jorge Luis, en fin a todos aquellos que de una forma u otra me apoyaron siempre, y sepan que siempre van a tener un lugarcito aquí en mi corazón que es tan noble.

A mi dúo de tesis, por estar junto a mí en todo momento, y en especial por esta bella amistad que fue creciendo junto a este triunfo.

A nuestros tutores por su paciencia, esmero, y en especial a ti Leysi por tu ayuda incondicional, y a Yuneimy que aunque no tuvo nada que ver con mi tesis siempre me apoyó.

Agradezco eternamente a esta revolución por ser tan grande, y en especial a nuestro comandante en jefe, al cual pensé siempre conocerlo frente a frente en esta universidad y dios no me dio la oportunidad, al cual le debo respeto y amor, por ser único e incomparable: Fidel Castro Ruz, a ti por esta extraordinaria idea y permitirnos forjarnos como todos unos profesionales, Gracias.

Yenisleydis

A todo el colectivo de profesores, que durante 5 años me han formado como profesional de los cuales guardo las mejores experiencias.

A mi compañera de tesis Yenisleydis Ledesma Rodríguez, por las carreras que juntos hemos dado, por compartir alegrías y tristezas, por ser la amiga que no he de olvidar.

A mis tutores y en especial a Armando Robert Lobo, por la confianza depositada y por todas y cada unas de las experiencias que me supo trasmitir.

A mi numerosa familia que espera lo mejor de mí, la cual no puedo defraudar.

A mis compañeros y amigos por estar allí siempre que necesité de su ayuda.

A esta universidad, que me ha formado y preparado como profesional y revolucionario.

A la Revolución Socialista y a su líder Fidel Castro Ruz por crear centros como la UCI.

Yunier.

DEDICATORIA.

Dedico este trabajo, fruto de tanto esfuerzo en especial a mi mamá, por ser la razón de mí existir que si no fuera por ella no estuviera aquí realizando nuestro sueño, a mi papá, a mi padrastro y a mi familia por ofrecerme todo su amor y apoyo incondicional, por haber sabido forjar en mí los mejores valores, y a quienes le debo cuanto soy.

Yenisleydis

Dedico este trabajo especialmente a mi madre Mercedes Roget Ortiz, por el amor y la dedicación conque me ha educado, el constante apoyo y confianza sin la cual no hubiese llegado hasta aquí.

A mi padre Luis David Boza Rios por su amor y apoyo durante toda la carrera.

A toda mi familia, de la cual siento orgullo de ser miembro.

A mis primos Carlos Antonio y Lilianne de la Caridad que sirva de motivación mi esfuerzo en sus proyecciones futuras.

Yunier

RESUMEN

En la Universidad de las Ciencias Informáticas (UCI) se desarrollan un número significativo de proyectos en centros de desarrollo de software vinculados a las facultades. El Centro de Tecnología de Gestión de Datos (DATEC) es uno de ellos, el mismo desarrolla sus productos utilizando herramientas como *"Visual Paradigm for UML"* y Ext JS durante el proceso de desarrollo de software. Esta investigación surge producto al inconveniente que posee la herramienta Visual Paradigm de no proveer la funcionalidad de generación de código para Ext JS a partir de un diseño de clases dado. El presente trabajo tiene como objetivo desarrollar una extensión de la herramienta *"Visual Paradigm for UML"* para soportar el Desarrollo Dirigido por Modelos (MDD) con Ext JS. Con esta extensión se pretende reducir el tiempo de desarrollo en la construcción de aplicaciones y garantizar un alto nivel en la calidad de la aplicación. Como resultado se obtuvo un prototipo funcional de extensión de la herramienta CASE *"Visual Paradigm for UML"* para el MDD con Ext JS. EL *plugin*[1] implementado se sometió a pruebas funcionales mediante las cuales se comprobó el correcto funcionamiento del mismo. El resultado alcanzado es de suma importancia pues beneficia directamente el proceso de desarrollo de software en DATEC.

Palabras claves: Ext JS, MDD, *plugin*, Visual Paradigm

[1] Un accesorio de software o paquete de hardware que se utiliza en conjunto con una aplicación existente o dispositivo para ampliar sus capacidades o proporcionar funciones adicionales.

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN.

La Universidad de las Ciencias Informática[2] (UCI) tiene un papel importante en el desarrollo de la industria cubana del software. Su modelo de producción está concebido en Centros de Desarrollo vinculados a las facultades y especializados en áreas temáticas. Dichos centros vinculan la producción, la investigación y la formación de los estudiantes en profesionales de la informática de alto nivel.

El Centro de Tecnología de Gestión de Datos (DATEC), tiene como misión proveer soluciones integrales y consultorías relacionadas con tecnologías de bases de datos y el análisis de información. La creación de nuevas tecnologías de bases de datos, procesamiento y presentación de la información a partir de proyectos de investigación y desarrollo, con enfoque en tecnologías soberanas. De esta forma el centro contribuye al cumplimiento de las misiones fundamentales de la universidad: la formación y la producción de software con profesionales integrales comprometidos y con un alto nivel científico y productivo. El mismo desarrolla un conjunto de activos informáticos para la producción de software sobre la base de un modelo de Línea de Producto de Software (LPS) que fortalece la tecnología de desarrollo, reduciendo en tiempo y elevando la calidad de sus productos.

El Centro DATEC está compuesto por cuatro líneas de productos de software: PostgreSQL, Almacenes de Datos, Bioinformática e Integración de Soluciones. Esta última desarrolla activos que serán reutilizados en su conjunto para formar productos; dichos productos son básicamente aplicaciones enriquecidas para internet. Entre las tecnologías que se potencia para el desarrollo de estas aplicaciones se encuentran PostgreSQL, PHP y Java Script.

Ext JS es un marco de Java Script para el desarrollo de aplicaciones enriquecidas para la web que goza de mucho prestigio. Su uso se ha generalizado por las potencialidades y calidad del mismo, constituyendo actualmente una tecnología base en la línea de Integración de Soluciones. A pesar de ser una tecnología madura existen pocas herramientas que brindan soporte al marco, en este sentido las herramientas de Ingeniería de Software Asistida por Computadora (CASE del inglés Computer Aided Software Engineering), como por ejemplo *"Visual Paradigm for UML"* son un elemento a favor del éxito cuando pueden ser utilizadas eficazmente en un proyecto. Los modelos diseñados a través de

[2] Universidad de las Ciencias Informáticas, Km 2 ½ carretera San Antonio de los Baños, Torrens, Boyeros, Ciudad de la Habana, Cuba.

"Visual Paradigm for UML" constituyen abstracciones que ocultan las complejidades de las tecnologías, para posteriormente a través de transformaciones generar el código fuente que resulta repetitivo y tedioso.

La herramienta *"Visual Paradigm for UML"* no provee la funcionalidad de generación de código fuente para Ext JS a partir de un diseño de clases dado, como consecuencia se requieren de desarrolladores experimentados capaces de traducir los artefactos del diseño a código fuente o en el peor y más común de los casos se presta poca atención a las tareas de diseño y se produce un solapamiento con las de implementación en detrimento de la calidad de la aplicación. Las funcionalidades de generación de código en las herramientas CASE contribuyen positivamente a la construcción de aplicaciones reduciendo los tiempos de desarrollo y garantizando elevados niveles de calidad.

En función de lo antes expuesto se identifica el **Problema Científico.**

¿Cómo contribuir a la construcción de aplicaciones con Ext JS a partir de los artefactos del diseño elaborados en la herramienta *"Visual Paradigm for UML"*?

Se define como **Objeto de Estudio:** Desarrollo Dirigido por Modelos (MDD).

Campo de Acción: Desarrollo de extensiones para la herramienta de modelado *"Visual Paradigm for UML"*.

Se persigue con ello el **Objetivo general** de: Desarrollar una extensión de la herramienta *"Visual Paradigm for UML"* para soportar el Desarrollo Dirigido por Modelos con Ext JS.

Para su consecución se han planteado los siguientes **Objetivos específicos:**

- ✓ Analizar la extensión de la herramienta *"Visual Paradigm for UML"* y el soporte al Desarrollo Dirigido por Modelos con UML.
- ✓ Identificar los elementos tecnológicos del marco Ext JS que serán tratados con el enfoque de Desarrollo Dirigido por Modelos para la generación de código fuente a partir de los artefactos del modelo de diseño.
- ✓ Realizar el análisis, diseño e implementación de la extensión.
- ✓ Realizar pruebas funcionales a la extensión implementada.

Para la realización de los objetivos propuestos se han planteado las siguientes **Tareas de investigación:**

1. Revisión bibliográfica de la documentación técnica de *"Visual Paradigm for UML"* referida a la extensión de la herramienta.
2. Estudio del Desarrollo Dirigido por Modelos y su vinculación con UML.
3. Definición de los elementos del marco Ext JS que serán modelados con UML haciendo uso de perfiles.
4. Definición de implementación de RPC (Remote Procedure Call, por sus siglas en inglés) con Ext.Drect.
5. Definición de los requisitos funcionales y no funcionales para el correcto funcionamiento de la extensión de *"Visual Paradigm for UML".*
6. Realización del diseño de clases para facilitar la implementación del *plugin.*
7. Selección de patrones a emplear para el desarrollo del *plugin.*
8. Implementación de escenarios de prueba e integración de *plugin* a la herramienta de *"Visual Paradigm for UML".*
9. Implementación del diseño para dar cumplimiento a los requisitos funcionales.
10. Realización de pruebas funcionales para comprobar el correcto funcionamiento del *plugin.*

Para lo que se refiere ha de utilizar los **Métodos:**

Teóricos:

- **Análisis y Síntesis**: Para el procesamiento de la información y arribar a las conclusiones de la investigación, así como para precisar la tecnología para la generación de código.
- **Histórico – Lógico:** Para determinar el proceso evolutivo del desarrollo dirigido por modelos en la industria del software.
- **Inducción y deducción:** A partir del estudio de principios y propuesta del MDD arribar a proposiciones específicas para su implementación.
- **Método Sistémico:** Para determinar los componentes de la solución, definir las relaciones entre estos e identificar las propiedades relevantes.

Empíricos:

- **Observación:** Para la percepción selectiva y sistemática de generación de código a través de modelos de UML.

Posibles resultados: Se espera obtener como principal resultado un prototipo funcional de extensión de la herramienta CASE *"Visual Paradigm for UML"* para el desarrollo dirigido por modelos con Ext JS.

Tal extensión beneficia directamente el proceso de desarrollo de software en DATEC, dado que a partir de un diseño se generaría el código fuente de los componentes diseñados en la herramienta *"Visual Paradigm for UML".* Se espera que la documentación técnica generada por la investigación y la implementación del *plugin* sirva de ayuda para posteriores implementaciones de extensiones de aplicaciones en el Centro y la Universidad.

Estructuración del trabajo de diploma:

Capítulo 1: MDD y el proceso de desarrollo de software
En este capítulo se analizan los elementos arquitectónicos a tener en cuenta para la implementación de extensiones a *"Visual Paradigm for UML"* y los principios de MDD con UML que serán aplicados para la generación de código. Se identifican los elementos tecnológicos de Ext JS que serán soportados por la extensión. Por último la selección de herramientas y tecnologías para el proceso de desarrollo del *plugin.*

Capítulo 2: Análisis y Diseño del Plugin
En este capítulo se define todo lo referente a las funcionalidades que debe realizar el sistema, se basa principalmente en las descripciones generalas del funcionamiento del mismo. Además se elaboran una lista de requisitos funcionales y no funcionales que se deben tener en cuenta para la realización del *plugin*. Se identifica el actor, casos de usos del sistema y la relación existente entre ellos, así como el diseño de clases con el propósito de describir cómo se debe implementar el sistema, y se describen las clases más relevantes. Se aborda sobre los patrones de diseño y arquitectónicos presentes en la implementación del *plugin.*

Capítulo 3: Implementación y Pruebas del Plugin
En este capítulo se presenta el modelo de implementación a través del diagrama de componentes. Además se realizan pruebas de caja negra al *plugin*, para comprobar su correcto funcionamiento mediante los casos de prueba.

CAPÍTULO 1: MDD y el Proceso de Desarrollo de Software.

En este capítulo se presenta brevemente los conocimientos básicos para el desarrollo de software con el Desarrollo Dirigido por Modelos (MDD). El mismo refleja todo un proceso vinculado al Lenguaje Unificado de Modelado (UML) y proporciona beneficios para el desarrollo del software utilizando modelos como guías en el proceso de desarrollo. Además se documenta la configuración para extensiones (*plugin*) para la herramienta *"Visual Paradigm for UML"*. Se definen los elementos tecnológicos soportados por Ext JS aplicando un perfil UML. Por último se seleccionan las herramientas y tecnologías para implementar el *plugin*.

1.1 Fundamentos Tecnológicos para el soporte de MDD.

1.1.1 Modelo

En el campo de las ciencias se encuentran inmersos todo una gama de modelos, específicamente en las ciencias puras y aplicadas se encuentran muy relacionadas con el empleo de los modelos científicos que no son más que una representación abstracta, conceptual, gráfica o visual, física, matemática, de fenómenos, sistemas o procesos a fin de analizar, describir, explicar, simular, en general, explorar, controlar y predecir esos fenómenos o procesos. Un modelo permite determinar un resultado final a partir de datos de entrada. Se considera que la creación de un modelo es una parte esencial de toda actividad científica. (1)

Existe poca teoría acerca del empleo de modelos en la ciencia moderna la cual ofrece una colección creciente de métodos, técnicas y teorías acerca de diversos tipos de modelos. En la práctica, diferentes ramas o disciplinas científicas tienen sus propias ideas y normas acerca de tipos específicos de modelos. Sin embargo, por lo general todos siguen los principios del modelado. Para hacer un modelo es necesario plantear una serie de hipótesis, de manera que lo que se quiere representar esté suficientemente plasmado en la idealización, aunque también se busca, normalmente, que sea lo bastante sencillo como para poder ser manipulado y estudiado. (1)

Estos modelos debido a la evolución de la sociedad, los cambios económicos, tecnológicos y culturales, son aplicados en diferentes ramas como en la ingeniería de software principalmente en el proceso de desarrollo del software, esto conlleva a nuevas exigencias del mercado y de los clientes. Ante estos cambios, se deben plantear nuevos modelos organizativos o analizar los antiguos buscando alternativas más eficaces que nos permitan aprovechar mejor todos los recursos disponibles a las empresas. (2)

Proveer modelos facilita la comunicación, tanto como para clientes como para especialistas en la elaboración de un proyecto de software. Por su naturaleza se dice que los modelos son simplificaciones; por lo que, un modelo de un procesos de software es una simplificación o abstracción de un proceso real. Se puede definir un modelo de procesos del software como una representación abstracta de alto nivel de un proceso software. (3)

Cada modelo es una descripción de un proceso que se presenta desde una perspectiva particular, donde se describen una sucesión de fases y un encadenamiento entre ellas. Según las fases y el modo en que se produzca este encadenamiento, tenemos diferentes modelos de proceso. Un modelo especifica la solución que será aplicada para problemáticas o incertidumbres que se presenten durante el desarrollo del sistema de software. Un modelo es una descripción simplificada de un proceso del software que presenta una visión de ese proceso. Estos modelos pueden incluir actividades que son parte de los procesos y productos de software y el papel de las personas involucradas en la ingeniería del software. (3)

1.1.2 Papel de UML en el Desarrollo Dirigido por Modelos

La Informática desde su inicio ha estado condicionada por cambios de paradigmas que han sido adoptados por la comunidad de programadores. El cambio de la programación estructurada a la programación orientada a objeto fue uno de ellos. La programación orientada a objeto (POO) tiene su origen en Simula 97 un lenguaje para la creación de simuladores, creado por Ole-Johan Dhi y Kristen Nygaard en Oslo, Noruega. La misma fue tomando posición dominante a mediados de 1980, en gran parte por la influencia de C++ extensión del lenguaje C. Muy aparejado al tema de la POO se daba los primeros paso a las metodologías de desarrollo las cuales guiaran el proceso de construcción de software. De esta forma surge la necesidad de modelar estas metodologías.

Para ese entonces fueron tres las metodologías que más se acercaron a la programación orientada a objeto James Rumbaugh en 1994 con el OMT (Object-model-ingtechnique), que era mejor para el análisis orientado a objeto, el Método Booch de Grady Booch, que era mejor para el diseño orientado a objetos y posteriormente en 1995 se uniera Ivar Jacobson, creador del método de ingeniería de software orientado a objetos. Las tres metodologías eran conocidas como “Los Tres Amigos”.

Los Tres Amigos fundaron el consorcio UML Partners en 1996 con el fin de desarrollar y especificar el UML. El borrador de la especificación UML 1.0 de UML Partners fue propuesto a la OMG en enero de 1997. Durante el mismo mes la UML Partners formo una Fuerza de Tarea Semántica, encabezada por Cris Kobryn y administrada por Ed Eykholt, para finalizar las semánticas de la especificación y para integrarla con otros esfuerzos de estandarización. El resultado de este trabajo, el UML 1.1, fue

presentado ante la OMG en agosto de 1997 y adoptado por la OMG en noviembre de 1997. Actualmente la versión liberada presentada por la OMG es UML 2.3.

Definición de UML:

"El Lenguaje Unificado de Modelado (UML), se define como un lenguaje visual de modelado de propósito general, que se utiliza para especificar, visualizar, construir y documentar los artefactos de un sistema software". (4)

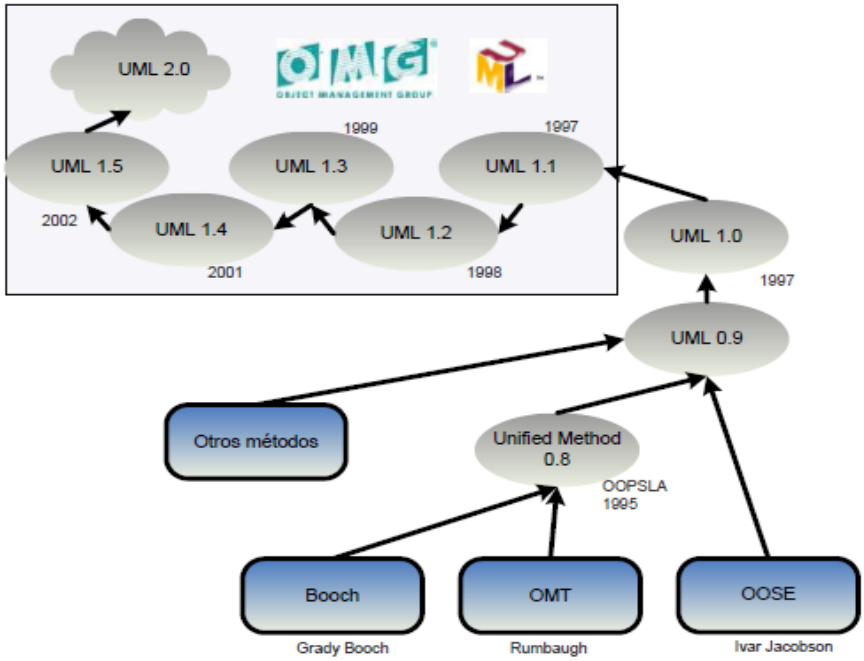


Figura # 1: Proceso evolutivo del UML

A lo largo de los años se ha visto surgir el Desarrollo de Software Dirigido por Modelos como una nueva área dentro del campo de la Ingeniería de Software (5). El MDD plantea una nueva forma de entender el desarrollo y mantenimiento de sistemas de software con el uso de modelos como principales artefactos en el proceso. Es una aproximación para el desarrollo de sistemas de software basado en la separación entre la especificación de la estructura, funcionalidades esenciales del sistema y la implementación final, usando plataformas de implementación específicas.

Dada la relevancia que tiene el Desarrollo Dirigido por Modelo no es de extrañar que aparezcan constantemente nuevas propuestas basadas en este esquema de desarrollo enfocados a los más diversos dominios de aplicación. En este contexto, el contar con un lenguaje de modelado adecuado es fundamental para la correcta aplicación de las distintas propuestas de MDD. Por este motivo,

muchas de las propuestas definen su propio lenguaje de modelado que se conocen como Lenguaje de Modelado Específico de Dominios (LMED)(5). Estos proveen la precisión semántica para describir sin ambigüedad los modelos conceptuales que participan en el proceso de desarrollo de software.

Existe otra alternativa para obtener un lenguaje de modelado que se ajuste a las necesidades de una propuesta de MDD: extender UML con la semántica requerida. UML por ser tan genérico no puede especificarlo todo con la precisión suficiente para el proceso de transformación de los modelos completos. Por esta razón se definen mecanismos de extensiones, perfil de UML que incorpora la precisión semántica de un LMED, convirtiendo UML en un LMED. Contar con una propuesta de MDD que de soporte a Líneas de Desarrollo de Software permite a los grupos de trabajo hacer mayor énfasis en la realización de modelos y las posibles transformaciones aplicadas a los mismos, aprovechando la gran variedad de tecnología UML existente reduciendo la curva de aprendizaje según autores (5).

Actualmente, la especificación de UML definida por OMG, incorpora una serie de características para mejorar las posibilidades de extensión de este lenguaje mediante el uso de perfiles de UML. Las nuevas características permiten definir la precisión semántica requerida para las diferentes propuestas de MDD, provocando un incremento considerable de perfiles de UML en los últimos años.

1.1.3 Enfoque MDD

El MDD se ha convertido actualmente en un importante paradigma de la Ingeniería de Software con el fin de manipular amplias complejidades y requerimientos de sistemas con gran cantidad de software. La idea fundamental de MDD es sustituir al código de lenguajes de programación específicos por modelos. De este modo y en el contexto de este paradigma, los modelos son considerados como entidades de primera clase, permitiendo nuevas posibilidades de crear, analizar y manipular sistemas a través de diversos tipos de herramientas y de lenguajes.

Cada modelo trata generalmente un aspecto, y las transformaciones entre modelos proporcionan un vínculo que permite la implementación automatizada de un sistema a partir de sus correspondientes modelos, generando también modelos, por lo tanto constituyen una parte integral de esta propuesta basada en modelos. Estas transformaciones requieren soporte especializado en varios aspectos para que: modeladores de sistemas, desarrolladores de transformaciones y desarrolladores de herramientas puedan aplicarla en su máximo potencial. (6)

MDD se sintetiza como la combinación de tres ideas complementarias:

1. **La representación directa**, ya que el foco del desarrollo de una aplicación se desplaza del dominio de la tecnología hacia las ideas y conceptos del dominio del problema, de este modo se reduce la distancia semántica entre el dominio del problema y su representación.

2. **La automatización**, porque promueve el uso de herramientas automatizadas en aquellos procesos que no dependan del ingenio humano, de este modo se incrementa la velocidad de desarrollo del software y se reducen los errores debidos a causas humanas.

3. **Los estándares abiertos**, debido a que éstos no solo ayudan a eliminar la diversidad innecesaria sino que, además, alientan a producir, a menor costo, herramientas tanto de propósito general como especializadas. (7)

1.1.4 Proceso de Desarrollo en MDD.

Se puede dividir, el proceso MDD en tres fases:

Primera fase: se construye un Modelo Independiente de Plataforma (PIM por sus siglas en inglés), que representa el modelo de más alto nivel del sistema, se desarrolla independientemente de cualquier tecnología y no contienen detalles de la plataforma concreta en que la solución va ser implementada. Estos modelos surgen como resultado del análisis y diseño.

Segunda fase: luego, se transforma a uno o varios Modelos Específico de Plataforma (PSM por sus siglas en inglés), derivados de la categoría anterior, que contienen detalles de la plataforma o tecnología con que se implementará la solución. Esto no es más que la combinación de los diferentes PIM que especifican el sistema con los detalles de cómo se utiliza un tipo de plataforma concreta. (8)

Tercera fase: por último, se genera un Modelo de implementación (IM por sus siglas en inglés), lo cual se traduce a código fuente a partir de cada PSM. (7)

Figura # 2: Proceso de transformación de modelos en MDD

El término “plataforma” en MDD se utiliza generalmente para hacer referencia a los detalles tecnológicos y de ingeniería que no son relevantes a la funcionalidad esencial del sistema. Esto se basa en la separación fundamental que se realiza en MDD entre la especificación de un sistema y su plataforma de ejecución. Se define plataforma como la especificación de un entorno de ejecución para un conjunto de modelos. (9)

El beneficio principal del enfoque MDD es que una vez que se ha desarrollado cada PIM, se puede derivar, automáticamente, el resto de los modelos aplicando las correspondientes transformaciones (7).

1.2 Elementos arquitectónicos para las extensiones a “Visual Paradigm for UML”.

1.2.1 Implementación de plugin para la herramienta “Visual Paradigm for UML”.

La implementación de extensiones para aplicaciones constituye un mecanismo para la incorporación de funcionalidades que la aplicación no provee a los usuarios. Por lo general se asocian las extensiones con *plugin*, que son fragmentos de software que interactúan con el núcleo de la aplicación para proporcionar algunas funcionalidades que en la mayoría de los casos son muy específicos. *“Visual Paradigm for UML”* es una de las herramientas CASE que goza de gran prestigio para el modelado de software, pero, presenta inconveniente para la generación de códigos de dominios en específicos como es el caso de Java Script. Sin embargo cuenta con los medios para extender funcionalidades dando soporte a las extensiones de aplicación. La aplicación provee de forma libre una interfaz de programación (API por su siglas en inglés Interfaz de Programación de Aplicaciones) permitiendo a los desarrolladores implementar y reutilizar clases e interfaces, desarrollando funciones agregadas que son útiles para el desarrollo de software.

¿Cómo se implementa un plugin para la herramienta “Visual Paradigm for UML”?

Para la implementación de un *plugin* en *“Visual Paradigm for UML”* es importante conocer la estructura de desarrollo, así como la de integración con la herramienta. Para la consecución de las acciones se debe realizar como:

Primer paso: La herramienta *“Visual Paradigm for UML”* provee el mecanismo de extensión a través de un openapi.jar. Es una librería que está ubicada en el paquete de instalación de la herramienta, dentro del paquete “lib/openapi.jar”. Una vez conocida cual es la librería de construcción de *plugin* es necesario tener en cuenta la estructura que conforma el *plugin* para su desarrollo. Para el IDE Netbeans tiene la estructura correspondiente a un paquete que lleva el nombre de *plugin* el cual contiene tres paquetes asociados, el primero, a la configuración del *plugin*, el segundo a las acciones

correspondientes y el tercero a los formularios o diálogos de la implementación. De ellas es importante dominar su funcionamiento y la relación que se establece por medio del openapi.jar incorporado al proyecto.

Para el primer paquete es necesario conocer que está conformado por un conjunto de clases que permite cargar y configurar el *plugin*, las mismas son Plugin.xml y Plugin.java.

Implementación de Plugin.xml.

Esta clases permite definir por medio de un script xml la configuración del *plugin* para ser cargado por la clase Plugin.java mediante la implementación de la interfaz VPPlugin la cual implementa los métodos *load* y *unload*, habilitando la carga y descarga del *plugin*. La misma permite el enlace con las librerías asociadas a la implementación así como las configuraciones de las acciones tanto a nivel de herramienta como a nivel de contexto cargando las clases correspondiente a dicha acción.

Ejemplo de la implementación del archivo Plugin.xml:

```
<plugin id="vpextmdd.Vpextmdd"
name="PlugIn de ejemplo VP."
description="PlugIn de ejemplo VP."
provider="VisaulParadim"
class="vpextmdd.Vpextmdd">
<runtime>
<library path="lib/vpextmdd.jar" relativePath="true"></library>
<library path="lib/AbsoluteLayout.jar" relativePath= "true"></library>
</runtime>
<actionSets>
<actionSet id="vpextmdd.actions.ActionSet1">
<action
                        id="vpextmdd.actions.Action1"
                        actionType="generalAction"
                        label="GenerateDataAccessLayer"
                        tooltip="Esta opción es para la realización de casos de pruebas"
                        icon="icons/add.png"
                        style="normal"
                        menuPath="Tools/Report"
```

```
                toolbarPath="vpextmdd.actions.Toolbar1/#">
                <actionController class="vpextmdd.actions.GenerateDataAccessLayer" />
</action>
</actionSet>
<contextSensitiveActionSet id="vpextmdd.actions.actions.ActionSet">
                        <contextTypes all="true">
                        </contextTypes>
                        <action
                                id="vpextmdd.actions.actions.Action1"
                                label="Generar Diagrama de Clases ORM Ext JS..."
                                icon="icons/add.png"
                                menuPath="OpenSpecification">
                                <actionController class="vpextmdd.actions.GenerateORMContextAction"/>
                        </action>
        </contextSensitiveActionSet>
</actionSets>
</plugin>
```

Ejemplo implementación de la clase Plugin.java:

```
public class Plugin implements com.vp.plugin.VPPlugin{

public void loaded(VPPluginInfovppi) {

}

publicvoidunloaded() {

}

}
```

El segundo paquete contiene las acciones del *plugin* definidas a nivel de herramientas y de contexto. Dichas clases en dependencia de la acción, implementa interfaces asociadas a las acciones, VPActionController y VPContextAction. Ambas clases definen el performAction el cual permite ejecutar acciones referentes al evento onClick de la acción.

El tercer paquete contiene los diálogos que se desea mostrar, los diálogos se muestra mediante el método performAction asociado a la clase de acción correspondiente al dialogo. Ejemplo de implementation:

```
public class ContextActionController implements com.vp.plugin.action.VPContextActionController {

ViewManagerviewManager = ApplicationManager.instance().getViewManager();

public void performAction(VPAction action, VPContextvpc, ActionEventae) {

FrmWindowsfrm = new FrmWindows();

viewManager.showDialog(frm);

   }

public void update(VPAction action, VPContextvpc) {

throw new UnsupportedOperationException("Not supported yet.");

}

}
```

Una vez implementado y estructurado el proyecto se está en condiciones de implementar una plantilla de *plugin* para la Herramienta *"Visual Paradigm for UML"* como resultado de lo antes explicado y primer resultado de la investigación.

Estructura de una plantilla de plugin:

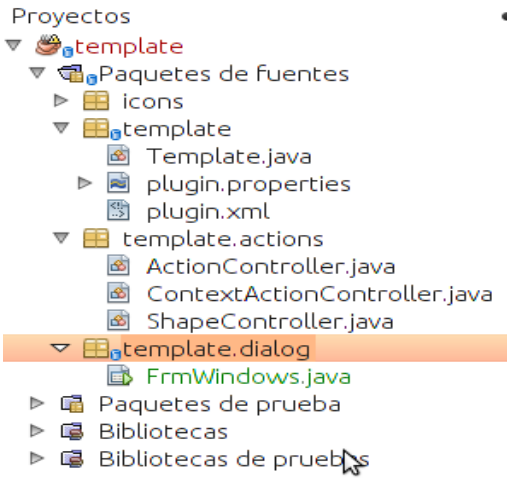


Figura # 3: Estructura de un proyecto plugin para "Visual Paradigm for UML" en IDE NetBeans

Segundo paso: integración de *plugin* a la herramienta *"Visual Paradigm for UML".* Para la integración de *plugin*, *"Visual Paradigm for UML"* propone la siguiente estructura de despliegue de *plugin*. Se crea una carpeta con el nombre de *plugins* dentro de la carpeta de instalación de Visual Paradigm la cual contiene la siguiente estructura de paquete:

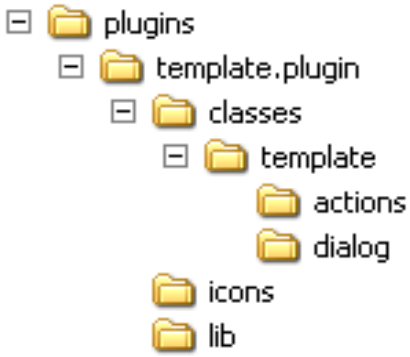


Figura # 4: Estructura de despliegue de plugin para "Visual Paradigm for UML"

Si se observa la estructura de implementación del *plugin*, es diferente a la de integración con la herramienta, lo que dificulta el proceso de pruebas al integrar el *plugin* implementado con *"Visual Paradigm for UML"*. La solución a este inconveniente fue implementar una herramienta de despliegue de *plugin* que facilite la integración del mismo con *"Visual Paradigm for UML"* se obtiene como segundo resultado de la investigación, la herramienta de despliegue de *plugin* "Deployer" pasando los valores iniciales "nombre del *plugin*", "dirección de proyecto *plugin* (origen ejemplo: /home/yunier/NetBeansProjects/Código_Fuente/vpextmdd/src/vpextmdd)" y "dirección donde será desplegado (destino ejemplo: /opt/VP_Suite3.1)".

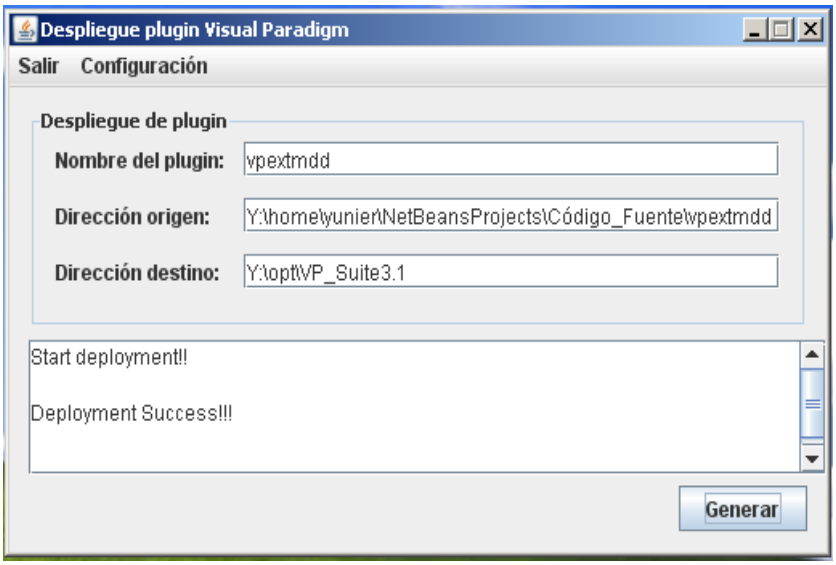


Figura6: Herramienta de despliegue de plugin para "Visual Paradigm for UML".

De esta forma se está, en condiciones de implementar un *plugin* para la herramienta *"Visual Paradigm for UML"*, no sin antes mencionar el dominio de la arquitectura del openapi.jar para el cual se ofrece la documentación de la misma, donde se facilita la comprensión de la arquitectura y jerarquías existentes en el proceso de construcción de *plugin*.

1.3 Elementos tecnológico soportado por Ext JS.

1.3.1 Definición de los elementos tecnológico soportados por Ext JS aplicando un perfil UML.

Para el desarrollo de interfaces con Ext JS es relevante el manejo de datos persistentes en componentes, así como el uso de métodos RPC (Remote Procedure Call por su siglas en inglés) implementado por el marco a través de Ext.Direct. El marco implementa en el paquete Ext.data dos clases fundamentales, las cuales facilitan todo el proceso de almacenamiento de la información estas son: **Ext.data.Record** y **Ext.data.Store.**

Record: guarda un conjunto de valores de los objetos para posteriormente ser almacenados en el Store.

Store: encapsula la información del cliente recogidos en los Record, por lo que, contiene una colección de objetos Record.

El Store provee los servicios de datos a diferentes componentes como los que a continuación se muestra en la figura 4.

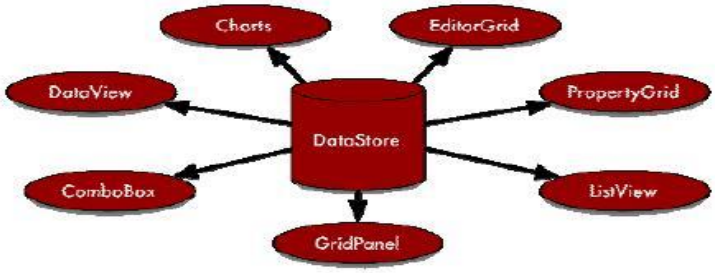


Figura # 5: Recursos que consumen del DataStore

Cada uno de estos componentes presenta un flujo de datos para su funcionamiento los cuales involucran diferentes instancias para su implementación como son las clases Ext.data.DataReader, Ext.data.DataProxy, Ext.data.DataWriter representada en la figura 5.

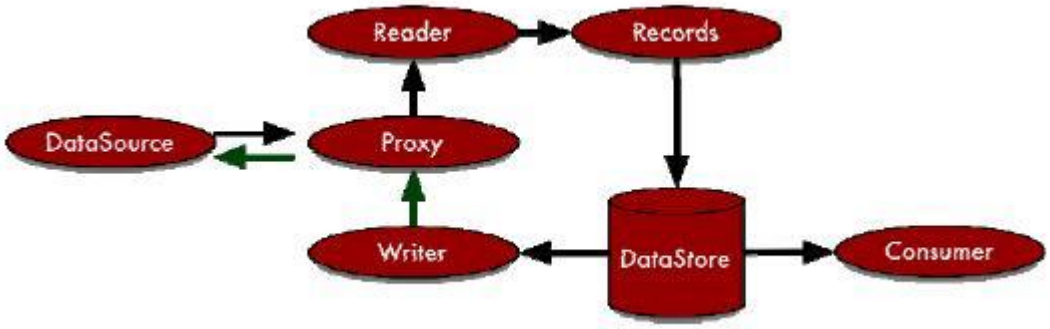


Figura # 6: Flujo de datos del DataStore

En la actualidad esta implementación no satisface las necesidades producto al aumento de flujo de información hacia bases de datos que hoy demanda las aplicaciones por lo que se propone la implementación de patrones de diseño como **Active Record** para el manejo de datos.

Por las razones antes mencionadas y la solución adoptada por parte del equipo de desarrollo se decide que el proceso de generación de clases del *plugin* se incorpore por cada clase Record una clase Store que implementa dichos patrones y los estereotipos definidos en el perfil UML para Java Script.

RPC: La llamada a procedimientos remotos es una tecnología basada en un protocolo de comunicación que permite a un programa ejecutar un servicio que se encuentra en otro ordenador. (Procesamiento distribuido, arquitecturas cliente servidor)

Ext.Direct: Ext.Direct se utiliza para realizar RPC, es una técnica donde el servidor expone sus objetos y luego mediante Java Script tiene acceso a ellos de una manera muy sencilla. Se encarga de realizar automáticamente las peticiones que se realizan del lado del servidor. Es una plataforma independiente de lenguaje que permite la comunicación entre el cliente de una aplicación Ext JS y las plataformas del servidor. Ext.Direct es un nuevo paquete de ExtJS3.0 que ayuda a la comunicación entre el cliente y el servidor. Además, de introducir nuevas clases para la integración con el servidor. Las nuevas clases también se añaden al espacio de nombres Ext.data para trabajar con Ext.data.Stores que están respaldadas por datos de un método Ext.Direct, también utiliza una arquitectura de proveedor, cuando uno o más proveedores lo utilizan para el transporte de datos hacia y desde el servidor.

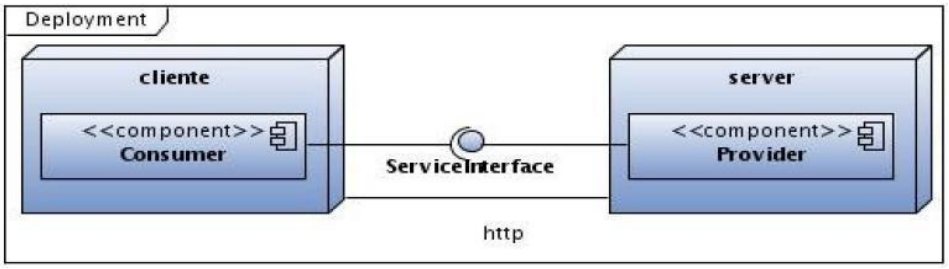


Figura # 7: Comunicación entre Cliente y Servidor

1.4 Metodología para el desarrollo de software

1.4.1 OpenUp

OpenUp es una variante ágil del Proceso Unificado que aplica el desarrollo iterativo e incremental dentro de la estructura del ciclo de vida. Se adopta el pragmatismo, y la filosofía ágil que se centra en la colaboración natural del desarrollo de software. Es una metodología ligera que puede ser utilizada en varios tipos de proyectos de software.

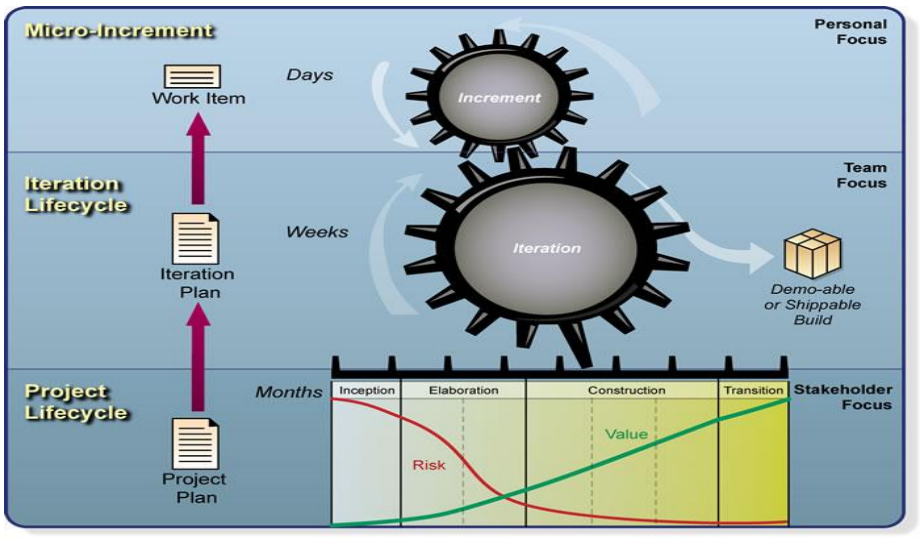


Figura # 8: Capas de la Metodología "OpenUP"

Desde la perspectiva de los stakeholders[3] del proyecto OpenUp estructura el ciclo de vida en 4 fases: Inicio, Elaboración, Construcción y Transición. El ciclo de vida provee la visibilidad y los puntos de decisión tanto para stakeholders como miembros del equipo, permitiendo una vigilancia efectiva del proceso de desarrollo y facilitado la toma de decisiones apropiadas en cada instante.

Desde la perspectiva del equipo el proyecto consta de iteraciones planeadas con encajonamiento de tiempo en intervalos que no pasan de unas pocas semanas y centradas en producir de una manera predecible un incremento del valor para los stakeholders. El artefacto Plan de Iteración define el entregable de la iteración y si este entregable es demostrativo o explotable y el artefacto Plan de Proyecto define concretamente el ciclo de vida y su resultado final es la liberación de la aplicación. Es muy importante tener en cuenta que cada iteración concluye obligatoriamente con una entrega concreta del producto que necesariamente tiene que ser "demostrativa" o "explotable", siendo esta la forma que tiene la metodología para demostrar al cliente que se le está agregando valor al producto. Esta es la clave detrás del discurso ágil de OpenUp y todos los miembros deben estar enfocados en ella. El equipo se auto-organiza centrado en cómo lograr los objetivos de la iteración y obtener el entregable. Por supuesto el producto entregable (demostrativo o explotable) es el primer objetivo, le suceden la reducción de riesgos. Se emplea un ciclo de vida de la iteración que se organiza como el conjunto del micro-incremento necesario para lograr los objetivos de la iteración con énfasis en la

[3] Stakeholder es un término inglés utilizado por primera vez por R. E. Freeman en su obra: "Strategic Management: A Stakeholder Approach", (Pitman, 1984) para referirse a «quienes pueden afectar o son afectados por las actividades de una empresa»..

liberación de un entregable consistente y estable.

Desde la perspectiva individual de cada miembro del equipo el proyecto consta de micro-incrementos, pequeñas unidades de trabajo, que producen un continuo y mesurable ritmo de progreso del proyecto generalmente contado en horas o pocos días. Una colaboración intensiva entre los comprometidos en el micro-incremento le da el carácter incremental al desarrollo. Cada micro-incremento provee información de retroalimentación que permite tomar decisiones adaptativas al desarrollo en la iteración.

Respecto al planteamiento de la necesidad de una colaboración intensiva OpenUp está pensado para equipos pequeños donde los canales de comunicación son pocos (de hecho el primer principio ágil reza los individuos y la interacción por encima de los procesos y herramientas) en contraposición una producción de factoría requiere de líneas de producción con entradas y salidas perfectamente definidas que permitan desacoplar al máximo la interacción entre individuos, promoviendo el ensamblaje en el sentido de que cada cual produce una pieza (salida) para ello sabe específicamente como hacerlo y consume exclusivamente las piezas que recibió de antemano (entradas), por supuesto no descarta la interacción entre los individuos pero se busca minimizarla.

OpenUp es recomendable para equipos pequeños, donde sus miembros se encuentren trabajando en un mismo sitio interactuando cara a cara. Un equipo incluye interesados, desarrolladores, arquitectos, gestor de proyecto, y probadores. El equipo toma sus propias decisiones sobre que se requiere realizar, cuales son las prioridades y cómo abordar los requerimientos y necesidades de los interesados. Junto con la colaboración intensa la presencia de los interesados en el equipo es crítica para el éxito.

El ciclo de vida del proyecto provee tanto a stakeholders como a los miembros del equipo de visibilidad, de puntos de sincronización, y de puntos de decisión sobre el proyecto, hace posible la vigilancia del proyecto facilitando en cada momento la toma de decisión. (10)

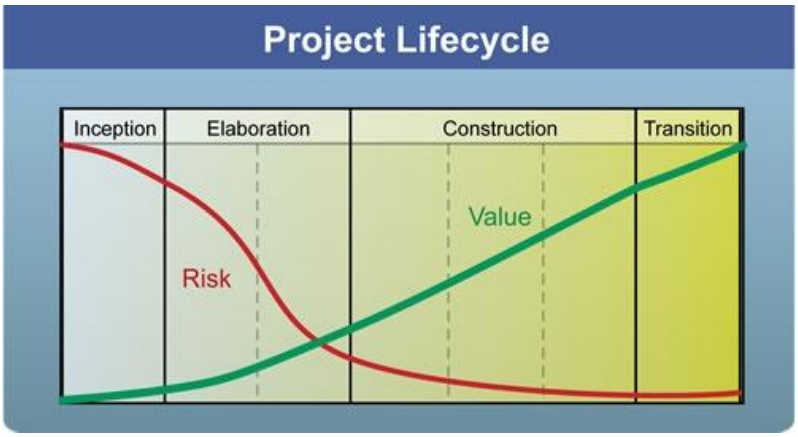


Figura # 9: Ciclo de vida de OpenUP

Teniendo en cuenta lo antes expuesto se decide asumir como metodología de desarrollo OpenUp en consideración con las políticas del centro DATEC para el desarrollo de software.

1.5 Lenguaje de Programación

Un lenguaje de programación describe un conjunto de acciones que un equipo debe ejecutar. Está conformado por un conjunto de símbolos y reglas sintácticas y semánticas que definen su estructura y el significado de sus elementos y expresiones. Al conjunto de instrucciones que se genera se conoce como código fuente de un programa.

- ✓ **Programación Orientada a Objeto**

La programación orientada a objeto (POO) es un paradigma de programación que define los programas en términos de "clases de objetos", objetos que son entidades que presentan estados, comportamientos e identidad. La POO expresa un programa como parte del conjunto de objeto, facilitando la creación de programas, mantenimiento y reutilización de código. La POO incorpora nuevos conceptos como son: Clases, Objeto, Métodos, Eventos, Estados, Atributos, Componentes de Objetos, Representación de Objetos. Presenta características propias como es la Abstracción, Encapsulamiento, Principios de Ocultación y Polimorfismo.

1.5.2 Lenguaje de programación Java

Java es un lenguaje de programación orientado a objeto, de plataforma independiente desarrollado por *Sun Microsystems* en los años 90. El lenguaje toma sintaxis de lenguajes como C y C++, aunque tiene un modelo de objeto más simple y elimina herramientas de bajo nivel. Su característica distintiva lo ha llevado a niveles muy alto en la preferencia de los desarrolladores de la comunidad internacional.

Para la implementación del *plugin* se empleó este lenguaje, por ser el lenguaje de programación propuesto por el API de desarrollo de la herramienta *"Visual Paradigm for UML"*.

1.6 Herramientas a utilizar.

1.6.1 Visual Paradigm for UML.

Visual Paradigm for UML es una herramienta que soporta el ciclo de vida completo en el desarrollo de software: análisis y desarrollos orientados a objetos, construcción, prueba y despliegue. Permite dibujar todo tipo de diagrama de clases, código inverso, generación de código a partir de diagramas y generar documentación (11). Entre sus características fundamentales se tiene:

Multiplataforma: Soportada en plataforma Java para Sistemas Operativos *Windows, Linux, Mac OS*.
Interoperabilidad: Intercambia diagramas UML y modelos con otras herramientas. Soporta la Importación y Exportación a formatos *XMI y XML* y archivos *Excel*. Permite importar proyectos de *Rational Rose*[4] y la integración con *Microsoft Office Visio*[5].
Modelado de Requisitos: Captura de requisitos mediante diagramas de requisitos, modelado de caso de uso y análisis textual.
Colaboración de Equipo: Realiza el modelado simultáneamente con el *Paradigm TeamWork Server* y *Subversión*.
Generación de Documentación: Comparte y genera documentación de diagramas y diseños en formatos *PDF, HTML, Microsoft Work.*
Editor de Detalles de Caso de Uso: Entorno para la especificación de detalles de casos de usos, incluyendo la especificación del modelo general y las descripciones de los casos de uso.
Ingeniería de Código: Permite la generación de código e ingeniería inversa para los lenguajes: *Java, C, C++, PHP, XML, Python, C#, VB .Net, Flash, ActionScript, Delphi y Perl.*
Modelado de Procesos de Negocio: Visualiza, comprende y mejora los procesos de negocio con la herramienta para procesos de negocio.

[4] Rational Rose herramienta de diseño de software destinado para el modelado visual y componentes para la construcción de aplicaciones de software a nivel empresarial utilizando Lenguaje Unificado de Modelado (UML).

[5] Un programa de dibujo y diagramación para Windows de Microsoft que incluye una variedad de formas pre-dibujadas y elementos de imagen que se pueden arrastrar y soltar en la ilustración. Los usuarios pueden definir sus propios elementos y colocarlos en la paleta de Visio. El paquete de Visio es parte de la marca de Microsoft Office. Ediciones Standard y Professional están disponibles.

Integración con Entornos de Desarrollo: Apoyo al ciclo de vida completo de desarrollo de software en IDE como: *Eclipse, Microsoft Visual Studio, NetBeans, Sun ONE, Oracle JDeveloper, Jbuilder* y otros.

Modelado de Bases de Datos: Generación de bases de datos y conversiones de diagramas entidad-relación a tablas de bases de datos, además de mapeos de objetos y relaciones.

Se utilizó esta herramienta por sus facilidades, además de ser la herramienta de modelado escogida por la Universidad para el desarrollo de software, asumida por el centro DATEC en sus producciones.

1.6.2 IDE NetBeans 6.9

El IDE *NetBeans* es un entorno integrado de desarrollo galardonado disponible para *Windows, Mac, Linux y Solaris*. *NetBeans* consiste en un IDE de código abierto y una plataforma de aplicaciones que permiten a los desarrolladores crear rápidamente aplicaciones web, escritorio y aplicaciones móviles utilizando la plataforma Java, así como *JavaFX, PHP, Java Script y Ajax, Ruby y Ruby onRails, Groovy y Grails, y C/C++*.*NetBeans* IDE 6.9 introduce el Compositor de *JavaFX*, una herramienta de diseño visual para construir visualmente aplicaciones *JavaFX* interfaz gráfica de usuario, similar a la del constructor *Swing GUI* para aplicaciones *Java SE* con el compositor de *JavaFX*. Los desarrolladores en el Compositor de *JavaFX* pueden crear rápidamente, visualmente editar y depurar aplicaciones dinámicas de Internet (RIA) y los componentes se unen a diversas fuentes de datos, incluidos los servicios Web. (12)

Se asume por las características y comodidades que ofrece el IDE NetBeans 6.9 como herramienta para el desarrollo de *plugin* para la herramienta *"Visual Paradigm for UML"*.

Conclusiones parciales.

Después de realizar un estudio sobre la evolución de UML y su vinculación con el Desarrollo Dirigido por Modelos, enfocado al proceso de desarrollo de software; el análisis de las metodologías y tecnologías necesarias para el desarrollo de la herramienta, y teniendo en cuenta principalmente las exigencias y necesidades del cliente, se define, OpenUP como metodología de desarrollo de software ya que es una metodología ágil para proyectos de corta duración diseñada para pequeños equipos de trabajo. Utilizando como lenguaje de modelado UML adoptado por la Universidad como estándar para el desarrollo de software; permitiendo la visualización de los artefactos del sistema. Se emplea Java como lenguaje de programación, propuesto por el API de desarrollo de la herramienta *"Visual Paradigm for UML"*. Como herramienta de modelado *"Visual Paradigm for UML"* en su versión 6.1, adoptado en el Centro DATEC en su entorno tecnológico.

CAPÍTULO 2: Análisis y Diseño del Plugin.

En el presente capítulo se describen los conceptos más importantes en el entorno donde estará el sistema. Se muestran los requerimientos funcionales y no funcionales que se deben tener en cuenta para la implementación del *plugin.* Además, se identifica el actor, casos de usos y las relaciones existentes entre ellos. Se aborda acerca de cómo debe funcionar el sistema y se hace una descripción general de las actividades. Se muestra además los diagramas de clases del diseño, así como los diagramas de secuencia correspondientes que serán objeto de automatización.

2.1 Propuesta de Solución

Se propone realizar una extensión de la herramienta *"Visual Paradigm for UML",* mediante un *plugin* que permita a partir del diseño Entidad Relación obtener un Diagrama de Clases con sus relaciones asociadas. El mismo tendrá aplicado un perfil de UML definido por medios de estereotipos, que expresan la semántica del perfil. Así como, la generación de código fuente partiendo de la interpretación de los modelos de UML llevados al lenguaje Java Script.

2.2Modelo de Dominio

Un Modelo del Dominio captura los tipos de objetos más importantes que existen, o los eventos que suceden en el entorno donde estará el sistema, se identifican conceptos, se definen estos conceptos y se unen o relacionan en un diagrama de clases UML. El objetivo fundamental de este modelo es comprender y describir los conceptos más importantes dentro del contexto del sistema. El modelo de dominio es una representación visual del entorno real del proyecto.(13)

2.2.1Diagrama conceptual del dominio

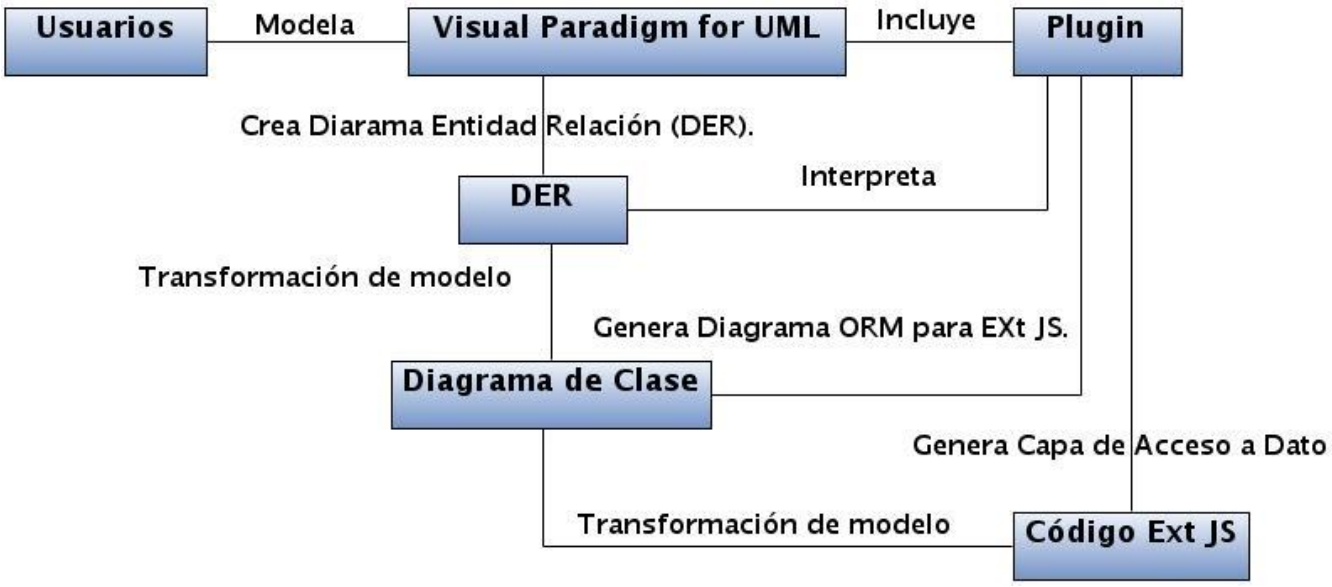


Figura # 10: Modelo de Dominio

Definición de conceptos del Modelo de Dominio

- **Usuarios:** son los desarrolladores responsables de modelar en la herramienta *"Visual Paradigm for UML".*

- **Visual Paradigm:** es una herramienta de modelado que permite crear diversos modelos entre los que se encuentra el modelo entidad relación que representa el punto de partida para las posteriores operaciones.

- **Vpextmdd:** representa el *plugin*, contiene todas las funcionalidades que permite a partir de un diagrama entidad relación generar el diagrama de clases, y a partir de este generar el código para Ext JS**.**

- **Diagrama Entidad Relación:** es una percepción del mundo real representado por entidades, que son objetos que existen y son importantes en el problema a resolver, que incluyen atributos y se relacionan entre ellos. Es el punto de partida, para la generación del diagrama de clases a través de transformación de modelos.

- **Diagrama de Clases:** es un diagrama representado por clases, que son objetos relacionados entre sí, que contienen atributos y operaciones. Representan la fuente de obtención de código a partir de la realización del diagrama entidad relación antes mencionado.

- **Código Ext JS:** Contiene el código Ext JS referente al diagrama de clases, que se obtiene aplicando transformaciones a los modelos.

2.3 Especificación de los Requisitos del sistema

2.3.1 Requisitos Funcionales

Los requisitos funcionales (RF) son capacidades o condiciones que el sistema debe cumplir. Los requisitos funcionales definen las funciones que el sistema será capaz de realizar. Expresan la naturaleza del funcionamiento del sistema cómo interacciona el sistema con su entorno y cuáles van a ser su estado y funcionamiento. (14)

Los requisitos funcionales deben:

- Estar redactados de tal forma que sean comprensibles para usuarios sin conocimientos técnicos avanzados (de Informática).
- Especificar el comportamiento externo del sistema y evitar, en la medida de lo posible, establecer características de su diseño.
- Priorizarse (al menos, se ha de distinguir entre requisitos obligatorios y requisitos deseables).

Por esta razón se identifican los siguientes requisitos funcionales:

RF1. Generar diagramas de clases a partir de diagramas de objeto relacional.

Descripción: se obtiene el Diagrama de Clases a partir del diseño del diagrama objeto relacional.

Entrada: nombre del diagrama, espacio de nombre (dirección donde se van a guardar las clases que se generan), prefijo de nombre, métodos a generar (*set, get*, para relación de uno a uno, para relación de uno a mucho, remotos (*load, save, remove*))

Salida: diagrama de clases.

RF2. Incluir en las clases métodos remotos (*load, save, remove*).

Descripción: se incluyen los métodos remotos (*load, save, remove*) en cada clase del diagrama generado.

Entrada: métodos remotos (*load, save, remove*).

Salida: las clases con los métodos remotos incluidos.

RF3. Incluir en las clases métodos *set* por atributos.

Descripción: se incluyen las operaciones set por atributo en cada clase del diagrama generado.

Entrada: métodos *set*.

Salida: las clases con los métodos *set* incluidos.

RF4. Incluir en las clases métodos *get* por atributos.

Descripción: se incluyen las operaciones *get* por atributo en cada clase del diagrama generado.

Entrada: métodos *get.*

Salida: las clases con los métodos *get* incluidos.

RF5. Incluir en las clases métodos de relación de uno a uno.

Descripción: se incluyen las operaciones para la relación de uno a uno en cada clase del diagrama generado.

Entrada: métodos para la relación de uno a uno.

Salida: las clases con los métodos para la relación de uno a uno incluidos.

RF6. Incluir en las clases métodos de relación de uno a muchos.

Descripción: se incluyen las operaciones para la relación de uno a mucho en cada clase del diagrama generado.

Entrada: métodos para la relación de uno a mucho.

Salida: las clases con los métodos para la relación de uno a mucho incluidos.

RF7. Generar implementación de la capa de acceso a datos a partir del diagrama de clases.

Descripción: se obtiene el código fuente para Ext JS a partir del diagrama de clases generado.

Entrada: nombre del proveedor, servicios vinculados.

Salida: vista previa del código a generar y código fuente para Ext JS.

RF8. Vincular servicios.

Descripción: se incluyen en el código fuente los métodos correspondientes a los servicios remotos que desee consumir.

Entrada: métodos remotos.

Salida: se vinculan los servicios

RF9. Mostrar Vista Previa.

Descripción: se muestra en un *TabbePanel* en la interfaz del caso de uso generar capa de acceso a datos el código que se genera correspondiente a cada clase.

Salida**:** se muestra el código.

2.3.2 Requisitos No Funcionales

Los requisitos no funcionales (RNF) son propiedades o cualidades que el producto debe tener. Estas propiedades o cualidades se refieren a las características que hacen al producto atractivo, usable, rápido o confiable. Por lo general los requisitos no funcionales son fundamentales en el éxito del producto; normalmente están vinculados a los requisitos funcionales, es decir, una vez que se conoce lo que el sistema debe hacer se puede determinar cómo ha de comportarse, qué cualidades o propiedades debe tener. (14)

Los requisitos no funcionales:

_ Han de especificarse cuantitativamente, siempre que sea posible para que se pueda verificar su cumplimiento.

Se definen los siguientes requisitos no funcionales asumiendo los de la herramienta *"Visual Paradigm for UML",* ya que el *plugin* por sí solo no cumple funcionalidad*:*

Requisitos de Software

- La herramienta *"Visual Paradigm for UML"*.
- Se debe tener el JRE (Java Runtime Environment).

Requisitos de Hardware

- Intel Pentium III, Compatible con procesador a 1, 0 GHz o superior.
- Mínimo 512 MB de RAM (Random Access Memory, por sus siglas en inglés), pero se recomienda 1,0 GB.
- Un mínimo de 800 MB de espacio en disco.

Restricciones del diseño y la implementación

- Se hace uso de la herramienta *"Visual Paradigm for UML"* en su versión 6.1 y IDE NetBeans 6.9.
- El lenguaje de programación que será usado para la implementación es Java siguiendo el paradigma de la Programación Orientada a Objeto.

Requisitos de Usabilidad

- Facilidad de uso por parte de los usuarios: La interfaz debe ser lo más descriptiva posible, permitiendo que las operaciones a realizar por los usuarios estén bien descritas, de manera que se puedan entender claramente.
- La interfaz debe tener mensajes contextuales asociados a los objetos.
- Permitir la accesibilidad a las operaciones, en caso de no tener mouse.

Requisitos de Soporte

- Proveer un manual de usuario.

Requisitos de Portabilidad

- El *plugin* una vez integrado a la herramienta visual Paradigm, podrá ser instalado y disponer del mismo en diferentes sistemas operativos por ser visual Paradigm una herramienta multiplataforma.

2.4 Modelo de Casos de Usos del Sistema

El modelo de casos de uso del sistema representa las relaciones existentes entre actores y casos de uso. Un modelo de casos de uso describe la funcionalidad propuesta del nuevo sistema, representa una unidad discreta de interacción entre un usuario (humano o máquina) y el sistema. Un caso de uso es una unidad de trabajo significativo. Cada caso de uso tiene una descripción que especifica la funcionalidad que se incorporará al sistema propuesto. Un caso de uso puede 'incluir' la funcionalidad de otro caso de uso o puede 'extender' a otro, con su propio comportamiento (15).

2.4.1 Actores del sistema

Un actor es un usuario del sistema, que usa un caso de uso para desempeñar alguna porción de trabajo que es de valor para el negocio. El conjunto de casos de uso al que un actor tiene acceso define su rol global en el sistema y el alcance de su acción. Además son generalmente responsables de realizar actividades que serán automatizadas en el futuro sistema (16).

Nombre del Actor	Descripción
Desarrollador	Es el encargado de realizar diagramas de clases de objeto relacional, para generar de forma automática el diagrama de clases con cada una de sus relaciones, y posteriormente generar el código.

Tabla # 1: Descripción del actor del sistema

2.4.2 Diagrama de Casos de Uso del Sistema

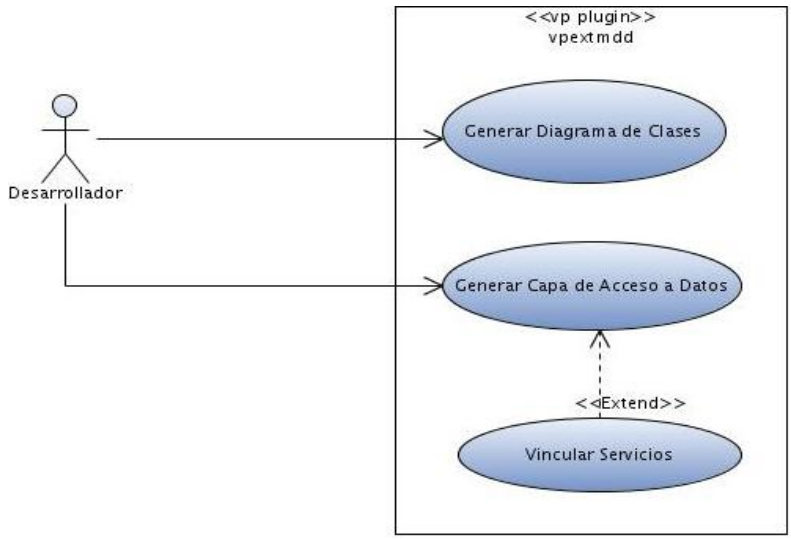


Figura # 11: Diagrama de Casos de Uso del Sistema

2.4.3 Descripción textual de los casos de uso del sistema

Caso de Uso del Sistema: Generar Diagrama de Clases.

Caso de Uso:	Generar Diagrama de Clases.

Actores:	Desarrollador
Resumen:	El caso de uso se inicia cuando el desarrollador define las entidades que serán llevadas a diagrama entidad relacional, para que posteriormente se genere el diagrama de clases. El sistema una vez realizado el diagrama entidad relación, debe permitir al desarrollador desde el contexto de trabajo generar el diagrama de clases, una vez realizado, el desarrollador deberá registrar los datos a través de una interfaz y de esta forma se termina el caso de uso.
Precondiciones:	-
Referencias	**RF1, RF2**, **RF3, RF4, RF5, RF6.**
Prioridad	Crítico

Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1. El desarrollador realiza el diagrama entidad relación. 2. El desarrollador hace clic derecho desde el contexto de trabajo y selecciona la opción generar diagrama de clases.	3. El sistema muestra una interfaz que incluye todo lo referente a la figura 12. - La interfaz le permitirá al desarrollador tener todas las entidades seleccionadas para transformar a clases. - Incluir métodos remotos (*save, load, remove*). - Incluir un las clases métodos set y get por atributos. - Incluir en las clases métodos de relación de uno a uno y de uno a

	muchos.
4. El desarrollador registra la configuración de los datos.	5. Si el desarrollador desea mantener todas las entidades seleccionadas o realiza otra selección múltiple o sencilla, el sistema subraya la selección. 6. Si el desarrollador decide incluir métodos remotos, el sistema adiciona en cada clase del diagrama que se genera, los métodos remotos (*save, remove, load*). 7. Si el desarrollador decide incluir métodos set y *get*, el sistema adiciona en cada clase del diagrama que que se genera, los métodos *set* y *get* por atributos. 8. Si el desarrollador decide incluir métodos para la relación de uno a uno y de uno a mucho, el sistema adiciona en cada clase del diagrama que se genera dichos métodos, referidos a la relación existente entre las entidades, es una opción obligatoria.
9. Presiona el botón “Aceptar”.	10. El sistema visualiza el diagrama de clases que se genera y termina el caso de uso.
Flujos Alternos	

Acción del Actor	Respuesta del Sistema
9.1 Si presiona el botón " Cancelar" se deja de realizar automáticamente la acción y finaliza el caso de uso.	

Prototipo de Interfaz

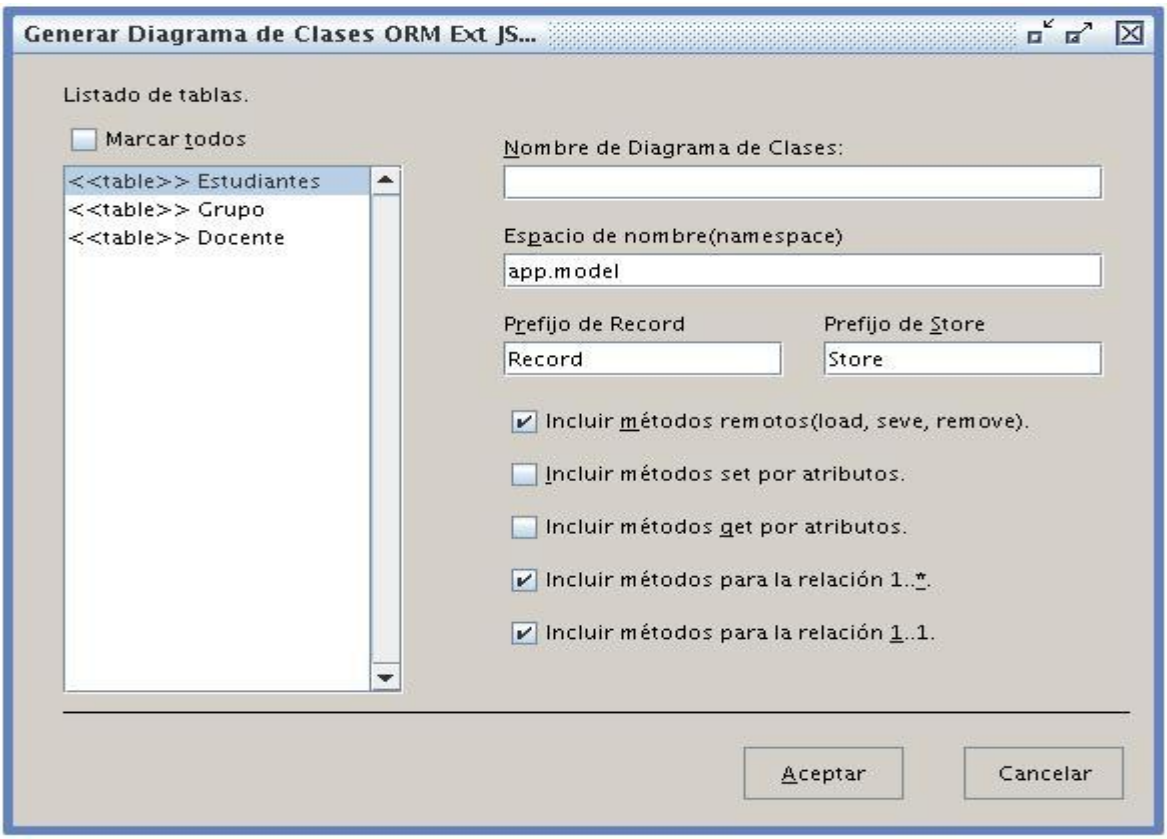


Figura # 12: Prototipo IU Generar Diagramas de Clases ORM EXT JS

Poscondiciones	Quedó generado el diagrama de clases.

Tabla # 2: Descripción textual del CU: Generar Diagrama de Clases

Caso de Uso del Sistema: Generar Capa de Acceso a Datos.

Caso de Uso:	Generar Capa de Acceso a Datos.

Actores:	Desarrollador
Resumen:	El caso de uso se inicia cuando el desarrollador define el diagrama de clases que será llevado a código, así como los servicios que se va a consumir. El sistema le debe permitir al desarrollador desde el menú de herramienta, generar la capa de acceso a datos, una vez realizada la acción se mostrará una interfaz donde se le da la posibilidad al desarrollador de registrar la configuración de los datos y de esta forma se termina el caso de uso.
Precondiciones:	-
Referencias	**RF7, CU:** Vincular Servicios(extiende)**, RF9**
Prioridad	Crítico

Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1. El desarrollador desde el menú de herramienta selecciona la opción de generar la capa de acceso a datos.	2. El sistema muestra una interfaz que incluye todo lo referente a la figura 13. - La interfaz la permitirá al desarrollador tener todas las clases seleccionadas para codificar. - Si desea consumir servicios remotos con algún proveedor de Ext.Direct. - Y una vista previa del código a generar.
3. El desarrollador registra la configuración de los datos.	4. Si el desarrollador desea mantener todas las clases seleccionadas o realiza otra selección múltiple o sencilla, el

	sistema subraya la selección. 5. El desarrollador selecciona los servicios remotos a consumir, el sistema subraya los servicios seleccionados.
6. Presiona el botón “Generar”.	7. El sistema muestra una vista previa de la generación de código por cada clase.
8. Presiona el botón “Guardar”.	9. El sistema muestra una ventana de dialogo, en la cual se especificará la dirección donde será guardado el código generado para cada clase.
10. El desarrollador selecciona la dirección donde se guardará el código y presiona el botón guardar.	11. El sistema guarda el código en la dirección especificada y termina el caso de uso.
Flujos Alternos	
Acción del Actor	**Respuesta del Sistema**

Prototipo de Interfaz

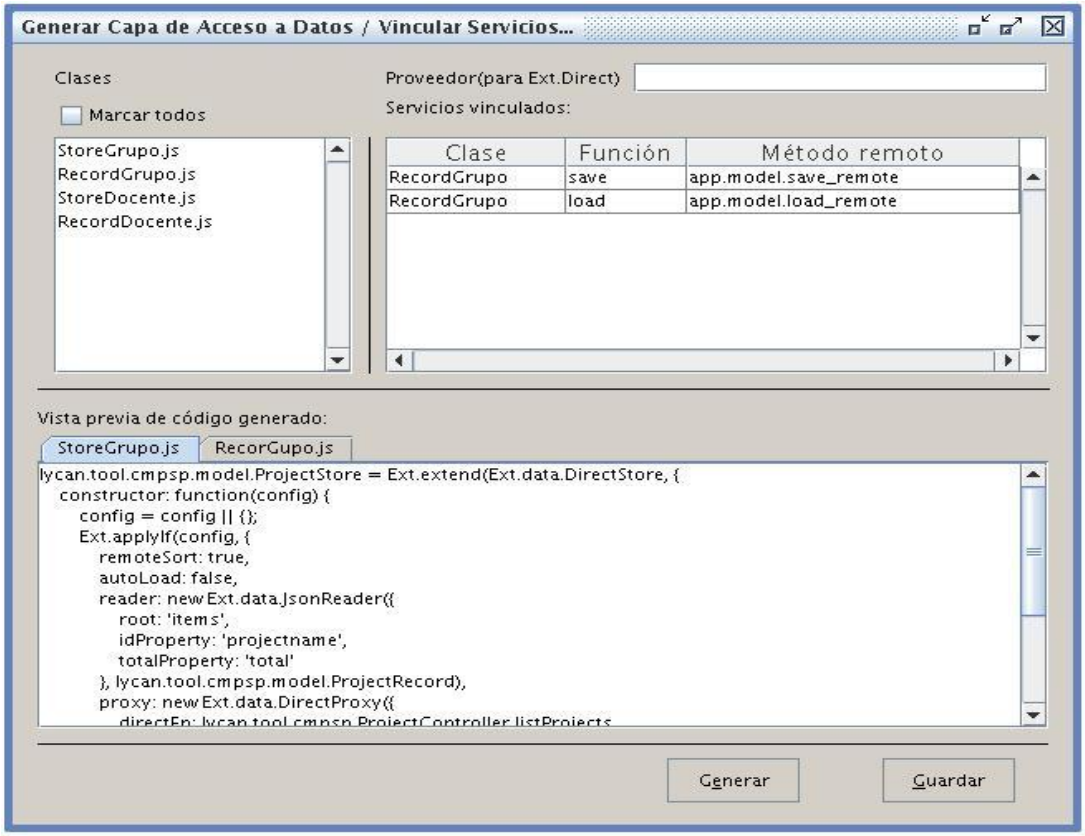


Figura # 13: Prototipo IU Generar Capa de Acceso a Datos/Vincular Servicios

Poscondiciones	Quedó generado el código.

Tabla # 3: Descripción textual del CU: Generar Capa de Acceso Datos

Caso de Uso del Sistema <<Extendido>>: Vincular Servicios.

Caso de Uso:	Vincular Servicios.
Actores:	-
Resumen:	El caso de uso se inicia una vez iniciado el caso de uso base Generar Capa de Acceso a Datos. El caso de uso vincular servicio puede no ejecutarse, eso lo decide el desarrollador cuando selecciona los

	servicios que se van a consumir. Una vez seleccionados estos servicios el sistema muestra las funciones que se realizan, así como los métodos correspondientes.
Precondiciones:	**-**
Referencias	**RF9,** Caso de uso base: Generar Capa de Acceso.
Prioridad	Secundario

Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
	1. El sistema a través de la interfaz referente a la figura 13, muestra una tabla con todos los servicios que se pueden consumir.
2. El desarrollador selecciona los servicios que puede consumir.	3. El sistema subraya el servicio seleccionado, y vincula este servicio remoto al código que se genera.
Flujos Alternos	
Acción del Actor	**Respuesta del Sistema**

Tabla # 4: Descripción textual del CU: Vincular Servicios

2.4.4 Matriz de Trazabilidad

La matriz de trazabilidad es una técnica que relaciona los requisitos funcionales a los diferentes elementos del desarrollo. Es una herramienta que se utiliza para saber que requisito quedan cubiertos por casos de uso.

	RF1	**RF2**	**RF3**	**RF4**	**RF5**	**RF6**	**RF7**	**RF8**	**RF9**
CU1	X	X	X	X	X	X			

CU2							X	X	X
CU3								X	

Tabla # 5: Matriz de trazabilidad para los casos de uso

Gracias a estos datos se puede saber o identificar qué requisitos quedan cubiertos por caso de uso, y de esta forma garantizar que todos los elementos para el desarrollo del *plugin* sean ejecutados correctamente.

2.5 Modelo de diseño

El modelo de diseño es un modelo de objetos que describe la realización de casos de uso, y sirve como una abstracción del modelo de aplicación y su código fuente. El modelo de diseño se utiliza como parte esencial para las actividades en ejecución y prueba, que se basa en el análisis y los requisitos de la arquitectura del sistema. Representa los componentes de aplicación y determina su colocación adecuada y el uso dentro de la arquitectura en general del sistema(17).

2.5.1 Diagrama de clases del diseño

Diagrama de Clases del Diseño - CU Generar Diagrama de Clases

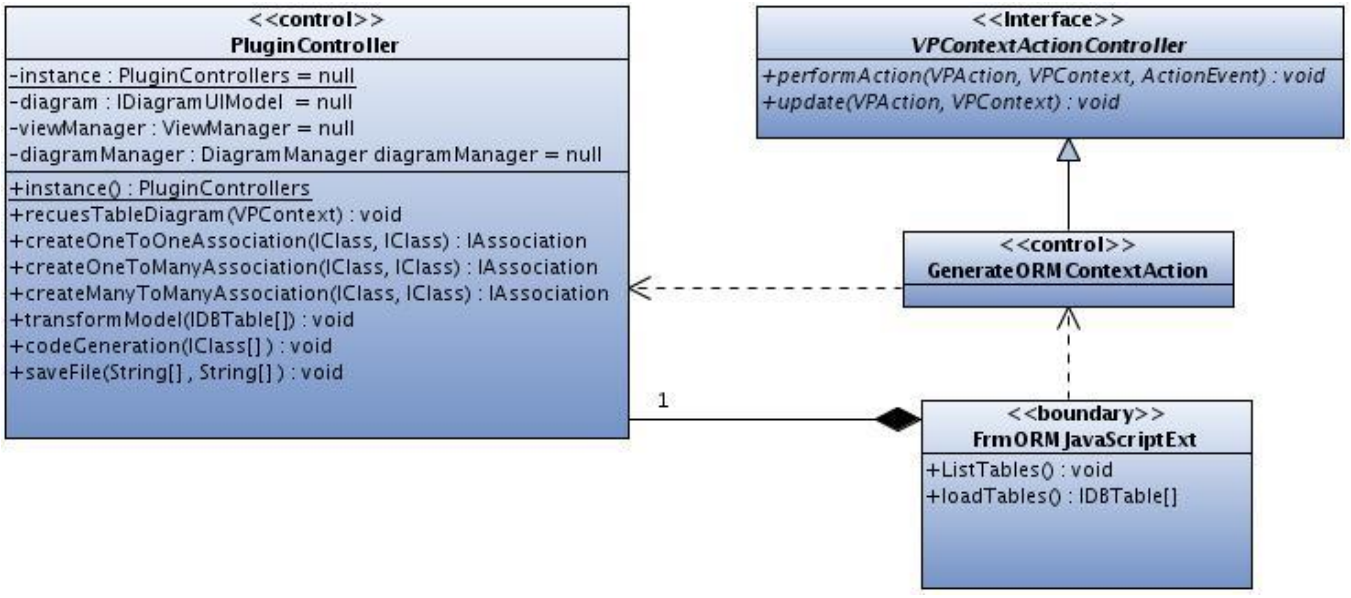


Figura # 14: Diagrama de Clases del Diseño - CU Generar Diagrama de Clases

Diagrama de Clases del Diseño - CU Generar Capa de Acceso a Datos.

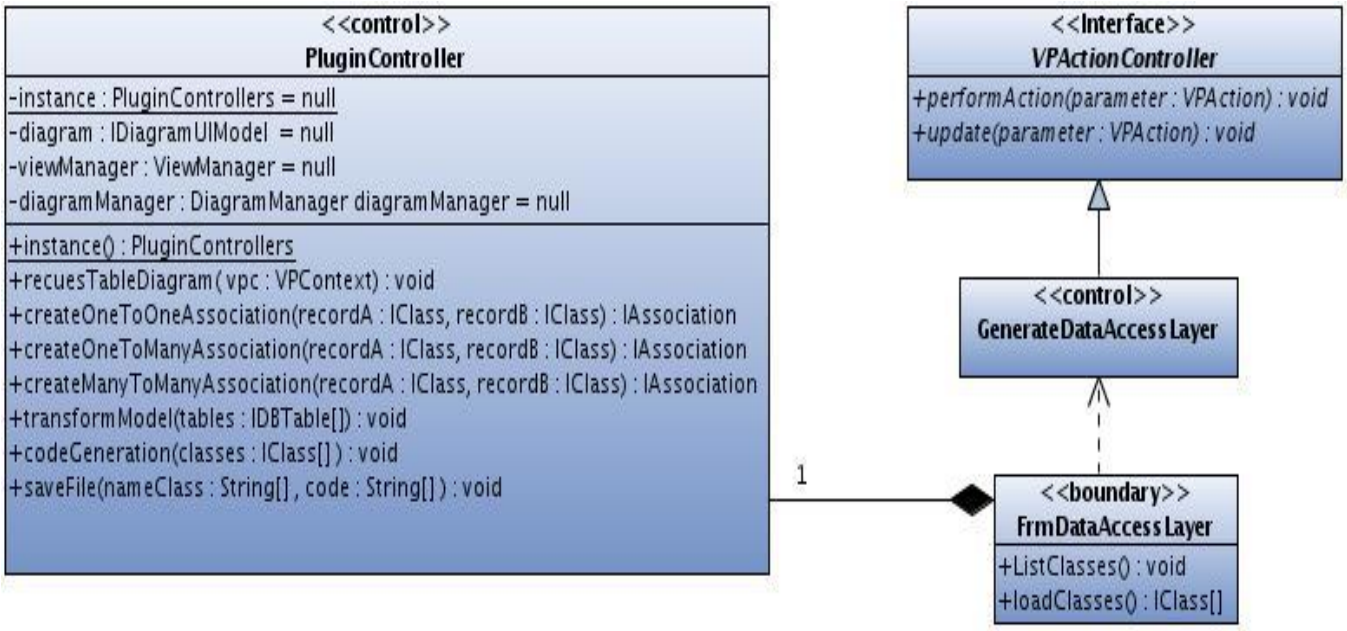


Figura # 15: Diagrama de Clases del Diseño - CU Generar Capa de Acceso a Datos

2.5.2 Descripción de clases relevantes del diseño

Diagrama de Clases del Diseño - CU Generar Diagrama de Clases

#	Clase	Tipo de clase	Descripción	Relación con otra clase	Dependencia
01	VPContextActionController	Interface	Es la clase que brinda el OpenAPI por defecto, la que permite actualizar cada acción realizada a través del contexto, así como ejecutar las acciones en el propio contexto.	02	Herencia

02	GenerateORMContextAction	Control	Es la clase permite la visualización de las interfaces.	03	Composición
03	PluginController	Entity	Es la clase que contiene todos los elementos del formulario, además, del control de todas las acciones que se realizan.	-	-

Tabla # 6: Descripción de las clases relevantes del diseño para el CU: Generar Diagrama de Clases

Diagrama de Clases del Diseño - CU Generar Capa de Acceso a Datos.

#	Clase	Tipo de clase	Descripción	Relación con otra clase	Dependencia
01	VPActionController	Interface	Es la clase que brinda el OpenAPI por defecto, la que permite actualizar cada acción realizada a través de la herramienta, así como ejecutar las acciones.	02	Herencia
02	GenerateDataAccessLayer	Control	Es la clase permite la visualización de las interfaces.	03	Composición

03	PluginController	Entity	Es la clase que contiene todos los elementos del formulario, además, del control de todas las acciones que se realizan.	-	-

Tabla # 7: Descripción de las clases relevantes del diseño para el CU: Generar Capa de Acceso a Datos

2.5.3 Patrones Utilizados

Muchas son los autores que definen que es un patrón de diseño, pero todas convergen en la definición dada por Christopher Alexander y la cual tomamos como punto de partida:

"… cada patrón describe un problema que ocurre una y otra vez en nuestro entorno, y luego describe el núcleo de la solución a ese problema, de tal forma que puede usar esa solución un millón de veces más, sin haberlo hecho de la misma forma dos veces"(18).

Para la implementación y funcionalidad del *plugin* debe implementarse patrones arquitectónicos como los que a continuación se describe, así como los patrones de diseño GOF con el fin de agilizar el proceso de implementación al desarrollador a través de la transformación de modelos.

2.5.3.1 Patrones Arquitectónicos

- **Active Record (Registro Activo)**

Descripción:

Un Active Record, es un objeto que contiene una estructura de datos del registro en un recurso externo, como una fila en una tabla de base de datos y añade un poco de lógica de dominio para ese objeto, donde un objeto contiene los datos y el comportamiento del mismo. Gran parte de estos datos es persistente, y deben ser almacenados en una base de datos. Active Record utiliza el enfoque más obvio: poner la lógica de acceso a datos en el objeto del dominio. De esta manera todas las personas pueden leer y escribir los datos de la base de datos (19).

Según el libro Los Patrones de Arquitectura de Aplicación Empresarial de Martin Fowler describe el patrón Active Record como:

"…un objeto que representa una fila en una tabla de una base de datos, encapsula su acceso a la base de datos y añade lógica de negocio. Es la aproximación más obvia, poniendo el acceso a la base datos en el propio objeto de negocio. De este modo es evidente como manipular la persistencia a través del mismo".

¿Cuándo utilizarlo?

Active Record es una buena opción cuando la lógica de dominio no es demasiado compleja, tales como crear, leer, actualizar y eliminar, derivaciones y validaciones sobre la base de un trabajo único según la estructura. Active Record tiene la principal ventaja de la simplicidad. Es fácil de construir redes activas y son fáciles de entender. El principal problema con ellos es que funcionan bien sólo si los objetos de Active Record se corresponden directamente con las tablas de base de datos: un esquema isomorfo[6].

.

- **Foreign Key Mapping (Mapeo de llave foránea)**

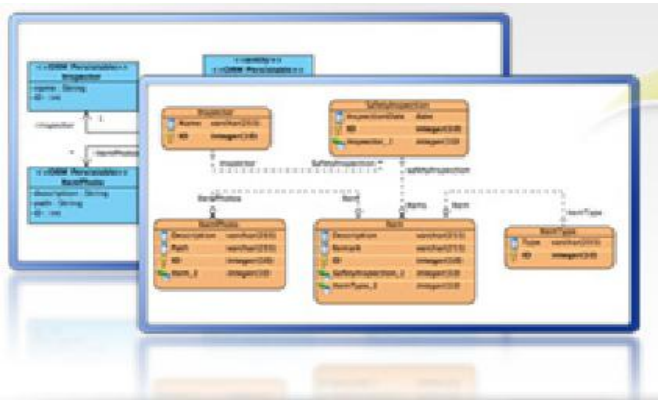


Figura # 16: Ejemplo de transformación Foreign Key Mapping

Descripción:

El mapeo de llave foránea, mapea una asociación entre objetos hacia una llave foránea como referencia entre las tablas. Donde los objetos se refieren el uno al otro directamente a través de referencias existentes entre ellos. Para guardar estos objetos a una base de datos es vital salvar a estas referencias. Esto se complica aún más por el hecho de que los objetos pueden tener colecciones

[6] Isomorfo: que tienen la misma forma. http://www.wordreference.com/definicion.

de referencias a otros objetos. Esta estructura viola la primera forma normal de bases de datos relacionales. Una asignación de clave externa asigna mapas de una referencia de objeto a una clave externa en la base de datos. La clave obvia a este problema es la identidad de Campo. Cada objeto contiene la clave de la base de datos de la tabla de base de datos correspondiente. Si dos objetos están unidos entre sí con una asociación, esta asociación puede ser sustituida por una clave externa en la base de datos.(19).

¿Cómo funciona?

Para la implementación del *plugin* el patrón de mapeo de llave foránea, se utiliza en la transformación de los objetos, garantizando la relación entre las clases una vez ejecutado el primer paso de transformación de los modelos. Este patrón permite la comunicación entre las clases una vez transformados dichos modelos, manteniendo la cardinalidad entre las clases relacionadas.

2.5.3.2 Patrones de diseño GOF

Los patrones de diseño Gof, son clasificados según su propósito en creacionales, estructurales y de composición, mientras que respecto a su ámbito se clasifican en clases y objetos.(20)

- ✓ **Según su propósito:**

1. De Creación: Resuelven problemas relativos a la creación de objetos.
2. Estructurales: Resuelven problemas relativos a la composición de objetos.
3. De Comportamiento: Resuelven problemas relativos a la interacción entre objetos.

- ✓ **Según su ámbito:**

1. Clases: Relaciones estáticas entre clases.
2. Objetos: Relaciones dinámicas entre objetos.

- ➢ **Factory Method (Método de fabricación)**

```
public IOperation createOperationGet(String name) {
    IOperation operation = IModelElementFactory.instance().createOperation();
    String operationName = name.substring(0, 1).toUpperCase().concat(name.substring(1));
    operation.setName("get".concat(operationName));
    return operation;
}
```

Figura # 17: Ejemplo del método de fabricación

Descripción:

El Método de Fabricación, es de tipo creación a nivel de clases que define una interfaz para crear un objeto, permitiendo a las subclases decidir de qué clase instanciarlo. Permite que una clase delegue en sus subclases la creación de objetos. Con este método se gana en flexibilidad, se facilita en cuanto a que se hace natural, la conexión entre jerarquías de clases paralelas, que son aquellas que se generan cuando una clase delega algunas de sus responsabilidades en una clase aparte. Ambas jerarquías de clases paralelas son creadas por un mismo cliente y el patrón método de fabricación establece la relación entre parejas de subclases.

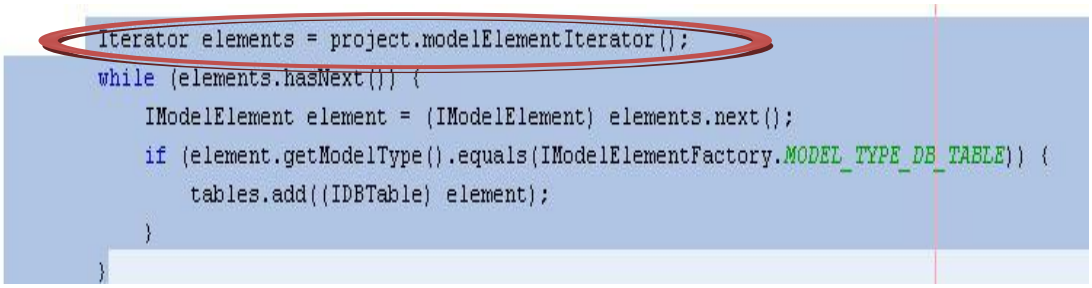
¿Cuándo utilizarlo?

Se debe utilizar cuando una clase no puede adelantar las clases de objetos que debe crear, cuando una clase pretende que sus subclases especifiquen los objetos que ella crea, además cuando una clase delega su responsabilidad hacia una clase, entre varias subclases auxiliares y se quiere tener localizada a la subclase delegada.(20)

¿Cómo funciona?

Para la implementación del *plugin* el patrón método de fabricación, se utiliza directamente en la implantación, garantizando la instanciación de los objetos, utilizando métodos creacionales, para crear componente y garantizar a través de estos, que el *plugin* funcione de forma correcta.

- **Iterator (Iterador)**



```
Iterator elements = project.modelElementIterator();
while (elements.hasNext()) {
    IModelElement element = (IModelElement) elements.next();
    if (element.getModelType().equals(IModelElementFactory.MODEL_TYPE_DB_TABLE)) {
        tables.add((IDBTable) element);
    }
}
```

Figura # 18: Ejemplo del método de iteración

Descripción:

El patrón Iterador, es de tipo comportamiento a nivel de objetos que proporciona un modo de acceder secuencialmente a los elementos de un objeto agregado sin exponer su representación interna. Permite realizar recorridos sobre objetos compuestos independientemente de la implementación de estos. Con la aplicación de este patrón se incrementa la flexibilidad, dado que para permitir nuevas formas de recorrer una estructura basta con modificar el iterador en uso, cambiarlo por otro o definir uno nuevo. Además se facilitan el paralelismo y la concurrencia, puesto que, como cada iterador tiene consciencia de su estado en cada momento, es posible que dos o más iteradores recorran una misma estructura simultánea o solapadamente.

¿Cuándo utilizarlo?

Se debe utilizar cuando se quiere acceder a los elementos de un objeto agregado sin mostrar su representación interna, cuando se quieren permitir recorridos múltiples en objetos agregados y cuando se quiera proporcionar una interfaz uniforme para recorrer diferentes estructuras de agregación.(20)

¿Cómo funciona?

Para la implementación del *plugin* el patrón iterador al igual que el método de fabricación, se utiliza directamente en la implantación, para iterar sobre el proyecto, obteniendo una colección de elementos y captar de esta forma información a través de operadores secuenciales.

- **Singleton (Solitario)**

Descripción:

Es de tipo creacional, a nivel de objetos. Su propósito es garantizar que una clase sólo tenga una única instancia, proporcionando un punto de acceso global a la misma. El acceso a la “Instancia Única” es controlado. Permite refinamientos en las operaciones y en la representación, mediante la

especialización por herencia de “Solitario”. Es fácilmente modificable para permitir más de una instancia y, en general, para controlar el número de las mismas (incluso si es variable). (20)

¿Cuándo utilizarlo?

Se utiliza cuando debe haber únicamente una instancia de una clase y debe ser claro su acceso para los clientes.

¿Cómo funciona?

El patrón Singleton o Solitario es implementado por la clase PluginController.java que permite la creación de objetos a través de instancias, manteniendo la consistencia entre los objetos.

2.6 Descripción del Proceso de Transformación de los Modelos en “Visual Paradigm for UML”.

El proceso de transformación de los modelos en la herramienta *“Visual Paradigm for UML”* se realizan a partir de la obtención de los componentes presentes en los diagramas.

Diagrama Entidad Relación	
Componentes	**Descripción**
IDBTable	Interfaz que define una tabla o entidad en el diagrama entidad relación
IDBColumn	Interfaz que define una columna dentro de una tabla en un diagrama entidad relación
IDBForeignKey	Interfaz que define en una tabla la llave foránea para las relaciones entre tablas.
IRelationShip	Interfaz que define una relación entre cualquier objeto IModelElement.
Diagrama de Clases	
IClass	Interfaz que define una clase en un diagrama de clases.

IAttribute	Interfaz que define los atributos de una clase.
IOpreration	Interfaz que define las operaciones asociadas a la clase.
IAssociation	Interfaz que define la relación entre clases de asociación que permite definir la agregación o composición.
IDiagramUIModel	Interfaz que permite visualizar los diagramas.
IDiagramElement	Interfaz que define los elementos de un diagrama.
ApplicationManager	Interfaz que permite el control de la aplicación.

Tabla # 8: Descripción de los componentes, para el proceso de transformación.

Para la transformación de Diagrama Entidad Relación a Diagrama de clases se captura todos los elementos *IDBTables* presentes en el diagrama activo. El proceso es sencillo se crea una instancia de la clase *VPContext* descrita en el openapi.jar la cual permite obtener el diagrama activo mediante el método *getDiagram()*, posteriormente se captura todos los componente, en este caso se refiere a la interfaz más genérica asociado a los componentes, el *IModelElement* del cual heredan cada unos de los componentes antes mencionados. Luego se verifica si el componente es de tipo Modelo de Base de Datos en este caso una tabla, de esta forma tenemos el objeto tabla del cual podemos extraer información de sus columnas que serán convertidos a atributos en la clase asociada a la tabla.

Para crear la clase se hace mediante el uso de la interfaz *IModelElementFactory* la cual implementa un patrón *singleton*, creamos una instancia y mediante el método de fabricación *createClass()*, es creada la clase. La clase está compuesta por un nombre, atributos y operaciones que son incluidas a la misma. Para dar nombre a la clase basta con cambiar mediante el método *setName()* asociado a la *IClass,* pasando como nombre el de la tabla mediante el método *getName()* asociado al *IDBTable*. Para la creación de atributos y operaciones se utiliza la misma interfaz *IModelElementFactory* definida por el openapi.jar para la creación de objetos *IModelElement.* Luego se itera sobre la tabla obteniendo una colección de campos para agregarlos como atributos a la clase mediante los métodos *addAttribute()* y *addOperation().* Si los métodos o atributos presentan estereotipos se les define mediante el método *addStereotype()* del objeto *IAttribute o IOperation.* De esta forma queda establecida la transformación de un objeto tabla a clase.

Para definir las relaciones entre clases se obtiene las relaciones entre tablas a través del mapeo de llaves foránea patrón arquitectónico de Fowler. Las relaciones son capturadas mediante la iteración de las *IRelationShip* asociadas a las tablas por medio del método *fromRelationshipIterator()* determinando cada una de las relaciones de una tabla con todas las demás. Una vez definida la relación que se establece entre las tablas, se procede a crear la *IAssociation* haciendo uso nuevamente de la interfaz *IModelElementfactory* definiendo la clase de donde parte la relación y hacia qué clase está dirigida.

Una vez creado las clases y las relaciones entre ellas solo queda visualizar el diagrama, para ello es necesario haber creado un *IDiagramUIModel* mediante la interfaz *ApplicationManager.* Esta permite crear un diagrama, de tipo "Diagrama de Clase" adicionando *IDiagramElement* al diagrama, convirtiendo los *IClass* o sea las clases a elementos del diagrama mediante el método *createDiagramElement()* asociado a la interfaz *ApplicationManager.* Una vez convertido las clases y adicionado al *IDiagramUIModel* es visualizado mediante la *ApplicationManager* a través del método *openDiagram(),* visualizando en el área de trabajo de *"Visual Paradigm for UML"* el diagrama de clases creado mediante la transformación del modelo Entidad Relación.

2.7 Diagramas de interacción

2.7.1 Diagramas de secuencia

Un diagrama de secuencia muestra la interacción entre usuarios, sistemas y subsistemas, y hace hincapié en la ordenación del tiempo de los mensajes.

Diagrama de Secuencia – Escenario: Generar Diagrama de Clases

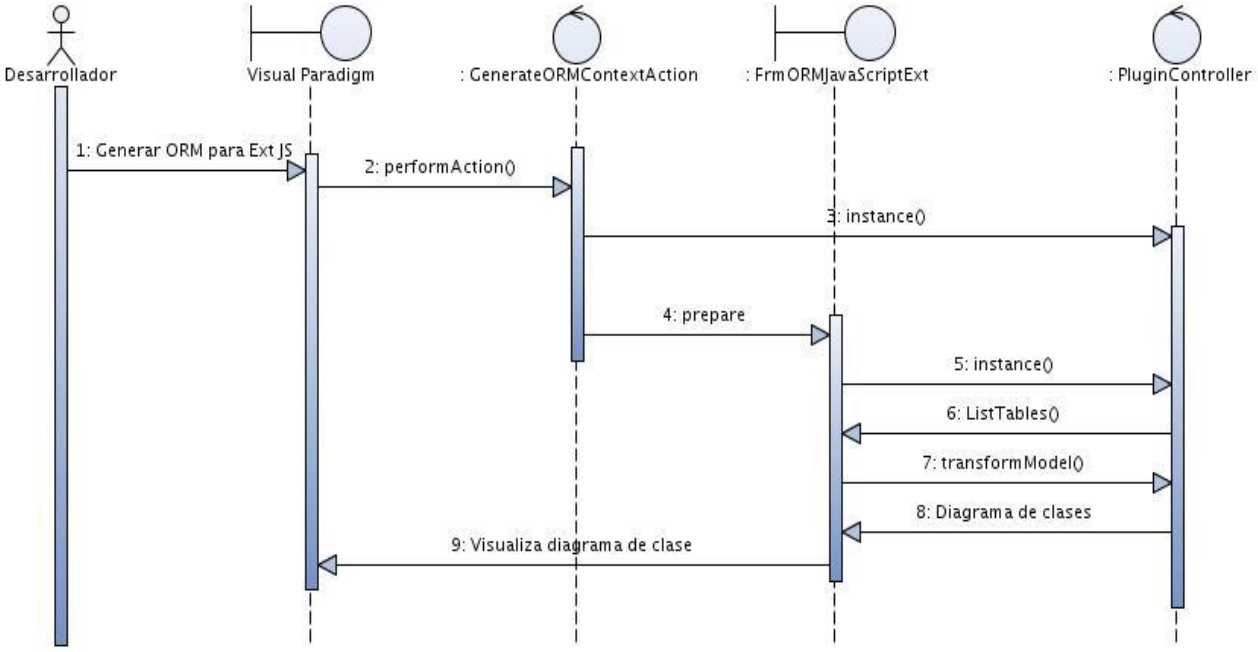


Figura # 19: Diagrama de Secuencia – Escenario: Generar Diagrama de Clases

Diagrama de Secuencia – Escenario: Generar Capa de Acceso a Dato.

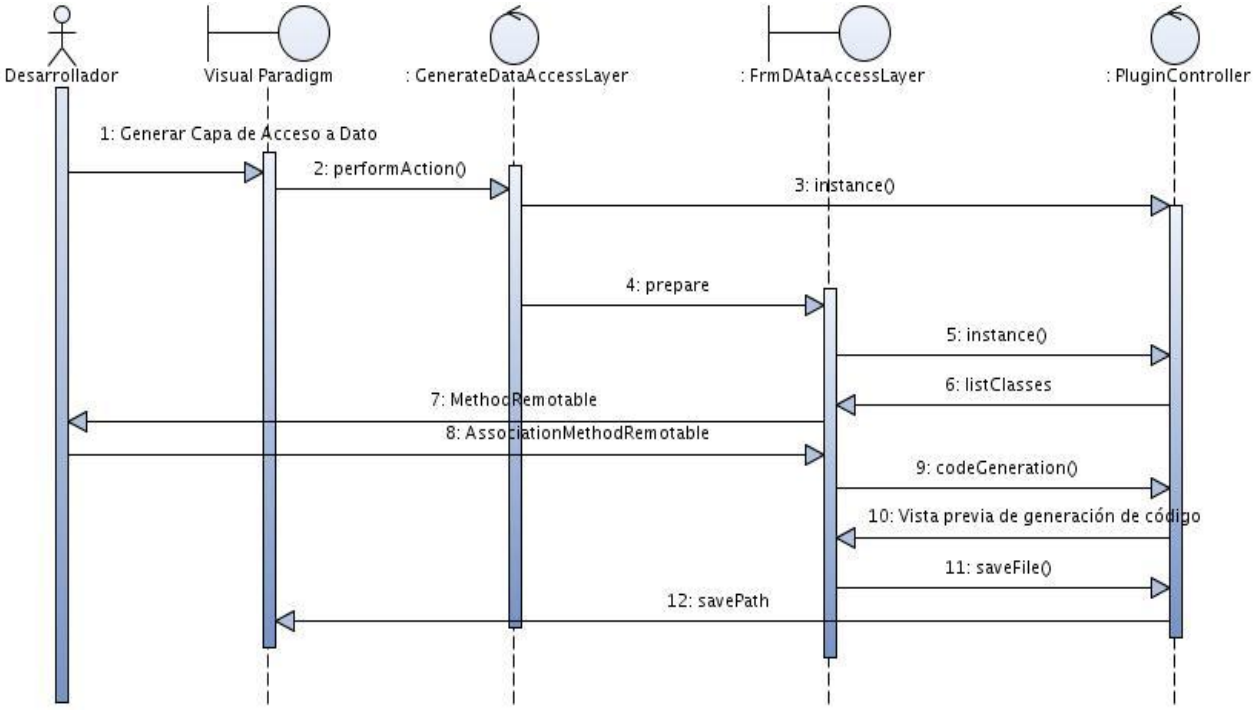


Figura # 20: Diagrama de Secuencia – Escenario: Generar Capa de Acceso a Dato

Conclusiones parciales.

En el presente capítulo se definieron los conceptos más importantes para el desarrollo de la aplicación, la especificación de los requisitos funcionales y no funcionales para el correcto funcionamiento del sistema. Se identificó, el actor, casos de usos y la relación existente entre ellos reflejada en el diagrama de casos de uso del sistema, se describieron detalladamente los casos de uso del sistema para obtener así un mayor entendimiento de los requisitos funcionales previamente establecidos. Se especifica la estructura del sistema, a través de los diagramas de clases del diseño y los diagramas de secuencia. Se describen, además las clases que ocupan un lugar relevante en el diseño, así como los patrones de diseño empleados para el proceso de desarrollo del *plugin*.

CAPÍTULO 3: Implementación y Pruebas del Plugin.

En el presente capítulo se muestra el modelo de implementación que pone en práctica el diseño de la solución que se va a realizar, así como el diagrama de componentes del *plugin*. Se describen las pruebas a realiza, con el objetivo de comprobar las funcionalidades del *plugin* en los diferentes escenarios, para de esta forma verificar en todos los casos que los resultados de las pruebas sean los esperados.

3.1 Modelo de Implementación

El modelo de implementación describe cómo los elementos de diseño se implementan en componentes. El modelo de implementación describe los componentes a construir y su organización en nodos físicos en los que funcionará el sistema. Está compuesto por los diagramas de despliegue y componentes. (21)

Un diagrama de despliegue muestra las relaciones físicas entre los componentes de hardware y software en el sistema final, es decir la configuración de los elementos de procesamiento en tiempo de ejecución y los componentes de software. Solo aquellos que son utilizados en tiempo de compilación deben mostrarse en el diagrama de componente por lo antes expuesto de define: realizar el modelo de implementación mediante el Diagrama de Componente, considerado aceptado para la descripción de los componentes del diseño ya que los componentes de un *plugin* son compilados por la herramienta *“Visual Paradigm for UML”* iniciada la aplicación.

3.1.1Diagrama de Componente

El diagrama de componentes es usado para estructurar el modelo de implementación, describe los elementos físicos del sistema y sus relaciones. Muestran las opciones de realización incluyendo código fuente, binario y ejecutable. Un componente representa un elemento físico que forma parte del sistema, se puede representar por nodos y sus operaciones solo se pueden alcanzar a través de interfaces. (22)

Existen diferentes tipos de componentes, como son:

- **Executable:** Especifica un componente que se puede ejecutar en un nodo.
- **Library:** Especifica una biblioteca de objetos estática o dinámica.
- **Table:** Especifica un componente que representa una tabla de una base de datos.

- **File:** Especifica un componente que representa un documento que contiene código fuente o datos.
- **Document:** Especifica un componente que representa un documento.

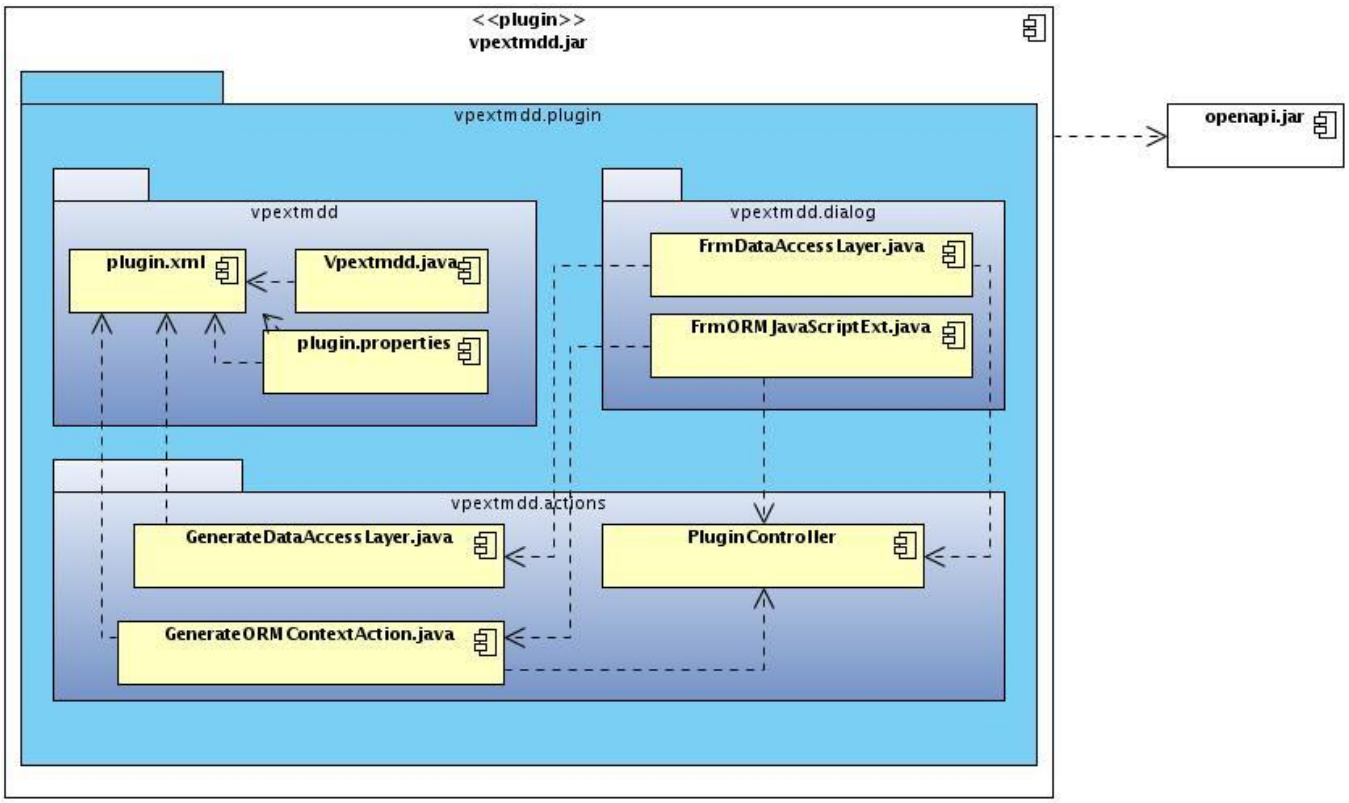


Figura # 21: Diagrama de Componentes

Descripción de los componentes más relevantes

El presente modelo de implementación describe cada uno de los componentes asociados al diseño de clases propuesto para la construcción de *plugin* para la herramienta de modelado *"Visual Paradigm for UML"*, así como la relación de dependencia entre los componentes que la integran.

Nombre del plugin: "*vpextmdd"* representa la aplicación el modelado en *"Visual Paradigm"* para el marco *Ext JS* con la aplicación de *Desarrollo Dirigido por Modelos*.

Paquete vpextmdd.plugin

Es el paquete raíz (nombre.plugin) de la estructura de un *plugin* en "*Visual Paradigm for UML"* contenedor de paquete asociado la configuración de *plugin*, acciones y diálogos presentes en la implementación.

Paquete vpextmdd

Agrupa las clases asociadas a la configuración del *plugin* estas son: plugin.xml, Vpextmdd.java y plugin.properties definidas en la estructura que propone la herramienta para la extensión.

Paquete vpextmdd.actions

Agrupa las clases referentes a las acciones tanto de contexto (GenerateORMContextAction.java), de herramientas (GenerateDataAccessLayer.java) y las clases controladoras de acciones (PluginController.java).

Paquete vpextmdd.dialog

Agrupa los formularios presentes en la aplicación como es el caso de FrmDataAccessLayer.java y FrmORMJavaScriptExt.java.

Plugin.xml

Su función consiste en la configuración del *plugin,* define el nombre del *plugin*, descripción, proveedor, librerías a utilizar y las acciones a ejecutar.

Vpextmdd.java

Su función está dirigida a la carga y descarga del *plugin* mediante la clase VPPluginInfo que provee el openapi.jar, capturando la información definida en el archivo **plugin.xml.** Vpextmdd.java implementa la interfaz VPPlugin definida en el openapi.jar la misma implementa los método **loaded()** y **unloaded()** para su función.

Plugin.properties

Su función es definir propiedades, asociados a los eventos y acciones del plugin.

GenerateORMContextAction.java

Es las clases referentes al control de las acciones de contexto la cual implementa la interfaz VPContextActionController definida por el openapi.jar para el manejo de las acciones a nivel de contexto haciendo uso del patrón *"Command".*

GenerateDataAccessLayer.java

Es la clase referente al control de las acciones a nivel de herramientas, la misma implementa la interfaz VPActionController definida por el openapi.jar para el manejo de acciones de herramientas haciendo uso del patrón *"Command".*

PluginController.java

Es la clase controladora de acciones que son relevantes en el *plugin*, directamente a la transformación de modelo y la captura de información de los mismos y el manejo de datos. En ella está contenido todo el método relevante para las funcionalidades principales del *plugin* como es la creación del diagrama de clases y la generación de código para Ext JS.

FrmORMJavaScriptExt.java

Representa el formulario asociado a generar el diagrama de clases para Ext JS en él se capturan los datos relevantes para la generación de diagrama de clase.

FrmDataAccessLayer.java

Representa el formulario asociado a la generación de la capa de acceso a dato y la vinculación de los servicios remotos del lado del servidor, así como la creación del proveedor de servicios que dará soporte a la conexión.

3.2 Pruebas de Software

La prueba del software es un elemento crítico para la garantía de la calidad del software. El objetivo de la etapa de pruebas es garantizar la calidad del producto desarrollado. Las pruebas son un proceso que se enfoca sobre la lógica interna del software y las funciones externas, es un proceso de ejecución de un programa con la intención de descubrir un error. Una prueba tiene éxito si descubre un error no detectado hasta entonces. Además, esta etapa implica:(23)

- Verificar la interacción de componentes.
- Verificar la integración adecuada de los componentes.
- Verificar que todos los requisitos se han implementado correctamente.
- Identificar y asegurar que los defectos encontrados se han corregido antes de entregar el software al cliente.

➢ **Nivel de Prueba**

Para comprobar que el *plugin* funcione de forma correcta, se realizan **Pruebas de Desarrollador** pues es el primer nivel de prueba y estas son diseñadas e implementada por el equipo de desarrollo.

- **Técnica de Prueba**

Como técnica de prueba se aplicarán las **Pruebas de Funcionalidad** que son pruebas que se realizan fijando su atención en la validación de las funciones, métodos, servicios y caso de uso. Se analiza cada funcionalidad implementada para verificar que se cumplan todos los requisitos establecidos y de esta forma satisfacer las necesidades existentes.

- **Tipo de prueba**

Dentro de la pruebas de funcionalidad se realizan las **Pruebas Funcionales** que tienen como objetivo asegurar el trabajo apropiado de los requisitos funcionales, incluyendo la navegación, entrada de datos, procesamiento y obtención de resultados, se enfoca en los requisitos funcionales y casos de uso. El método que utiliza este tipo de prueba es el método de Caja Negra.

- **Método de prueba**

El método de prueba a utilizar es el método de **Caja Negra** que consiste en las pruebas que se llevan a cabo sobre la interfaz del software. También conocidas como Pruebas de Comportamiento, estas pruebas se basan en la especificación del programa o componente a ser probado para elaborar los casos de prueba. Las pruebas de Caja Negra pretenden demostrar que las funciones del software son operativas, que la entrada se acepta de forma adecuada y que se produce un resultado correcto.

Para diseñar los casos de prueba de Caja Negra se utilizó la **Técnica de la Partición de Equivalencia** que divide el campo de entrada de un programa en variables de equivalencia con juegos de datos de entrada y salida. Las variables de equivalencia representan un conjunto de estados válidos y no válidos para las condiciones de entrada de un programa. Se definen dos tipos de variables de equivalencia, las *válidas*, que representan entradas válidas al programa, y las *no válidas*, que representan valores de entrada erróneos, aunque pueden existir valores no relevantes a los que no sea necesario proporcionar un valor real de dato.

3.2.1 Casos de prueba

Las pruebas de Caja Negra se aplicarán mediante los Casos de Prueba que constituyen un conjunto de entradas, condiciones de ejecución y resultados esperados desarrollados para cumplir un objetivo en particular o una función esperada.

En la siguiente tabla se muestran las variables V1, V2, Vn…, que representan valores de entrada de datos para los casos de pruebas. Ver (**Anexo 1**)

A continuación se presentan los casos de pruebas, para los casos de uso Generar Capa de Acceso a Datos y Generar Diagrama de Clases. No se le realizó al caso de uso Vincular Servicio porque el

método de prueba empleado, se realiza a través de la interfaz del programa y Vincular Servicio no presenta una interfaz gráfica, es un extendido de Generar Capa de Acceso a Datos. Estas pruebas se realizan a través de una matriz de datos, para comprobar que el *plugin* funcione de forma correcta, donde:

V: indica válido, I: indica inválido, NA: que no es necesario proporcionar un valor del dato en este caso, ya que es irrelevante. Ver (**Anexo 2 y 3**)

Después de realizar las pruebas de Caja Negra mediante los Casos de Prueba asociados a cada Caso de Uso, se comprobó el correcto funcionamiento del *plugin* y la correcta validación de los campos, verificando que solo se acepten los caracteres válidos para los mismos. Cada dificultad detectada en el desarrollo del *plugin* y resueltas a raíz del trabajo continuo de los desarrolladores, fueron recogidas en la planilla de No Conformidades. En la siguiente tabla se muestra un resumen de las dificultades encontradas:

PD: Pendiente **RA:** Resuelta

	Fecha	Versión	Caso de Prueba	Cant. de No Conformidad	Cant. de No Conformidad PD	Cant. De No Conformidad RA
1	0/05/2011	1.0	CU-Generar Diagrama de Clases	2	2	-
3	/06/2011	1.1	CU-Generar Diagrama de Clases	2	-	2

Tabla # 9: Resumen de los resultas de las pruebas aplicadas

Para más información referente a las dificultades encontradas en el proceso de desarrollo del *plugin*: ver **(Anexo 4).**

Conclusiones parciales

En este capítulo se realizó el modelo de implementación con el propósito de mostrar los componentes del sistema y sus relaciones, a través del diagrama de componentes. Además, se realizaron pruebas para validar que los requisitos fueron implementados correctamente a través del método de caja negra, aplicando la técnica de partición de equivalencia.

CONCLUSIONES

Como resultado del presente trabajo de diploma y para dar cumplimiento a los objetivos propuestos se puede concluir lo siguiente:

- ✓ Se analizaron los aspectos fundamentales sobre la extensión de la herramienta *"Visual Paradigm for UML"* y el soporte al Desarrollo Dirigido por Modelos con UML.
- ✓ Se definieron los elementos tecnológicos del marco Ext JS para la generación de código fuente a partir de los artefactos del modelo de diseño.
- ✓ Se realizó el análisis diseño e implementación de la extensión.
- ✓ Se probó el correcto funcionamiento de la herramienta, a través de las pruebas de caja negra realizadas mediante los casos de prueba.

RECOMENDACIONES

A partir de las experiencias obtenidas en el desarrollo del trabajo y con el fin de lograr un aprovechamiento óptimo del resultado alcanzado se recomienda:

- Actualizar el *plugin* a la versión 4.0 de Ext JS dado que el desarrollo del trabajo fue analizado para las ramas 2 y 3 las cuales incluyen la clase **Record** que deja de existir en la versión 4.0, siendo un elemento importante en el manejo de la solución.
- Incluir la detección de métodos remotos del lado del servidor, sin tener que definirlo por defecto, liberando de responsabilidad a los desarrolladores; asumiendo la automatización del proceso por la herramienta.
- Extender la implementación del *plugin* para generar el diseño de implementación del lado del servidor, asociando servicios remotos del lado del cliente.

REFERENCIAS

1. **La-Diversidad-de-Los-Objetos-1-1-Caracteristicas.** http://es.scribd.com/doc/25171397/1-La-Diversidad-de-Los-Objetos-1-1-Caracteristicas. [En línea]

2. **tutorial2.** http://www.adrformacion.com/cursos/metod5s/leccion1/tutorial2.html. [En línea]

3. **ModelosDeProcesoDeSoftware.** http://www.mitecnologico.com/Main/ModelosDeProcesoDeSoftware. [En línea]

4. **www.mastermagazine.info/termino/7006.php.** http://www.mastermagazine.info/termino/7006.php. [En línea]

5. *Perfiles UML y Desarrollo Dirigido por Modelos: Desafíos y Soluciones para Utilizar UML como Lenguaje de Modelado.* **Giachetti, Giovanni, Marín, Beatríz y Pastor, Oscar.** Valencia, España : SISTEDES, 2008. ISSN 1988–3455.

6. **L.Nahuel, L.Ocaranza, M.Pinasco.***Herramientas de soporte al proceso de desarrollo dirigido por modelos .* La Plata, Buenos Aires, Argentina : s.n.

7. **Modelos, Arquitectura de Software Dirigida por.***Arquitectura de Software Dirigida por Modelos.pdf.*

8. *MDA y el papel de los modelos en el proceso de desarrollo de software.* **Quintero, Juan Bernardo y Anaya, Raquel.** Número 8, Colombia, Medelli : s.n., Diciembre 2007. ISSN 1794-1237.

9. **Desarrollo de Software Dirigido por Modelos-manrubia Díez, Jorge y Cueva Lovelle, Juan Manuel.***Desarrollo de Software Dirigido por Modelos.* Diciembre 2006.

10. **1.5.0.1, OpenUP Version.** Ayuda de OpenUP Version 1.5.0.1. [En línea]

11. **visual-paradigm.** http://www.visual-paradigm.com/product/vpsuite/. [En línea]

12. **netbeans.org.** http://netbeans.org/index_es.html. [En línea]

13. **Ortiz, Kadir Hector.** http://www.eumed.net/libros/2009c/583/Representacion%20del%20Modelo%20de%20Objetos%20de%20Dominio.htm. [En línea]

14. **requirements.** http://elvex.ugr.es/idbis/db/docs/design/2-requirements.pdf. [En línea]

15. http://www.sparxsystems.com.ar/resources/tutorial/use_case_model.html. [En línea]

16. http://www.exa.unicen.edu.ar/catedras/modysim/teoria/casos_de_uso_a.pdf. [En línea]

17. *http://www.upedu.org/process/artifact/ar_desmd.htm.*

18. **Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides.***Design Pattens-Elements of Reusable Object Oriented Software.*

19. **Fowler, Martin.***Pattern of Enterprises Architecture.*

20. **Ingeniería, Facultad de Informática-Universidad Politécnica de Madrid-Unidad Doc de.***Patrones del "Grang of Four ".* España-Madrid : s.n.

21. **l.Jacobson, G.Booch, J.Rumbauggh.***El Proceso Unificado de Desarrollo.* s.l. : Addison Wesley, 2000.

22. **Daniele, Marcela.***Teoría 11: El Arte de Modelar UML.* 2007.

23. **TiposPruebasSoftware.**
http://www.cetic.guerrero.gob.mx/pics/art/articles/113/file.TiposPruebasSoftware.pdf. [En 27.

BIBLIOGRAFÍA

1. **adictosaltrabajo.** [En línea] http://www.adictosaltrabajo.com/tutoriales/tutoriales.php.

2. **artifact.** [En línea] http://www.upedu.org/process/artifact/ar_desmd.htm.

3. *Arquitectura de Software Dirigida por Modelos.pdf.*

4. **Modelos, Arquitectura de Software Dirigida por.** *Arquitectura de Software Dirigida por Modelos.pdf.*

5. **1.5.0.1, OpenUP Version.** Ayuda de OpenUP Version 1.5.0.1. [En línea]

6. Conferencia #7.Disciplina de Prueba. Ingeniería de Software II. *Conferencia #7.Disciplina de Prueba. Ingeniería de Software II.* [En línea]

7. **Clases_Teoricas_.** [En línea] http://www.fi.unju.edu.ar/materias/materia/IS1/document/Clases_Teoricas_2010/Introduccion-2010.pdf.

8. **Desarrollo de Software Dirigido por Modelos-manrubia Díez, Jorge y Cueva Lovelle, Juan Manuel.** *Desarrollo de Software Dirigido por Modelos.* Diciembre 2006.

9. *Desarrollo de Software Dirigido por Modelos.* **manrubia Díez, Jorge y Cueva Lovelle, Juan Manuel .** Diciembre 2006, Vol. Memoria Segundo Semestre. FICYT.

10. *Desarrollo de Software Dirigido por Modelos, conceptos teóricos y su aplicación práctica.* **Pons, Dra. Claudia y Giandini, Dra. Roxana .** 2007.

11. **Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides.** *Design Pattens-Elements of Reusable Object Oriented Software.*

12. **Jaque, Miguel.** [En línea] Diciembre del 2007. http://migueljaque.com/index.php/tecnicas/tecnicasmodnegocio/37-modelado_negocio/46-modelo-de-dominio.

13. **l.Jacobson, G.Booch, J.Rumbauggh.** *El Proceso Unificado de Desarrollo.* s.l. : Addison Wesley, 2000.

14. **UCI.** Entorno Virtual de Aprendizaje curso de Historia de la Informatica. [En línea] http://eva.uci.cu/mod/resource/view.php?id=9287&subdir=/Temas_Generales.

15. **Orellanos, Norah Velazco.** *Estadísticas de Inversión Extranjera Directa en los Países de la Comunidad Andina.*

16. **Eric Freeman, Elisabeth Freeman.** *Head First Design Patterns.*

17. **L.Nahuel, L.Ocaranza, M.Pinasco.** *Herramientas de soporte al proceso de desarrollo dirigido por modelos.* La Plata, Buenos Aires, Argentina : s.n.

18. **La-Diversidad-de-Los-Objetos.** [En línea] http://es.scribd.com/doc/25171397/1-La-Diversidad-de-Los-Objetos-1-1-Caracteristicas.

19. **mastermagazine.** [En línea] http://www.mastermagazine.info/termino/7006.php.

20. **MDA.** [En línea] http://es.scribd.com/doc/47014355/MDA.

21. *MDA y el papel de los modelos en el proceso de desarrolo de software.* **Quintero, Juan Bernardo y Anaya, Raquel.** Colombia, Medelli : s.n : ISSN 1794-1237., Diciembre 2007. Número 8.

22. **Systems, Por Popkin Software and.** *Modelado de Sistemas com UML.*

23. **ModelosDeProcesoDeSoftware.** [En línea] http://www.mitecnologico.com/Main/ModelosDeProcesoDeSoftware.

24. **netbeans.org.** [En línea] http://netbeans.org/index_es.html.

25. **Ortiz, Kadir Hector.** [En línea] http://www.eumed.net/libros/2009c/583/Representacion%20del%20Modelo%20de%20Objetos%20de%20Dominio.htm.

26. **Ingeniería, Facultad de Informática-Universidad Politécnica de Madrid-Unidad Doc de.** *Patrones del "Grang of Four ".* España-Madrid : s.n.

27. **Fowler, Martin.** *Pattern of Enterprises Architecture.*

28. *Perfiles UML y Desarrollo Dirigido por Modelos: Desafíos y Soluciones para Utilizar UML como Lenguaje de Modelado.* **Giachetti, Giovanni, Marín, Beatríz y Pastor, Oscar.** Valencia, España : SISTEDES, 2008. ISSN 1988–3455.

29. **Marciniak, J.J.** *Process Models in Software Engineering.* s.l. : 2nd Edition, John Wiley and Sons, New York, December 2001.

30. **Pruebas_Funcionales.** [En línea] http://carolina.terna.net/ingsw3/datos/Pruebas_Funcionales.pdf..

31. **PruebaCasoDePrueba.** [En línea] http://www.mitecnologico.com/Main/PruebaCasoDePruebaDefectoFallaErrorVerificacionValidacion.

32. **pruebas, tipos de.** [En línea] http://www.cetic.guerrero.gob.mx/pics/art/articles/113/file.TiposPruebasSoftware.pdf.

33. **Rational-Rose.** [En línea] http://searchcio-midmarket.techtarget.com/definition/Rational-Rose.

34. **requirements.** [En línea] http://elvex.ugr.es/idbis/db/docs/design/2-requirements.pdf.

35. Secretaria de Economia . [En línea] http://www.economia.gob.mx/swb/work/models/economia/Resource/516/1/images/EstadInverMexicoUE.pdf.

36. **sencha.** [En línea] http://www.sencha.com/products/extjs/.

37. **sencha-Direct.** [En línea] http://www.sencha.com/products/extjs/direct.php.

38. **slideshare.** [En línea] http://www.slideshare.net/guest83f0d26/mda-2596889.

39. **Daniele, Marcela.** *Teoría 11: El Arte de Modelar UML.* 2007.

40. **Marleysis López Duque, Raidel Oceguera Ravelo.** *Tesis: Herramienta en Matlab para la obtención de información de la base de datos y ficheros electroencefalograma del proyecto Mapeo Cerebral Humano Cubano.* La Habana, Cuba : s.n., Junio, 2010.

41. **Heritage, The American.** *The American Heritage Science Dictionary.*

42. **TiposPruebasSoftware.** [En línea] http://www.cetic.guerrero.gob.mx/pics/art/articles/113/file.TiposPruebasSoftware.pdf.

43. **Linet Lores Sánchez, Diana Monné Roque.** *Trabajo de Diploma: Aplicación de las pruebas de liberación al Sistema Informático de Genética Médica (Junio 2009).* junio 2009.

44. **tutorial2.** [En línea] http://www.adrformacion.com/cursos/metod5s/leccion1/tutorial2.html.

45. **tutoriales-basedat1.** [En línea] http://sistemas.itlp.edu.mx/tutoriales/basedat1/tema2_5.

46. **Larman, Craing.** *UML y patrones.*

47. **Visio.** [En línea] http://encyclopedia2.thefreedictionary.com/Visio.

48. **visual-paradigm.** [En línea] http://www.visual-paradigm.com/product/vpsuite/.