Universidad de las Ciencias Informáticas
Facultad 6

Título: Extensión para modelar soluciones de almacenes de datos en la herramienta de modelado “Visual Paradigm for UML”.

Trabajo de Diploma para optar por el título de Ingeniero en Ciencias Informáticas

Autores: Sergio Martin Torres
Alain Suárez Suárez

Tutores: Ing. Yanisbel González Hernández
Ing. Elio Luis Toledo García

La Habana 11/6/2012

“Año 54 de la Revolución”

DECLARACIÓN DE AUTORÍA

Declaramos ser autores del presente trabajo de diploma y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

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

Sergio Martin Torres

_____
Firma del Autor

Alain Suárez Suárez

_____
Firma del Autor

Yanisbel González Hernández

_____
Firma del Tutor

Elio Luis Toledo García

_____
Firma del Tutor

DATOS DE CONTACTO

Tutor: Ing. Yanisbel González Hernández.
Categoría Docente: Instructor.
Ingeniera Informática, Universidad de Ciencias Informáticas, 2006.
Correo Electrónico: yglezh@uci.cu

Co-Tutor: Ing. Elio Luis Toledo García.
Ingeniero Informático, Universidad de Ciencias Informáticas, 2011.
Correo Electrónico: eltoledo@uci.cu

Autor: Sergio Martin Torres
Universidad de las Ciencias Informáticas
La Habana, Cuba
Correo Electrónico: smartin@estudiantes.uci.cu

Autor: Alain Suárez Suárez
Universidad de las Ciencias Informáticas
La Habana, Cuba
Correo Electrónico: asuarezs@estudiantes.uci.cu

AGRADECIMIENTOS

A mis padres por estar siempre en todo momento brindándome su apoyo incondicional.

A mi familia por confiar siempre en mí y darme la oportunidad de convertirme en un profesional.

A Adriana por estar siempre dispuesta a escucharme y aconsejarme en todo momento.

A todas las personas y profesores que pusieron su granito de arena en mi formación como profesional.

A mis compañeros de estudios, en especial a Andry y Osvel que hasta el momento son mis ejemplos a seguir como profesional,

A todos ustedes, gracias.

Sergio

A mi madre que siempre ha estado conmigo y que es mi guía y ejemplo.

A mi tata Adair y a mi hija que son todo para mí.

A mi papá y abuelos que han dado todo para que sea quien soy hoy.

A mi novia y su familia por su apoyo.

Alain

DEDICATORIA

Primero agradecerles a mis padres por brindarme siempre su apoyo, indistintamente de sus diferencias. Alentándome constantemente a continuar mis estudios, dándome la oportunidad hasta el día de hoy de convertirme en un profesional, gracias a ustedes, hoy soy mejor persona.

A toda mi familia, en especial a mis abuelos, Lelo y Lela, que siempre se preocuparon y dieron lo mejor de sí por mi desarrollo como estudiante, ya que ellos nunca tuvieron la oportunidad que yo tuve ahora.

A mi novia Adriana por aconsejarme, apoyarme, escucharme y ayudarme en todo lo que necesité.

A mis compañeras y compañeros de estudio.

A todas las personas que confiaron en mí y me apoyaron durante estos cinco años.

Sergio

Le dedico esta tesis principalmente a mi madre que dio su vida por mí y que siempre me ha acompañado y ayudado.

A mi tata Adair que es mi vida y que siempre me apoya y apoyará incondicionalmente.

A mi pelusita para que siga el ejemplo de su padre.

A mi padre, abuelos y toda mi familia por ayudarme y guiarme para llegar a ser quien soy hoy, a mi hermano Vitali.

A mi novia Melissa, mis suegros Gricell y Pancho y a la Tía Gladys por su gran apoyo.

A todos los profesores que tuvieron paciencia conmigo y me enseñaron todo lo que sé.

Alain

RESUMEN

En este mundo moderno e interconectado de hoy, peor que no tener información disponible es tener abundante información y no saber qué hacer con ella. Para aprovechar una cantidad grande de información surge el concepto de Inteligencia de Negocios o BI (Business Intelligence). La BI puede generar escenarios, pronósticos y reportes que apoyen a la toma de decisiones, lo que se traduce en una ventaja competitiva. Las bases de datos tradicionales están diseñadas a partir de modelos Entidad-Relación, de ahí que se denominen bases de datos relacionales, estas al ser operacionales no están diseñadas para el trabajo con grandes cantidades de datos de más de cinco años de antigüedad. Para dar solución a la problemática que presentan las bases de datos relacionales se crean los almacenes de datos (AD), para el diseño de estos se necesita su modelado con anterioridad. En la UCI la herramienta informática que se utiliza para el modelado es Visual Paradigm for UML (VP) dado que tiene pagada su licencia, pero presenta el problema de no permitir el modelado de AD. La presente tesis persigue dar solución al problema realizando una extensión a VP que le permita modelar AD. Para realizar la extensión se propone un Perfil UML que respeta las reglas de diseño de un almacén, a su vez se realiza un estudio acerca de las tendencias presentadas por autores reconocidos e instituciones existentes, acerca de UML (Unified Modeling Language o Lenguaje Unificado de Modelado) y sus mecanismos de extensión, en aras de lograr una propuesta sustentada en fundamentos teóricos de la actualidad.

PALABRAS CLAVE: Almacenes de Datos, BI, Inteligencia de Negocio, Modelo dimensional, Perfil UML, UML.

TABLA DE CONTENIDO

Índice de figuras

Índice de tablas

INTRODUCCIÓN

La humanidad desde comienzos de la primera mitad del siglo XX hasta llegar a los albores del siglo XXI ha continuado realizando descubrimientos de nuevas tecnologías, mediante la adquisición y profundización de los nuevos conocimientos, impulsados siempre por el deseo de satisfacer el afán de saber. Los avances tecnológicos han afectado de forma regular y drástica el modo en que se desarrolla la vida actual en la sociedad, pues han ofrecido la posibilidad de extender el campo de la experimentación y adquisición de conocimientos mediante el desarrollo de tecnologías innovadoras.

La informática es parte de estas tecnologías, y tiene un papel importante y esencial en el grupo de ciencias que conforman la base para las nuevas Tecnologías de la Informática y las Comunicaciones (TIC). La ciencia informática estudia los fenómenos relacionados con los objetos de su dominio material y cuenta con un conjunto de métodos, técnicas y procedimientos[1] que le permiten captar y estudiar los fenómenos relacionados con el tratamiento sistemático de la información [1]. La informática agrupa todo aquello que tiene relación con el procesamiento de datos, utilizando la computadora y equipos de proceso automatizado de la información[2]. Cuba en los últimos años ha venido experimentado el auge de la informática, debido en parte al apoyo priorizado que brindó el consejo de estado a la fundación de la Universidad de la Ciencias Informáticas (UCI), la cual ocupa un papel cimero en el desarrollo de la industria cubana del software. Su modelo de producción ha ido evolucionando hasta llegar a un modelo orientado a Centros Productivos vinculados estrechamente con las facultades y áreas temáticas especializadas. Dichos centros vinculan la producción, la investigación y la formación de estudiantes como futuros profesionales altamente calificados.

Entre estos centros se encuentra el Centro de Tecnologías de Gestión de datos (DATEC), el cual tiene como misión proveer soluciones integrales así como consultorías relacionadas con tecnologías de bases de datos y el análisis de información. El centro también estudia 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.

DATEC está compuesto por cuatro líneas de producción de productos de software: PostgreSQL,

[1] **Procedimientos**: modelación, abstracción

Bioinformática, Integración de Soluciones y Almacenes de Datos. En cada una de las líneas se hace uso de un conjunto de herramientas informáticas con el objetivo de ayudar y agilizar el proceso de desarrollo de software. Una de estas herramientas es “Visual Paradigm for UML” (VP) la cual brinda la posibilidad de crear modelos UML para guiar el proceso de desarrollo de software. El Departamento de Almacenes de Datos utiliza dicha herramienta para el diseño de los almacenes y mercados de datos de sus proyectos productivos. El departamento “entre sus políticas de trabajo” establece la utilización de herramientas libres para el desarrollo de sus productos como parte de las normas establecidas por la dirección del país, con el fin de lograr en un breve tiempo la soberanía tecnológica en cuanto al desarrollo de productos informáticos. Para el proceso de modelado de la solución en esta entidad se dificulta la utilización de herramientas libres pues las principales herramientas especializadas en este proceso son propietarias y las existentes no cuentan con la funcionalidad de modelar estructuras dimensionales. Se han utilizado algunas alternativas pero no resuelven todas las necesidades del diseño pues consumen tiempo y no agilizan la obtención de la solución haciendo engorroso el proceso de desarrollo. Además después de modelar la estructura dimensional todavía el diseño debe pasar por el área de Inteligencia del Negocio (BI) para la elaboración del cubo dimensional.

Por lo anteriormente planteado se define como ***problema de la investigación*****:** ¿Cómo modelar estructuras de datos dimensionales con la herramienta “Visual Paradigm for UML”?

La investigación tiene como ***objeto de estudio*****:** Lenguajes de modelado y metamodelos, enmarcado en el ***campo de acción*****:** Desarrollo de extensiones para la herramienta “Visual Paradigm for UML” para modelar estructuras de datos dimensionales.

Para dar solución a la situación planteada anteriormente, la investigación tiene como ***objetivo general***: Desarrollar una extensión de la herramienta “Visual Paradigm for UML” para modelar estructuras de datos dimensionales, el cual se desglosa en los siguientes ***objetivos específicos*****:**

- ✓ Realizar un estudio del estado del arte de los conceptos, metodologías y herramientas para el desarrollo de la investigación.
- ✓ Realizar el análisis de la extensión para modelar soluciones de almacenes de datos (AD).
- ✓ Realizar el diseño de la extensión para modelar soluciones de AD.
- ✓ Realizar la implementación y pruebas de la extensión para modelar soluciones de AD.

A los cuales se les dará cumplimiento a través de las siguientes ***tareas de la investigación*****:**

- ✓ Revisión bibliográfica de la documentación técnica de *"Visual Paradigm for UML"* referida a la extensión de la herramienta.
- ✓ Análisis y selección de las metodologías, herramientas y tecnologías a utilizar en el desarrollo de la extensión para modelar soluciones de AD.
- ✓ Análisis y diseño de las estructuras de datos dimensionales para modelar soluciones de AD.
- ✓ Definición de los elementos para desarrollar estructuras de datos dimensionales en *"Visual Paradigm for UML"*.
- ✓ Implementación de la extensión para modelar soluciones de AD.
- ✓ Realizar pruebas a la extensión para modelar soluciones de AD.

Estructura de la tesis

El presente trabajo de diploma está estructurado de la siguiente forma: introducción, tres capítulos, conclusiones, recomendaciones, referencias bibliográficas, bibliografía, anexos y glosario de términos.

Capítulo 1: Fundamento Teórico. En este capítulo se abordarán los principales conceptos que serán de utilidad para el dominio de la presente investigación, así como la realización del estudio enfocado al Perfil UML y CWM (Common Warehouse Metamodel o Metamodelo Común de Almacén) que deberán ser soportados por la extensión. Se seleccionarán las herramientas, lenguaje de programación y metodología para el desarrollo de la investigación.

Capítulo 2: Análisis y diseño de la extensión para modelar soluciones de AD. En este capítulo se describe la propuesta de solución para la situación problemática anteriormente planteada. Se definirán las principales características que poseerá la aplicación informática a desarrollar, comenzando con los elementos arquitectónicos a tener en cuenta para la implementación de extensiones en VP, y el estudio de la planificación, la definición de los requisitos funcionales y no funcionales y el diseño. Además se generan todos los artefactos de estas fases durante el ciclo de vida del proceso de desarrollo de software seleccionado.

Capítulo 3: Implementación y pruebas de la extensión para modelar soluciones de AD. En este capítulo se realizará la representación del código fuente de las principales funcionalidades codificadas para la extensión, atendiendo a estándares de codificación ya creados. Se obtendrán resultados del diseño implementando la extensión informática y se garantizara la calidad del software mediante la realización de pruebas de aceptación.

1. CAPÍTULO 1: FUNDAMENTO TEÓRICO

1.1Introducción

En este capítulo se abordarán los principales conceptos que serán de utilidad para el dominio de la presente investigación, así como se realizará el estudio enfocado al Perfil UML y CWM que deberán ser soportados por la extensión. Además se seleccionarán las herramientas, el lenguaje de programación y la metodología para el desarrollo de la investigación.

1.2 Fundamentos Teóricos para el desarrollo de la extensión

1.2.1 Modelado

En el amplio campo que conforma la ciencia, existen una gran variedad de modelos, concretamente en las ciencias puras y aplicadas, las cuales se relacionan mediante el uso de 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[3].

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 orientado a objetos. 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, que sea lo bastante sencillo como para poder ser manipulado y estudiado[3].

Una de estas ramas comprende a la Ingeniera de Software principalmente en el proceso de desarrollo del software, debido a las nuevas exigencias del mercado y de los clientes. Ante estas exigencias, se han debido plantear nuevos modelos organizativos o analizar los antiguos buscando alternativas más eficaces que permitan aprovechar mejor todos los recursos disponibles en las empresas. Proveer modelos facilita la comunicación tanto para clientes como para especialistas, en la elaboración de un proyecto de software[4].

En la actualidad existe una asociación formada por grandes corporaciones, cuya mayoría constituyen parte de la industria del software, como IBM (International Business Machines), Apple, Sun Microsystems, Microsoft y HP (Hewlett-Packard). Dicha asociación responde a las siglas de OMG[2] y en su haber constan ciertos productos que siguen la ideología o teoría científica del uso de modelos.

[2] OMG Object Management Group

1.2.2 OMG y estándares de modelado

No se puede hablar del Lenguaje Unificado de Modelado (UML[3]) y Metamodelo Común de Almacén[4] (CWM) sin antes referirnos a la OMG. La OMG es un grupo sin ánimos de lucro que tiene como misión desarrollar estándares, dentro de sus creaciones se encuentran los de modelado como son UML y CWM. OMG también está dedicada a promover tecnología de negocio y optimización para la innovación a través de su Iniciativa de Ecología de Negocio.

En la informática, como en las demás ramas de la ingeniería se necesita organizar, planificar y diseñar el trabajo antes de ejecutarlo, para así reducir las probabilidades de error en la ejecución del mismo. Los diseños se entienden mejor si se representan gráficamente, por lo que surgió la necesidad de representar los diseños del software que se fueran a desarrollar. Para poder estandarizar estos diseños y que fueran entendibles para todos, surge UML, dando así, paso a una de las premisas de la ingeniería del software, la reutilización[5].

1.2.3 Unified Modeling Language (UML)

El lenguaje UML comenzó a gestarse en octubre de 1994, cuando Rumbaugh se unió a la compañía Rational fundada por Booch (dos grandes investigadores en el área de metodología del software). El objetivo de ambos era unificar dos métodos que habían desarrollado: el método Booch y el OMT (Object Modelling Tool). El primer borrador apareció en octubre de 1995. En esa misma época otro gran investigador, Jacobson, se unió a Rational y se incluyeron ideas suyas. Estas tres personas son conocidas como los “tres amigos”. Además, este lenguaje se abrió a la colaboración de otras empresas para que aportaran sus ideas. Todas estas colaboraciones condujeron a la definición de la primera versión de UML[6].

[3] UML de sus siglas en ingles que significan Unified Modeling Language
[4] MCA o CWM de sus siglas en ingles que significan Common Warehouse Meta-model

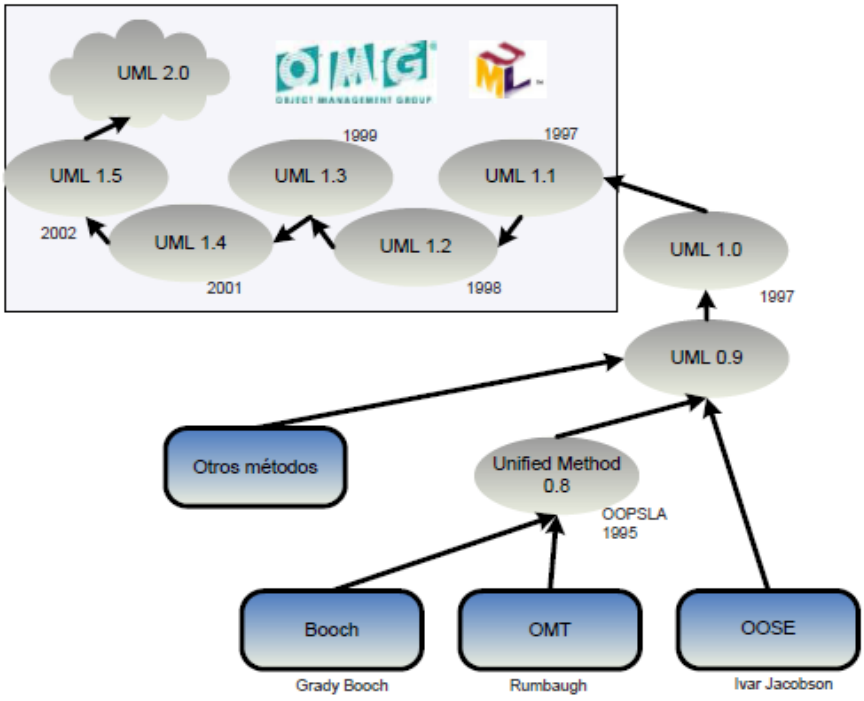

Figura # 1 Proceso evolutivo de UML

UML es además un método formal de modelado, lo que aporta las siguientes ventajas:

• Mayor rigor en la especificación.

• Permite realizar una verificación y validación del modelo realizado.

• Se pueden automatizar determinados procesos y permite generar códigos a partir de los modelos y a la inversa (a partir del código fuente generar los modelos). Esto permite que el modelo y el código estén actualizados, con lo que siempre se puede mantener la visión en el diseño, de más alto nivel, de la estructura de un proyecto[6].

UML es ante todo un lenguaje. Un lenguaje proporciona un vocabulario y reglas para permitir una comunicación. En este caso, este lenguaje se centra en la representación gráfica de un sistema.

Tal como indica su nombre, UML es un lenguaje de modelado. Un modelo es una simplificación de la realidad. El objetivo del modelado de un sistema es capturar sus partes más esenciales. Para facilitar este modelado, se realiza una abstracción y se plasma en una notación gráfica. Esto se conoce como modelado visual. Los objetivos de UML son muchos, pero se pueden sintetizar sus funciones:

- Visualizar: UML permite expresar de una forma gráfica un sistema de forma que otro lo puede entender.
- Especificar: UML permite especificar cuáles son las características de un sistema antes de su construcción.
- Construir: A partir de los modelos especificados se pueden construir los sistemas diseñados.
- Documentar: Los propios elementos gráficos sirven como documentación del sistema desarrollado que pueden servir para su futura revisión[6].

Un modelo UML está compuesto por tres clases de bloques de construcción:

- Elementos: son abstracciones de cosas reales o ficticias (objetos, acciones, etc.)
- Relaciones: relacionan los elementos entre sí.
- Diagramas: Son colecciones de elementos con sus relaciones[6].

Los conceptos de modelado de UML están agrupados en unidades de lenguaje (Language Unit). Una unidad de lenguaje consiste en una colección de conceptos fuertemente emparejados que proveen a los usuarios con el poder de representar aspectos del sistema a estudiar de acuerdo a un paradigma en particular. Desde la perspectiva del usuario esta partición de UML significa que el usuario solamente necesita preocuparse por aquellas partes que considere necesarias para el modelado. Por tanto un usuario de UML no necesita saber el lenguaje completo para usar UML efectivamente[7].

Esto ha sido posible en parte, gracias al Desarrollo Dirigido por Modelos o Model-Driven Development en inglés (MDD) que a lo largo de estos años se ha incorporado como una nueva área dentro del campo de la Ingeniería de Software.

1.2.4 Model-Driven Development (MDD)

El Desarrollo Dirigido por Modelos 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 considerado 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. Tiene como objetivo elevar el nivel de abstracción en el desarrollo de software dándole una mayor importancia al modelado conceptual y al papel de los modelos en el desarrollo de software actual. Por ello aparecen 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. Su idea fundamental es sustituir al código de lenguajes de programación específicos por modelos. De este modo 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.[8]

MDD se sintetiza como la combinación de tres ideas complementarias:

- 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.
- 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.
- 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[9].

Muchas de las propuestas basadas en MDD definen su propio lenguaje de modelado que se conocen como Lenguaje de Modelado Específico de Dominios (LMED)[10]. 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 como el Perfil UML que incorporan la precisión semántica de un LMED, convirtiendo UML en un LMED. Contar con una propuesta de MDD que dé 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 herramientas de software que dan soporte a UML[10].

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 UML. Las nuevas características permiten definir la precisión semántica requerida para las diferentes propuestas de

MDD, provocando un incremento considerable de Perfiles UML (UML Profiles) en los últimos años, pero CWM no pudo concebirse mediante el uso de perfiles.

1.2.5 Common Warehouse Metamodel (CWM)

CWM reutiliza conceptos de UML y tiene como base un modelo de objetos basado en una versión de metamodelo de UML en el cual los aspectos que no son relevantes en un AD, (definido por Bill Inmon: como una colección de datos orientados a temas específicos o ámbitos, ya sea empresa u organización debe ser integrado, no volátil y variable en tiempo, cuyo principal objetivo es ayudar a la toma de decisiones[11]) han sido removidos. Muchos de los tipos de objetos y asociaciones del núcleo de UML están reflejados en CWM. Donde sea apropiado, los tipos de modelo de objeto son divididos en subtipos para proveer tipos más específicos en CWM, normalmente con atributos adicionales o asociaciones adicionales. Todos los tipos de objetos de CWM son directa o indirectamente subtipos de tipos de modelos de objetos apropiados, y heredan sus atributos y asociaciones. Este acercamiento tiene varias ventajas, permite a la especificación de CWM centrarse en la inversión sustancial del desarrollo y refinación del metamodelo de UML. El dominio general de UML debe ayudar a la comprensión de las especificaciones de CWM y su modelo de objetos base. También posibilita la fácil inclusión de los modelos UML como parte de los metadatos del almacén de datos. CWM se divide en un conjunto de paquetes que ofrecen modularidad, lo cual a su vez permite la comprensión del metamodelo dividiéndolo en unidades pequeñas, permitiendo a los usuarios e implementadores ignorar los paquetes que no son relevantes para ellos[12].

1.3 Mecanismos de extensión

Para modelar lenguajes específicos de dominio (AD) OMG define dos posibilidades, o bien se define un nuevo lenguaje (alternativa de UML), o bien se extiende el propio UML, especializando algunos de sus conceptos y restringiendo otros, pero respetando la semántica original de los elementos de UML (clases, asociaciones, atributos, operaciones, transiciones, etc.). La primera opción es la adoptada por lenguajes como CWM, puesto que la semántica de algunos de sus constructores no coincide con la de UML. Para definir un nuevo lenguaje, sólo hay que describir su metamodelo utilizando el MOF[5], un lenguaje para describir lenguajes de modelado. Por otro lado, hay situaciones en las que es suficiente con extender el lenguaje UML utilizando una serie de mecanismos recogidos en lo que se denominan Perfiles UML.

[5] MOF: Meta Objet Facility

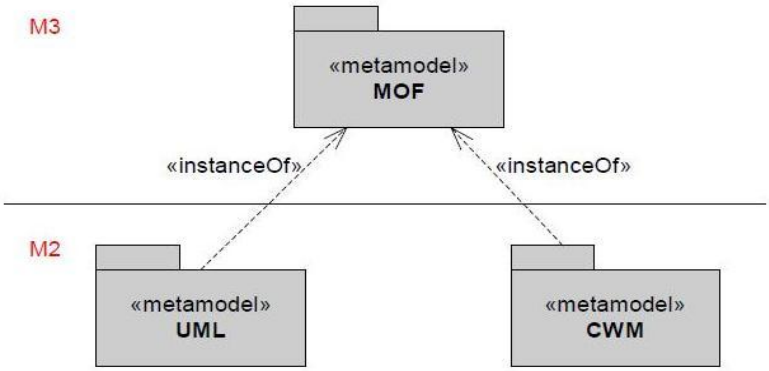

Figura # 2 Meta nivel en que se encuentran UML y CWM.

VP no comprende el modelado de una estructura de datos dimensional, al no estar dichas estructuras en el dominio de UML. Por lo que se necesitaría buscar la forma de poder modelar estructuras dimensionales.

Viéndose lo anteriormente descrito, se podría preguntar: Perfil UML o CWM? Para dar respuesta a esta interrogante se debe saber qué brinda un perfil o un metamodelo. Se debe conocer que tanto los perfiles como los metamodelos son mecanismos para adaptar un lenguaje de modelado a un dominio o situación específica.

Un metamodelo define conceptos y sus relaciones gracias al diagrama de clases (clase, atributos, asociaciones), define solamente la estructura, pero no la semántica.

Un modelo es una instancia de un metamodelo sí este respeta la estructura definida por el metamodelo. Un Perfil UML señala varias razones por las que un diseñador puede querer extender y adaptar un metamodelo existente, sea el del propio UML, el de CWM, o incluso el de otro perfil[13]:

- ✓ Disponer de una terminología y vocabulario propio de un dominio de aplicación o de una plataforma de implementación concreta.
- ✓ Definir una sintaxis para construcciones que no cuentan con una notación propia.
- ✓ Definir una nueva notación para símbolos ya existentes, más acorde con el dominio de la aplicación objetivo.
- ✓ Añadir cierta semántica que no aparece determinada de forma precisa en el metamodelo.
- ✓ Añadir cierta semántica que no existe en el metamodelo.
- ✓ Añadir restricciones a las existentes en el metamodelo, restringiendo su forma de utilización.

- ✓ Añadir información que puede ser útil a la hora de transformar el modelo a otros modelos, o a código.

Dado que VP usa el lenguaje de modelado UML y que CWM no es 100% compatible con el (la semántica de algunos de los constructores de CWM no coincide con la de UML), se consideró que se utilizará Perfiles UML como método de extensión de UML.

1.4 Estudio de herramientas de modelado que permiten el modelado dimensional.

1.4.1 ER/Studio

El ER/Studio es una herramienta de modelado de datos, que ha ganado un lugar a nivel internacional, puesto que ha alcanzado el interés de las principales empresas y compañías en el mundo, por poseer un soporte completo a las bases de datos, permitiéndole a las empresas descubrir, documentar y reutilizar sus activos de datos. ER/Studio se caracteriza por:

- Permitir a los arquitectos realizar ingeniería inversa, analizar y optimizar bases de datos existentes.
- Mejorar la consistencia de los datos.
- Trazar los orígenes de los datos y mejora la integración y exactitud.
- Comunicar eficientemente los modelos en la empresa.
- Posibilitar utilizar el modelado para producir esquemas XML.
- Generar código fuente para algunos diseños de base de datos.
- Construir modelos gráficos desde una base de datos o esquema existente.
- Poseer un ambiente completo para el análisis, diseño, creación, y mantenimiento de aplicaciones de base de datos[14].

1.4.2 IBM InfoSphere Data Architect

Es una solución de colaboración de diseño, que ayuda a descubrir, modelar, relacionar y estandarizar activos de datos diversos y distribuidos, sobre la base de las capacidades de modelado de datos completos de versiones anteriores.

Posee los siguientes requisitos:

- Diseñar modelos lógicos, físicos y dimensionales.
- Reducir el tiempo de desarrollo de sistemas de inteligencia de negocio.
- Permitir una rápida consulta para acelerar la presentación de informes de negocio.

Entre sus características se destaca:

- El fácil desarrollo de los modelos de datos en un entorno de datos heterogéneos.
- La gestión de cambios iterativos en la construcción de un AD.
- Pone en práctica los estándares corporativos.
- Posee un proceso automático de soporte.
- Acelera el diseño de almacenes, con las capacidades tridimensionales, tanto en los modelos de datos lógicos y físicos.
- Determina problemas mediante la identificación de las relaciones, que afectan el linaje de los datos a través de modelos de almacén y el diseño de BI.
- Proporciona una ganancia, productividad adicional con una serie de nuevas características incluyendo la función de los atributos reutilizables y ofrece mejoras significativas en el rendimiento en las áreas de diagramación, la carga de modelo, y la comparación[15].

1.4.3 Oracle SQL Developer Data Modelling

Lleva un control de versiones de colaboración y sistemas abiertos, permitiendo que varios usuarios puedan contribuir al desarrollo del mismo modelo y seguir en detalle cómo un contribuyente ha hecho los cambios en los modelos. Entre sus principales características contempla:

- La compatibilidad con Subversión.
- Incluye una gama completa de herramientas de modelado como: modelo lógico, modelo de tipo de datos, modelo relacional, modelo dimensional, diagramas de flujos de datos, etc.
- Permite diseñar modelos de datos para base de datos Oracle.
- Incluye compatibilidad con SQL Server y DB2.
- Ofrece capacidades de diseño y multi-generación, para producir esquemas conceptuales de entidad-relación, y para transformarlas en modelos relacionales.
- Brinda la oportunidad de trabajar en varios modelos a la vez.
- Posee una herramienta de informes para seguir los cambios en los patrones de relación creado a partir de los modelos y sugerir las mejores prácticas para los usuarios[16].

Todas estas herramientas tienen en común una característica fundamental, apoyar a los desarrolladores en la creación de diseños de AD en un tiempo relativamente corto, reduciendo los costos en tiempo y esfuerzo; pero a la vez todas presentan un inconveniente muy sensible, que consiste en que son herramientas privativas de gran demanda en el mercado del software por lo que la UCI no puede acceder a ellas.

1.5 Metodologías de desarrollo

Las metodologías de desarrollo de software tienden a ser indispensables para la documentación correcta y eficaz de un producto de software, debido a que estas ofrecen un conjunto de procedimientos, técnicas y ayudas a la documentación necesaria para la liberación y aceptación internacional del software. Ellas muestran paso a paso las acciones y actividades a realizar para lograr el producto deseado, indicando además que personas deben participar en el progreso de las actividades y qué papel deben tener. Una metodología es capaz de continuar uno o varios modelos del ciclo de vida, detallando la información que debe obtenerse como resultado de una determinada actividad del proyecto y la información necesaria para su comienzo, indicando que productos parciales y finales hay que obtener a lo largo del desarrollo del proyecto, pero no como hacerlo[17].

Entre las metodologías de desarrollo más conocidas están Programación Extrema (Extreme Programming, XP), clasificada como metodología de desarrollo ágil o ligera y Proceso Unificado de Desarrollo (Rational Unified Process, RUP) clasificada como metodología de desarrollo pesada.

1.5.1 Proceso Unificado de Desarrollo (Rational Unified Process, RUP)

Según la metodología de desarrollo RUP, un proceso de desarrollo de software define quién hace qué, cómo y cuándo. A su vez, precisa cuatro elementos fundamentales, trabajadores (roles) que responden a la pregunta ¿Quién?, las actividades que responden a la pregunta ¿Cómo?, los artefactos (productos), que responden a la pregunta ¿Qué?, y los flujos de trabajo de las disciplinas que responden a la pregunta ¿Cuándo?[18]

RUP se caracteriza por ser dirigido por casos de usos y centrado en la arquitectura. Los casos de uso son los que guían el proceso de desarrollo y la arquitectura muestra la visión común del sistema completo. También es iterativo e incremental, proponiendo en cada fase un desarrollo en iteraciones, estas hacen referencia a pasos durante el flujo de trabajo e incrementa el crecimiento del producto final[19].

La metodología está compuesta por cuatro fases de desarrollo de software:

- Inicio: donde se determina la visión y alcance del proyecto.
- Elaboración: donde se debe determinar una arquitectura óptima para el sistema.
- Construcción: donde se obtiene la capacidad operacional inicial.
- Transición: donde se debe obtener el release (versión operacional del producto) del proyecto[20].

1.5.2 Programación Extrema (Extreme Programming, XP)

La metodología de desarrollo Programación Extrema (XP), se ha convertido en la actualidad en una de las metodologías más exitosas, pues cuenta con la aceptación de la mayoría de los equipos de desarrollo de software. Su autor Kent Beck la definió como una metodología ligera para la creación de software debido a su simplicidad, la comunicación y la realimentación o reutilización del código desarrollado. Su característica más sobresaliente se encuentra en que el cliente forma parte del equipo de desarrollo, requisito necesario para alcanzar el éxito del proyecto[21].

1.5.3 ¿Qué es lo que propone XP?

- Empieza en pequeño y añade funcionalidad con retroalimentación continua.
- El manejo del cambio se convierte en parte sustantiva de proceso.
- El costo del cambio no depende de la fase o etapa.
- No introduce funcionalidades antes que sean necesarias.
- El cliente o el usuario se convierten en miembro del equipo[21].

Lo fundamental en este tipo de metodología es:

- La comunicación entre los usuarios y los desarrolladores.
- La simplicidad al desarrollar y codificar los módulos del sistema.
- La retroalimentación, concreta y frecuente del equipo de desarrollo, el cliente y los usuarios finales[21].

La metodología XP se divide en 4 fases.

- Planificación: Plantea la planificación como un permanente diálogo entre las partes, la empresarial (deseable) y la técnica (posible).
- Diseño: Plantea que el diseño adecuado para el software es el que funciona con todas las pruebas, no tiene lógica duplicada, manifiesta cada intención importante para los programadores, tiene el menor número de clases y métodos.
- Desarrollo: Plantea la recodificación, programación por parejas, la propiedad colectiva, integración continua del código.

- Pruebas: Plantea que no debe existir ninguna característica en el programa que no haya sido probada, los programadores escriben pruebas para chequear el correcto funcionamiento del programa, los clientes realizan pruebas funcionales[22].

1.5.4 ¿Por qué XP?

Se seleccionó XP en aras de simplificar el desarrollo de software y reducir en tiempo y costo, los esfuerzos invertidos en el proyecto. Requiere solo de un pequeño grupo de desarrollo para la aplicación de la metodología, entre 2-15 personas. Es mucho más fácil de implementar y de aprender, por lo que los equipos jóvenes pueden incorporarla de manera más natural. Posibilita que los programadores del proyecto puedan ser comunes (que posean conocimientos básicos). XP combina las que han demostrado ser las mejores prácticas en la industria del desarrollo del software, y las lleva al extremo. Permitirá todo el tiempo rediseñar el código generado (refactorización)[6], dejando el código siempre en el estado más simple posible. Se harán pruebas, no sólo de cada nueva clase (pruebas unitarias) sino que también los clientes comprobarán que el proyecto va satisfaciendo los requisitos (pruebas funcionales). Las iteraciones serán radicalmente más cortas de lo que es usual en otros métodos, esto permite beneficiarse de la retroalimentación tan a menudo como sea posible[23; 24].

Figura # 3 Fase de desarrollo de XP.

[6] **La refactorización** (del inglés Refactoring) es una técnica de la Ingeniería de Software para reestructurar un código fuente, alterando su estructura interna sin cambiar su comportamiento externo.

1.5.5 Roles y artefactos

Un rol es una definición abstracta de un conjunto de actividades realizadas y de artefactos obtenidos. Los roles son realizados típicamente por un individuo, o un conjunto de individuos, trabajando juntos en equipo. Un miembro del equipo de desarrollo del proyecto puede cumplir normalmente muchos roles o un único rol. Los roles no son individuos, sino responsabilidades que deben asumir las personas en un determinado proyecto y describen cómo estos se comportan en el negocio[25].

De acuerdo con las necesidades y características de esta investigación los roles definidos para el desarrollo de la solución serán: programador, cliente y encargado de pruebas.

Programador: En XP los programadores diseñan, programan y realizan pruebas. Son los responsables de tomar decisiones técnicas y construir el sistema. Los artefactos principales serán elaborados por él[24; 26].

Artefactos generados por el rol de programador:

Tarjetas CRC: El uso de tarjetas CRC (Clase, Responsabilidades y Colaboraciones) sirven para diseñar el sistema en conjunto entre todo el equipo. Estas tarjetas representan objetos, la clase a la que pertenece el objeto se puede escribir en la parte de arriba de la tarjeta, en una columna a la izquierda se pueden escribir las responsabilidades u objetivos que debe cumplir el objeto y a la derecha las clases que colaboran con cada responsabilidad.

Cliente: Es parte del equipo, determina qué construir y puede establecer pruebas funcionales[24].

Artefactos generados por el rol de cliente:

Historias de usuario: Las historias de usuario tienen el mismo propósito que los casos de uso. Las escriben los propios clientes, tal y como ven ellos las necesidades del sistema. Existen diferencias entre estas y la tradicional especificación de requisitos. La principal diferencia es el nivel de detalle. Las historias de usuario solamente proporcionarán los detalles sobre la estimación del riesgo y cuánto tiempo conllevará la implementación de dicha historia de usuario[26].

Encargado de Pruebas: Es el responsable de ayudar al cliente a escoger y escribir pruebas funcionales. Es responsable también de hacer funcionar las pruebas regularmente y comunicar sus resultados al resto del equipo. Debe ser una persona con capacidad de abstracción y mucho tacto y facilidad de comunicación, dado que está en un lugar intermedio entre los programadores y el cliente[24].

1.6 Lenguaje de programación y de marcas

1.6.1 Java

Los lenguajes de programación describen la forma en que las computadoras deben ejecutar un conjunto de instrucciones. Cada lenguaje está definido por un conjunto de símbolos, reglas sintácticas y semánticas que definen su estructura y significado de los elementos y expresiones que lo conforma. Las instrucciones que definen los lenguajes de programación se le conocen como código fuente del programa. El lenguaje de programación permite a un programador especificar sobre qué datos la computadora debe operar, cómo deben ser almacenados y transmitidos y cuáles acciones ejecutar ante determinadas circunstancias[27].

Entre los diversos lenguajes de programación se encuentra Java que es un lenguaje que emplea el paradigma de programación orientado a objeto para propósito general. Es descendiente de un lenguaje llamado Oak cuya intención era la creación de software para la televisión interactiva. Consta de plataforma independiente y fue 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 principal característica es la de ser un lenguaje compilado e interpretado. Todo programa en Java ha de compilarse y el código que se genera es interpretado por una máquina virtual, de este modo se consigue la independencia de la máquina, el código compilado se ejecuta en máquinas virtuales que si son dependientes de la plataforma[28].

Para la implementación de la extensión se empleó este lenguaje, por ser el lenguaje de programación propuesto por el API de desarrollo de la herramienta VP.

1.6.2 eXtensible Markup Language (XML)

XML (Lenguaje de Marcas Extensible) es un metalenguaje (un lenguaje que se utiliza para decir algo sobre otro lenguaje). Fue desarrollado por el consorcio web (W3C[7]) en 1995 que es un consorcio internacional que elabora recomendaciones para la Red Informática mundial (WWW[8])[29].

El XML es una adaptación del SGML (Standard Generalized Markup Language), un lenguaje que permite la organización y el etiquetado de documentos. Esto quiere decir que el XML no es un lenguaje en sí mismo, sino un sistema que permite definir lenguajes de acuerdo a las necesidades. El XHTML

[7] W3C de sus siglas en ingles que significan: World Wide Web Consortium
[8] WWW de sus siglas en ingles que significan: World Wide Web

(eXtensible HyperText Markup Language), el MathML (Mathematical Markup Language) y el SVG (Scalable Vector Graphics) son algunos de los lenguajes que el XML está en condiciones de definir.
Las bases de datos, los documentos de texto, las hojas de cálculo y las páginas web son algunos de los campos de aplicación del XML. El metalenguaje aparece como un estándar que estructura el intercambio de información entre las diferentes plataformas[29].
Los expertos nombran varias ventajas que derivan de la utilización del XML:

- Es extensible (se pueden añadir nuevas etiquetas tras el diseño del documento).
- Su analizador es estándar (no requiere de cambios para cada versión del metalenguaje) y facilita el análisis y procesamiento de los documentos XML creados por terceros.
- Numerosas herramientas de procesamiento.
- Admite la definición de vocabularios específicos.
- Separa contenido del procesamiento y visualización.
- Aumenta la seguridad mediante la validación de documentos.
- Formato abierto, respaldado por numerosas organizaciones[30].

Se empleara el uso de este lenguaje, pues para la correcta implementación de la extensión, el API de desarrollo de la herramienta VP, propone que es de vital importancia definir el XML que permitirá la integración de la extensión con la herramienta.

1.7 Herramientas para el desarrollo

1.7.1 Visual Paradigm for UML 8.0

"Visual Paradigm for UML" (VP) es una herramienta CASE[9] que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientado a objetos, construcción, pruebas y despliegue, aumentando la productividad, pues reduce el coste de los mismos en términos de tiempo y dinero. Permite dibujar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación. Es fácil de instalar, actualizar y es compatible entre ediciones. Entre sus principales características se evidencian[31]:
Editor de Detalles de Casos de Uso: Entorno para la especificación de los detalles de los casos de uso, incluyendo la especificación del modelo general y de las descripciones de los casos de uso.
Interoperabilidad: Intercambia diagramas UML y modelos con otras herramientas. Soporta la importación y exportación a formatos XMI y XML y archivos Excel.

[9] CASE por sus siglas en Ingles: (Computer Aided Software Engineering)

Colaboración de Equipo: Realiza el modelado simultáneamente con el Paradigm TeamWork Server y Subversión.
Modelado de Requisitos: Captura de requisitos mediante diagramas de requisitos, modelado de caso de uso y análisis textual.
Multiplataforma: Soportada en plataforma Java para Sistemas Operativos Windows, Linux, Mac OS.
Generación de Documentación: Comparte y genera documentación de diagramas y diseños en formatos PDF, HTML, Microsoft Work.
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.
Ingeniería Inversa: Código a modelo, código a diagrama.
Integración con Entornos de Desarrollo: Apoyo al ciclo de vida completo de desarrollo de software en IDEs 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 utiliza esta herramienta por sus facilidades, además por ser la escogida por la UCI para el desarrollo de software, siguiendo las indicaciones propuestas por el país para obtener la soberanía tecnológica en breve tiempo haciendo uso de las políticas de software libre.

IDE NetBeans 7.0

El IDE NetBeans 7.0 es un entorno integrado de desarrollo de software muy aceptado dentro de la comunidad internacional de programadores. El IDE se encuentra disponible para Windows, Mac, Linux y Solaris. Este IDE es gratuito y de código abierto. Consta con una plataforma de varias aplicaciones que permiten a los desarrolladores crear rápidamente aplicaciones web, escritorio y aplicaciones móviles utilizando la plataforma Java, así como JavaFX, PHP 5.3, Java Script y Ajax, Ruby y Ruby onRails, Groovy y Grails, y C/C++. NetBeans IDE 7.0 continua dando soporte para el compositor de JavaFX ahora con la versión JavaFX 2.0, la cual es 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 2.0 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. Ofrece entre sus mejoras[32]:

- jdk7 y el autocompletado de código, pistas.
- Integración con JSF y librerías del servidor.
- Integración con JUnit 4.8.2 y varias mejoras en JUnit
- Refactoring de renombrado y de borrado seguro.

Y de forma general:

- Wordwrap (ajuste automático de líneas) en el editor
- Mejorada la integración del Profiler
- La comprobación de cambios externos en ficheros ahora es menos intrusiva, cuando se cambia de tarea entre el IDE y otros programas.

Se asume, dada las características y comodidades que brinda el IDE NetBeans 7.0, como herramienta para el desarrollo de la extensión para VP.

1.8 Conclusiones del capítulo 1

Se realizó un detallado estudio sobre el estado del arte de UML y CWM y se consideró que se utilizará Perfiles UML como método de extensión de UML. Al analizar las metodologías y tecnologías necesarias para el desarrollo de la herramienta, y teniendo en cuenta principalmente las exigencias y necesidades del cliente, se definió XP como metodología de desarrollo de software. Se emplea como IDE para el desarrollo NetBeans 7.0 y Java como lenguaje de programación, propuesto por el API de desarrollo de la herramienta VP. Como herramienta de modelado VP en su versión 8.0, adoptado por DATEC en su entorno tecnológico.

2. CAPÍTULO 2: ANÁLISIS Y DISEÑO DE LA EXTENSIÓN PARA MODELAR SOLUCIONES DE AD.

2.1Introducción

En este capítulo se describe la propuesta de solución para la situación problemática anteriormente planteada. Se definirán las principales características que poseerá la aplicación informática a desarrollar, comenzando con los elementos arquitectónicos a tener en cuenta para la implementación de extensiones en VP, y el estudio de la planificación, la definición de los requisitos funcionales y no funcionales y el diseño. Además se generan todos los artefactos de estas fases durante el ciclo de vida del proceso de desarrollo de software seleccionado.

2.2 Propuesta de la extensión

2.3 Arquitectura para las extensiones a "Visual Paradigm for UML"

La implementación de extensiones para la informática constituye uno de los mecanismos para la incorporación de funcionalidades que no se previeron luego del lanzamiento de la aplicación informática. Este es un medio mediante el cual se puede proveer al usuario de nuevas funcionalidades que este requiera. Comúnmente se asocian la palabra extensiones con plugin, que no son más que fragmentos de software que interactúan con el núcleo de la aplicación informática para proporcionar una o varias funcionalidades que en gran parte de los casos son muy específicas. VP es una de las herramientas CASE que más prestigio posee en la actualidad debido a que permite de forma muy ágil y precisa el modelado de software. Pero presenta un inconveniente, no cuenta con un diagrama para modelar estructuras de datos dimensionales de AD. Sin embargo VP cuenta con los medios para extender funcionalidades que no posee, dando soporte a las extensiones de la aplicación. VP provee de forma libre una interfaz de programación (API), permitiendo a los desarrolladores implementar y reutilizar las clases e interfaces, desarrollando funciones agregadas que son útiles para el desarrollo del software en su totalidad[8].

2.3.1 ¿De qué forma se debe desarrollar una extensión para la herramienta CASE "Visual Paradigm for UML"?

Para extender o crear una extensión para VP es importante tener conocimiento de la estructura de desarrollo, así como la posterior integración de la extensión con la herramienta CASE. Es por ello que para su correcto desarrollo ha de realizarse la siguiente consecución de pasos[33]:

Primer paso: La herramienta CASE VP ofrece un medio para la extensión de la aplicación mediante la librería openapi.jar, que se encuentra ubicada dentro de los paquetes de instalación de la herramienta específicamente en el paquete “lib/openapi.jar”. Luego de conocer donde se encuentra la librería para la construcción de la extensión, es necesario tener en cuenta la estructura que debe tener la extensión, que para el caso del IDE de desarrollo NetBeans tiene la estructura de un paquete que lleva el nombre de la extensión, y contendrá a su vez tres paquetes asociados relativos a:

1. Configuración de la extensión.
2. Acciones correspondientes.
3. Formularios o diálogos de la implementación.

De cada uno de estos paquetes se debe dominar su funcionamiento y qué relación establecen con el openapi.jar incorporado al proyecto.

El primer paquete está asociado a la configuración de la extensión y es necesario conocer que se encuentra formado por las clases Plugin.xml y Plugin.java, las cuales permiten la carga y configuración del mismo.

Implementación de Plugin.xml.

Esta permite mediante un Script xml definido, la configuración de la extensión que va ha ser cargado por la clase Plugin.java a través de la implementación de la interfaz VPPlugin ofrecida por la librería openapi.jar, la cual implementa los métodos load y unload, habilitando la carga y descarga de la extensión. Esta permite la conexión entre 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 ver (**Anexo 1**)

Ejemplo de implementación de la clase Plugin.java

```
public class Plugin implements com.vp.plugin.VPPlugin {
// Asegurarse de que el constructor de la clase no tenga parámetros
public void loaded (com.vp.plugin.VPPluginInfo info) {
// Llamado cuando el plugin es cargado
        }
public void unloaded () {
// Llamado cuando el plugin es descargado
        }
}
```

El segundo paquete contiene las acciones correspondientes a realizar por la extensión, las cuales se encuentran definidas a nivel de herramienta y de contexto. Estas clases, en dependencia de cuál sea la acción, implementan la interfaz asociada a dicha acción, VPActionController y VPContextActionController. Dichas clases especifican el performAction pudiendo así ejecutar acciones referentes al evento onClick de la acción[33].

Ejemplos de Implementación:

```
public class ActionController implements com.vp.plugin.action.VPActionController {
ViewManager view = ApplicationManager.instance ().getViewManager();
 public void performAction(VPAction action) {
    view.showMessage("Mi primer plugin");
  }
 public void update(VPAction action) {
    throw new UnsupportedOperationException("Not supported yet.");
  }
}
public class ContexActionController implements com.vp.plugin.action.VPContexActionController {
  ViewManager view = ApplicationManager.instance().getViewManager();
public void performAction(VPAction action, VPContext vpc, ActionEvent ae) {
       view.showMessage("Mi primer plugin");
     }
public void update(VPAction action, VPContext vpc) {
    throw new UnsupportedOperationException("Not supported yet.");
     }
  }
```

Mientras que el tercer paquete del proyecto contiene los diálogos que se necesitan mostrar a través del método performAction, que está asociado a la clase acción correspondiente al dialogo que se necesita utilizar.

Al culminar con la implementación y estructuración del proyecto, se está en condiciones de desarrollar una plantilla de extensión para la herramienta CASE VP.

Estructura de la plantilla de la extensión.

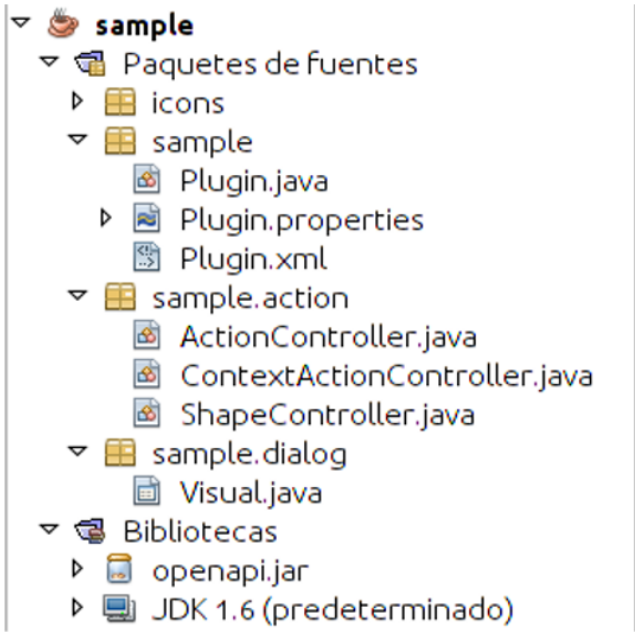

Figura # 4 Se muestra la estructura de un proyecto de extensión para "Visual Paradigm for UML" en el IDE NetBeans.

Segundo paso: Integración de la extensión con la herramienta CASE VP. Para dicha integración VP propone la siguiente estructura a seguir en el despliegue de la extensión. Hay que crear una carpeta con el nombre plugins dentro de los paquetes de instalación de la aplicación, es decir buscar dónde se encuentra instalado VP y crear la siguiente estructura de paquete de despliegue[33]:

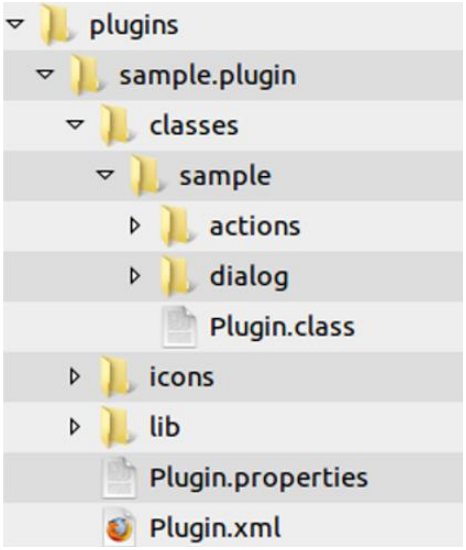

Figura # 5 Se muestra una estructura de despliegue de la extensión.

Si se observa dicha estructura de implementación de la extensión, difiere significativamente con respecto a la estructura de despliegue para la herramienta, lo cual puede dificultar el proceso de pruebas a la hora de integrar la extensión desarrollada con VP. Es por ello que para evitar dichos problemas se utilizará una herramienta de despliegue para la extensión, que facilite la integración del mismo con VP, la cual fue ideada y elaborada en el Departamento de Soluciones Integrales perteneciente a DATEC. La aplicación hace uso de tres argumentos:

1. En el primer paso: Nombre del plugin o extensión.

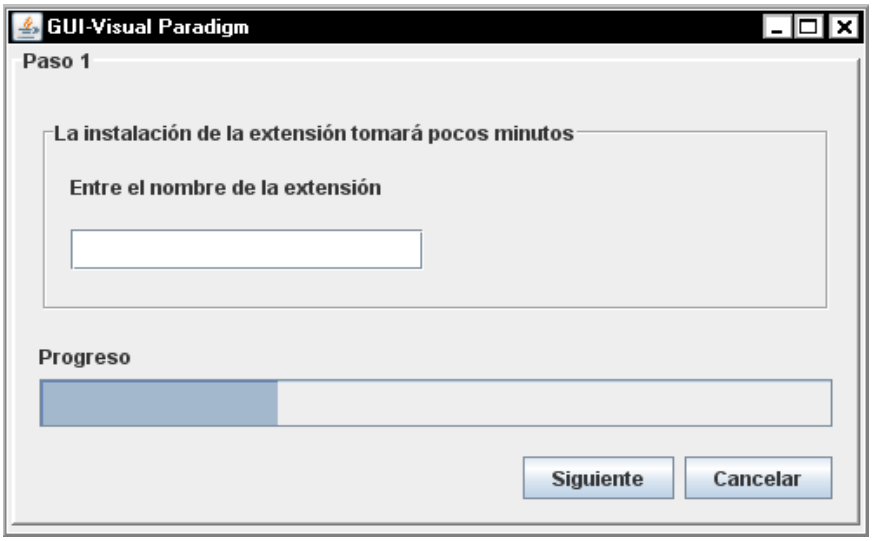

Figura # 6 Se muestra la interfaz número uno del despliegue de la extensión.

2. En el segundo: Dirección Origen

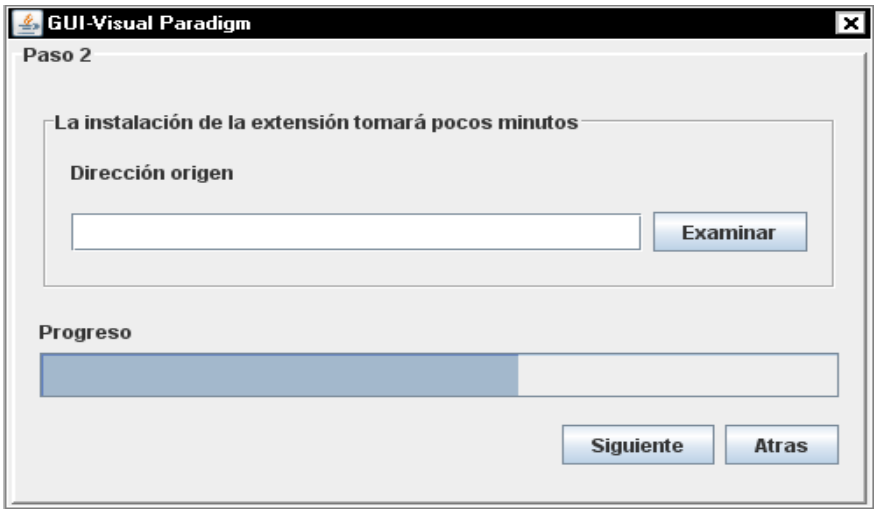

Figura # 7 Se muestra la interfaz número dos del despliegue de la extensión.

3. En el tercero: Dirección Destino

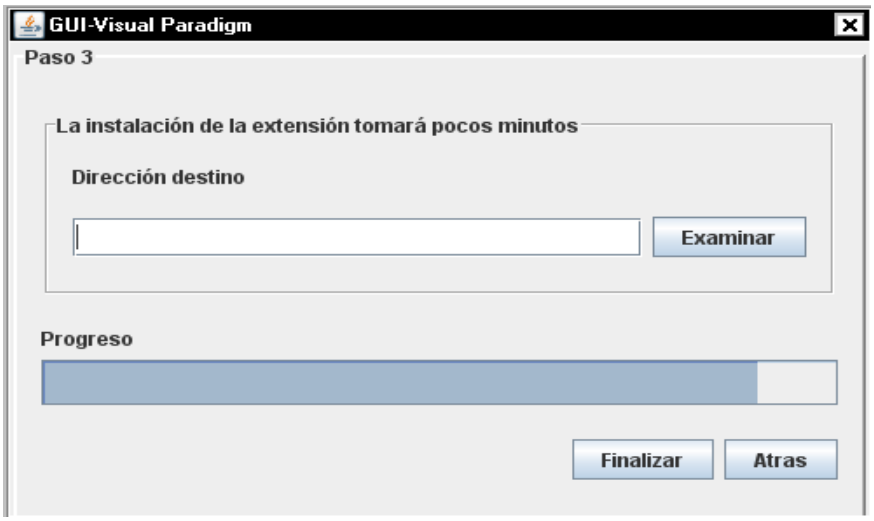

Figura # 8 Se muestra la interfaz número tres del despliegue de la extensión.

De esta forma se está en condiciones de desarrollar una extensión para la herramienta VP, no sin antes hacer referencia al dominio de la arquitectura del openapi.jar, para lo cual se posee la documentación de la misma, donde se facilita la comprensión de la arquitectura y jerarquías existentes en el proceso de construcción de la extensión.

2.4 Requerimientos

El desarrollo de la extensión está orientado principalmente a permitir el modelado de estructuras de datos dimensionales, de forma tal que facilite y agilice el tiempo de desarrollo de un almacén de datos.

2.4.1 Requerimientos funcionales

La extensión estará constituida por una serie de funcionalidades con el objetivo de brindar a los usuarios pasos automatizados en el proceso de creación de los almacenes. Dichas funcionalidades son:

1. Modelar lógicamente la estructura de un almacén de datos.
2. Representar las jerarquías en el modelado del DWH.
3. Generar el modelo físico a partir del modelo lógico.
4. Generar un XML con el diseño preliminar de los cubos OLAP a partir del diseño del DWH.

5. Generar el script para cargar el diseño en el gestor de BD.

2.4.2 Requerimientos no funcionales

Conocidas las funcionalidades que debe realizar la extensión se determina cómo ha de comportarse, qué cualidades debe tener, o cuán rápido debe ser. Estos requisitos son los que normalmente deben apuntar a la arquitectura. En la metodología XP no se definen artefactos para los requisitos no funcionales. Pero para un mejor funcionamiento es necesario que el producto cumpla con cualidades y propiedades, por lo que se decide identificar los requisitos no funcionales de la extensión.

Los requerimientos no funcionales identificados son:

Requerimientos de software

- Sistema operativo: GNU/Linux preferentemente Ubuntu GNU/Linux 10.4 o superior, Debían 6.0 GNU/Linux o superior, Windows XP o superior.
- Máquina Virtual de Java JDK y JRE tanto para GNU/Linux como para Windows.

Requerimientos de hardware

- Procesador a 2.0 GHz o superior, 512 MB de RAM preferentemente 1GB o superior, 300 MB libres de disco duro.

Restricciones del diseño y la implementación

- Será una extensión para VP, para el desarrollo de la misma se emplea la metodología XP. Se utiliza Java como lenguaje de programación.

Requisitos de soporte

- Proveer un manual de usuario.

Requisitos de portabilidad

- La extensión podrá ser integrada a VP en diferentes sistemas operativos, por ser VP multiplataforma.

Una vez detectados los requisitos funcionales y no funcionales, se procede a definir el Perfil UML necesario para el desarrollo de la extensión.

2.5 Perfil UML

El objetivo de este perfil es proporcionar un lenguaje común para diseñar estructuras dimensionales (AD). El perfil permite el modelado lógico del AD. El diseñador de almacenes tendrá a su disposición una herramienta que le permita modelar el diseño lógico del almacén en VP, ayudando así a abstraerse de todos los aspectos técnicos y de código, después de realizado el modelo lógico, VP permitirá realizar automáticamente el diseño físico del almacén haciendo más ágil el proceso de desarrollo del almacén, además de la posibilidad de obtener un fichero XML que admitirá importar la estructura dimensional en la herramienta Schema Workbench para el trabajo del personal de BI. Este modelado lógico permite a los diseñadores que reutilicen los modelos anteriores para realizar nuevos diseños de AD.

2.5.1 Modelo de dominio

El siguiente diagrama es un modelo que muestra los conceptos fundamentales en el modelado de estructuras dimensionales. Como puede observarse, el número de conceptos es relativamente pequeño y debería resultar relativamente familiar a cualquiera que haya trabajado con AD. Se debe aclarar que este modelo de dominio es un diagrama con los objetos que existen (reales) en el diseño de un almacén y las relaciones que hay entre ellos, pero no se encuentran reflejadas clases de software, aunque en el futuro un elemento del modelo de dominio pueda pasar a ser una clase de software. En el siguiente modelo de dominio se representan los conceptos fundamentales (representados como entidades) de un almacén de datos: Hecho y Dimensión así como las relaciones entre ellos.

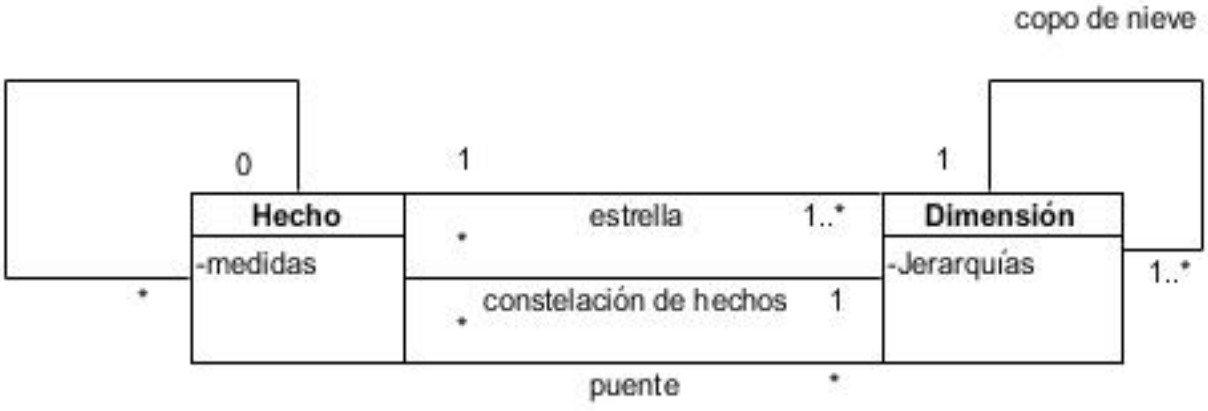

Figura # 9 Modelo de dominio.

Tabla # 1 Descripción de entidades del perfil.

No.	Entidad	Descripción	Estereotipo	Meta clase UML 2.0
1	hecho	Se denomina evento o hecho a una operación que se realiza en el negocio en un tiempo determinado. Son objeto de análisis para la toma de decisiones. Los hechos contienen medidas donde una Medida es una propiedad de un hecho (casi siempre numérica), que es usada para su análisis[34].	<<Hecho>>	Class
2	dimensión	Una dimensión es una característica de un hecho que permite su análisis posterior, en el proceso de toma de decisiones. Las dimensiones tienen atributos que se pueden agrupar en jerarquías[34].	<<Dimensión>>	Class

2.5.2 Especificación del perfil

El siguiente diagrama de Perfil UML demuestra los detalles reales del perfil con cada estereotipo, describe meta clases usando la notación de extensión (cabeza de flecha llena). También pueden verse algunas restricciones de este modelo que se plantean a continuación:

Un hecho puede relacionarse entre sí, pero, los hechos relacionados con otros hechos son ejemplos de malas prácticas, también pueden relacionarse con una o más dimensiones. Un puente es una relación de muchos a muchos entre un hecho y una dimensión, las relaciones uno a muchos entre hechos y dimensiones son relaciones normales. Las relaciones normales entre un hecho y sus dimensiones asociadas se denominan estrellas. Las dimensiones pueden relacionarse entre sí.

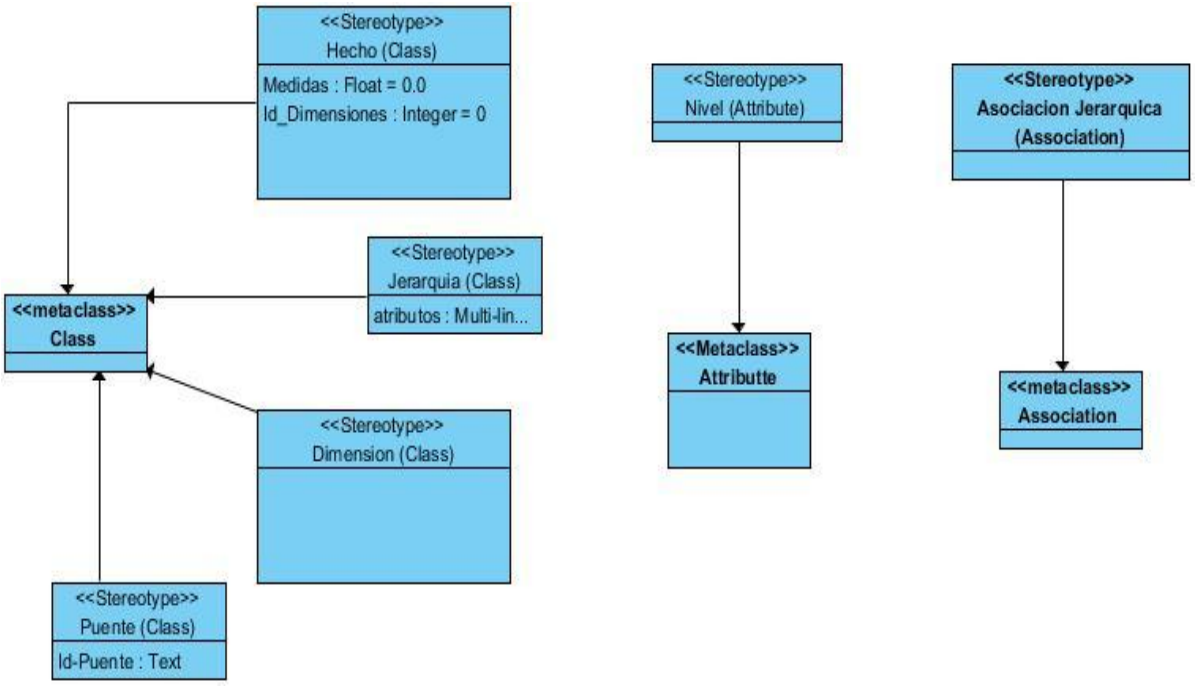

Figura # 10 Diagrama de Perfil UML 2.0.

Estereotipo <<Hecho>>

Meta clase(s) UML a la que amplia: Class

Semántica del estereotipo:

Un hecho representa el concepto tal como se define en un almacén de datos, es donde se representaran las medidas que generalmente son numéricas y que son la intersección de las dimensiones asociadas al hecho, un hecho debe estar relacionado con al menos una dimensión. No tiene razón de ser un almacén de datos sin un hecho, por lo que es obligatorio al menos modelar un hecho. Además contendrá los id de las dimensiones asociadas a él como llaves primarias de la tabla de hechos. Los hechos relacionados con otros hechos son ejemplos de malas prácticas. El hecho es una tabla en el modelo físico.

Estereotipo <<Dimensión>>

Meta clase(s) UML a la que amplia: Class

Semántica del estereotipo:

Una dimensión representa el concepto tal como se define en un almacén de datos, las dimensiones definen dominios. Una dimensión puede estar relacionada con un hecho o una dimensión. Contendrá atributos que se podrán agrupar en jerarquías. Es una tabla en el modelo físico.

Estereotipo <<Jerarquía>>

Meta clase(s) UML a la que amplia: Class

Semántica del estereotipo:

En la jerarquía es donde se encuentran los atributos que se pueden agrupar por niveles, por ejemplo en la dimensión Tiempo, donde pueden existir los atributos Mes, Año y Día. Se les puede otorgar niveles a estos, dado que en un orden lógico Año puede estar en el primer nivel, Mes en el segundo y Día en el tercero, para buscar una cantidad de productos X vendida en Y año, Z mes. Agrupar por jerarquías facilita al diseñador una mejor visión lógica para filtrar y obtener la información con mayor facilidad.

Estereotipo <<Puente>>

Meta clase(s) UML a la que amplia: Class

Semántica del estereotipo:

Un puente representa el concepto tal como se define en un almacén de datos, es una entidad que surge producto de la relación de muchos a muchos entre un hecho y una dimensión y contendrá como llaves primarias la llave primaria del hecho y de la dimensión que están implicadas en la relación de muchos a muchos. Podemos tomar como ejemplo el hecho Persona y la dimensión Organización Política, una persona puede estar asociada a varias organizaciones políticas (UJC, PCC) y una organización política puede tener a varias personas en su seno. En la metodología propuesta por DATEC se debe realizar un puente en estos casos, para facilitar la búsqueda en la base de datos. Es una tabla en el modelo físico.

Estereotipo <<Nivel>>

Meta clase(s) UML a la que amplia: Attributte

Semántica del estereotipo:

Un nivel representa el concepto tal como se define en un almacén de datos, es un atributo que forma parte de una jerarquía, y que se puede clasificar como perteneciente a un nivel. Se utilizará para diferenciar un atributo perteneciente a una jerarquía de otro no perteneciente a esta y que se encuentre en la misma dimensión.

Estereotipo <<Asociación Jerárquica>>

Meta clase(s) UML a la que amplia: Association

Semántica del estereotipo:

Una asociación Jerárquica es la relación entre una entidad dimensión y una entidad jerarquía, surge por la necesidad de identificar en un diagrama UML la existencia de jerarquías en una dimensión, para saber qué jerarquía pertenece a qué dimensión, una misma jerarquía solo puede tener una relación "Asociación Jerárquica" con una dimensión. Una dimensión puede tener varias relaciones "Asociación Jerárquica" con varias jerarquías.

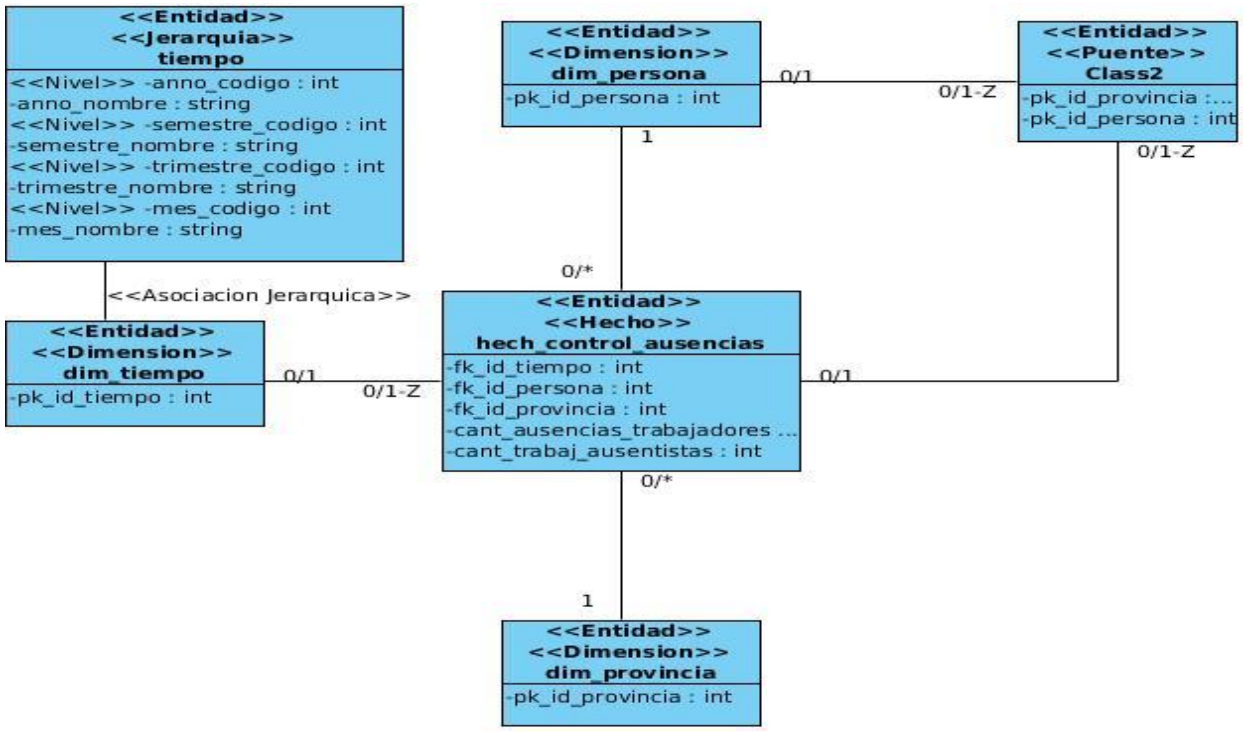

Figura # 11 Ejemplo de diagrama realizado en Visual Paradigm for UML con la extensión incorporada.

Como puede observarse en el ejemplo, existen tres entidades estereotipadas como dimensión con sus atributos, las cuales están relacionadas con la entidad estereotipada como hecho, el cual tiene sus atributos (llaves y medidas), representando la relación existente entre una tabla de hecho y sus dimensiones, asimismo se tiene una entidad estereotipada jerarquía para representar las jerarquías pertenecientes a determinada dimensión, la relación entre la dimensión y la Jerarquía es estereotipada Asociación Jerárquica para diferenciarla de las demás asociaciones a la hora de la conversión al modelo físico.

Habiendo detectado los requisitos funcionales y no funcionales, y elaborado el Perfil UML necesario para la extensión, se procede a definir los artefactos que se generaran a partir de los mismos como

parte de la fase de planificación, los artefactos generados a partir de los requisitos funcionales se denominan Historias de Usuario.

2.6 Planificación.

La metodología Programación Extrema asume que la planificación nunca podrá ser perfecta, debido a que variará en función de las necesidades del negocio, de ahí que es fundamental que se especifiquen los requisitos no funcionales y funcionales de la extensión, y donde se aplicarán diferentes iteraciones. Para hacerlo será necesaria la existencia de reglas que se han de seguir por las partes implicadas en el proyecto para que todas las partes tengan voz y se sientan realmente partícipes de la decisión tomada. En esta fase se elaboran las historias de usuarios que constituyen unos de sus artefactos más importantes así como el plan de iteraciones y el plan de entregas. Para realizar una correcta planificación la metodología XP propone que se reúna a los clientes, programadores y gerentes (tutores de tesis) del proyecto una vez determinadas las Historias de Usuario para trazar el plan de entregas (release plan) y el plan de iteraciones. Para dar comienzo a la planificación se definen las historias de usuario.

2.7 Historia de usuarios.

Las historias de usuarios sustituyen a los documentos de especificación de requisitos y a los casos de uso; Son una técnica para especificar requisitos, tanto no funcionales como funcionales del software a crear. En semejanza con los casos de uso tendrían el mismo propósito, el de conducir el proceso de creación de los test de aceptación empleados para verificar que han sido implementadas correctamente. Proporcionan los detalles sobre la estimación del riesgo y cuánto tiempo se empleará en la implementación. Las historias de usuario son escritas por el cliente, en lenguaje natural, y como descripciones cortas de lo que el sistema debe realizar o cumplir.

A continuación se muestra una de las historias de usuario definidas para la implementación de la extensión. El resto aparece en el artefacto “Plantilla de historias de usuario”.

Tabla # 2 Historia de Usuario: Crear diagrama de estructura dimensional.

Historia de Usuario	
Número: 1	**Nombre:** Crear diagrama de estructura dimensional

Usuario: Desarrollador	
Prioridad en el Negocio: Alta	**Nivel de Complejidad:** Alta
Puntos de Estimación: 3	**Iteración Asignada:** 1
Programador responsable: Alain Suárez Suárez – Sergio Martin Torres	
Descripción: Se le brinda la posibilidad al desarrollador de modelar un diagrama dimensional de un almacén, junto con sus relaciones, haciendo uso de estereotipos mediante la creación de un Perfil UML	
Información adicional (Observaciones):	

Para garantizar el desarrollo de la extensión propuesta, se realizó una estimación de la duración para cada una de las historias de usuarios identificadas, llegando a los resultados que se muestran a continuación:

Tabla # 3 Estimación de la duración del desarrollo de las historias de usuarios.

Historias de usuarios	Puntos de estimación
Crear diagrama de estructura dimensional	3 semanas
Transformar las estructuras dimensionales en modelos Entidad-Relación.	3 semanas
Generar fichero XML para el Pentaho Schema Workbench	2 semanas
Crear script para exportar a la base de datos.	1 semana

Luego de haber definido las historia de usuarios y la estimación por puntos de la duración de cada una de ellas, se decide desarrollar la extensión en cuatro iteraciones. Las iteraciones tienen como objetivo que el cliente decida cuáles historias de usuarios implementar según la prioridad que tengan en el negocio, las mismas no deben exceder las tres semanas. A continuación se expone el plan de las iteraciones.

2.8 Plan de iteraciones

Iteración 1: En esta iteración se desarrolla la historia de usuario número uno, la cual permitirá diseñar el modelo lógico de un almacén de datos, permitiendo a su vez el conjunto de relaciones que conforman el dominio de un almacén.

Iteración 2: En esta iteración se corregirán los errores o no conformidades del usuario identificados en la iteración anterior. Se desarrollará la historia de usuario número dos, la cual permitirá transformar de manera automática el diseño dimensional del almacén de datos a un diagrama entidad-relación.

Iteración 3: En esta iteración se corregirán los errores o disconformidades del usuario en la historia de usuario implementada en la iteración anterior. Se desarrollará la historia de usuario tres la cual permitirá generar un script xml de plantilla para la herramienta “Pentaho Schema Workbench”.

Iteración 4: En esta iteración se corregirán los errores o disconformidades del usuario en la historia de usuario implementada en la iteración anterior. Se desarrollará la historia de usuario número cuatro, la cual permitirá una vez generado el diagrama entidad-relación, exportar el script sql del almacén de datos. Además, se realizaran pruebas integrales a la extensión para validar todas sus funcionalidades corrigiendose los errores o las no conformidades que se identifiquen y las que no han sido satisfechas en las iteraciones anteriores.

Una vez establecidas las iteraciones se pasa a realizar el plan de duración de cada una de las iteraciones, garantizando de esta forma un desarrollo exitoso y con calidad. El plan de duración de iteraciones representa las iteraciones junto con las historias de usuarios que van a hacer implementadas en cada una de ellas y la duración total de cada iteración.

Tabla # 4 Plan de duración de las iteraciones.

Iteraciones	Historias de usuarios	Duración total de iteraciones
Iteración 1	Crear diagrama de estructura dimensional	3 semanas
Iteración 2	Transformar las estructuras dimensionales en modelos Entidad-Relación.	3 semanas

Iteración 3	Generar fichero XML para el Pentaho Schema Workbench	2 semanas
Iteración 4	Crear script para exportar la base de datos. Crear diagrama de estructura dimensional. Transformar las estructuras dimensionales en modelos Entidad-Relación. Generar fichero XML para el Pentaho Schema Workbench	2 semanas
Total	10 semanas	

Para la fase de implementación de la extensión se define el siguiente plan de entrega con una propuesta de la fecha aproximada en que se realizarán las versiones de prueba a la aplicación informática.

Tabla # 5 Plan de entrega.

Extensión para modelar soluciones de almacenes de datos en la herramienta de modelado **"Visual Paradigm for UML"**.	**Entregable**
Versión 0.1	Cuarta semana de marzo
Versión 0.2	Primera semana de abril
Versión 0.3	Tercera semana de abril

Versión 1.0	Primera semana de mayo

Una fase importante de todo desarrollo de software es la fase de diseño, pues es donde se definirá la arquitectura que dará soporte a la estructura del desarrollo del software.

2.9 Diseño

La metodología X.P sugiere que hay que conseguir diseños simples y sencillos. Procurando hacerlo todo lo menos complicado posible para conseguir un diseño que permita ahorrar tiempo y esfuerzo para desarrollar la solución. Para ello se apoya en las tarjetas CRC, las cuales permiten al programador centrarse y apreciar el desarrollo orientado a objetos, olvidándose de los malos hábitos de la programación estructurada clásica.

Las tarjetas C.R.C (Clase, Responsabilidades y Colaboraciones) representan objetos; la clase a la que pertenece el objeto se puede escribir en la parte de arriba de la tarjeta, en una columna a la izquierda se pueden escribir las responsabilidades u objetivos que debe cumplir el objeto y a la derecha, las clases que colaboran con cada responsabilidad.

La extensión contará con seis tarjetas CRC, en las cuales se representan cada una de las clases de la extensión informática con sus responsabilidades y colaboraciones con las demás clases, a continuación se muestra un ejemplo de la tarjeta CRC Funciones, el resto aparece en el artefacto “Plantilla modelo de diseño”.

Tabla # 6 Tarjeta CRC Funciones.

Tarjeta CRC	
Clase: Funciones	
Responsabilidades	**Colaboraciones**

Obtener asociaciones. Obtener clases. Obtener atributos. Obtener tablas. Obtener llaves primarias. Obtener columnas primarias. Obtener clases seleccionadas. Obtener asociación seleccionada. Obtener diagrama activo. Obtener mensaje de advertencia. Obtener listado real de asociaciones. Buscar asociación. Buscar clase. Crear atributo. Crear relación uno a uno. Crear relación uno a mucho. Crear relación mucho a mucho. Crear diagrama entidad relación. Cambiar nombre del atributo foráneo.	IModelElementFactory DiagramManager.

Los patrones de diseño son muy importantes en el desarrollo de software actual, pues cada patrón ayuda a los desarrolladores de software a crear un lenguaje común para comunicar ideas y experiencias acerca de los problemas y sus soluciones.

2.9.1 Patrones de diseño

Los patrones de diseño agilizan el desarrollo del software pues permiten reutilizar diseños que han demostrado ser la solución correcta a problemas comunes. Debido a ello muchos autores convergen en la definición dada por Christopher Alexander:

"… 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"[35]. Para la implementación y funcionalidad de

la extensión se hace uso de los patrones de diseño GOF con el fin de agilizar el proceso de implementación al desarrollador.

2.9.2 Patrones de diseño GOF

Los patrones de diseño GOF constituyen en la actualidad uno de los patrones más utilizados y conocidos en el diseño orientado a objetos, se dio a conocer a través del libro "Design Patterns: Elements of Reusable Object Oriented Sofware" escrito por los Gang of Four (GOF, que en español significa la pandilla de los cuatro) integrada por Erich Gamma, Richard Helm, Ralph Johnson y John Vlissides. Ellos recopilaron y documentaron un total de 23 patrones de diseño aplicados usualmente por expertos diseñadores de software orientado a objetos. Los patrones de diseño GOF se clasifican en[36]:

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:

- Clases: Relaciones estáticas entre clases.
- Objetos: Relaciones dinámicas entre objetos.

Factory Method (Método de Fabricación)

Para la implementación de extensiones en soluciones informáticas, se recomienda el uso de patrones como el conocido por método de Fabricación, el cual se utiliza directamente en la implantación, garantizando la instanciación de los objetos, dado que las extensiones en ocasiones son realizadas por externos al grupo de desarrollo que construyó el software para el cual se desea realizar la extensión.

Descripción:

El Método de Fabricación, centraliza en una clase constructora la creación de objetos de un subtipo de un tipo determinado, ocultando al usuario la casuística para elegir el subtipo que crear. Permitiendo que una clase delegue en sus subclases la creación de objetos ganando en flexibilidad, y facilitando 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.

¿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[37].

¿Cómo funciona?

Utilizando métodos de creación, para crear componentes y garantizar a través de estos, que la extensión funcione de forma correcta.

A continuación se muestra un ejemplo del empleo del patrón:

```
public IAttribute crearAtributo(String nombre){
IAttribute atributo = IModelElementFactory.instance().createAttribute();
String nom = nombre.substring(0, 1).toUpperCase().concat(nombre.substring(1))
   | atributo.setName(nom);
    return atributo;
}
```

Figura # 12 Ejemplo del empleo del patrón Método de Fabricación.

Iterator (Iterador)

Descripción:

El patrón Iterador, es de tipo: comportamiento a nivel de objetos y permite realizar recorridos sobre objetos compuestos independientemente de la implementación de estos. De esta forma se proporciona un modo de acceder secuencialmente a los elementos de un objeto agregado sin exponer su representación interna. 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.

¿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[37].

¿Cómo funciona?

Para la implementación de la extensión 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.

```
public LinkedList<IClass> getClases(IProject project) {
LinkedList<IClass> Listaclases = new LinkedList<IClass>();
Iterator elementos = project.modelElementIterator();

while (elementos.hasNext()) {
IModelElement elemento = (IModelElement) elementos.next();
if(elemento.getModelType().equals(IModelElementFactory.MODEL_TYPE_CLASS)){
                Listaclases.add((IClass) elemento);
            }
        }
        return Listaclases;
    }
```

Figura # 13 Ejemplo del empleo del patrón Iterador.

Singleton (Solitario)

Descripción:

El patrón Solitario, es de tipo comportamiento a nivel de objetos, su propósito es garantizar la existencia de una única instancia para una clase y la creación de un mecanismo de acceso global a dicha instancia. 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)[37].

¿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 Conections.java que permite la creación de objetos a través de instancias, manteniendo la consistencia entre los objetos.

2.10 Conclusiones del capítulo 2

A partir de las necesidades del Departamento de Almacenes de Datos de DATEC se llevó a cabo la descripción del Perfil UML para representar los patrones de análisis y diseño de los AD. Se diseñó un perfil que permite a VP extenderse e incorporar el dominio de diseño de estructuras dimensionales (AD). El perfil comprende los criterios de diseño planteados por el departamento para la creación de AD. En esta descripción, se confeccionaron plantillas que se encuentran contenidas por el Modelo de Dominio para el Análisis, Diseño e Implementación, la especificación del perfil y de las entidades que

integran los distintos modelos de dominio, la extensión contiene además los estereotipos necesarios, valores etiquetados y restricciones para modelar de manera formal los elementos utilizados durante las etapas de análisis, arquitectura y desarrollo de un almacén. Para el desarrollo de la extensión se identificaron cuatro historias de usuarios describiendo los requisitos funcionales de la extensión y seis tarjetas CRC que permitieron el correcto desarrollo de la misma. Se definieron un total de 51 tareas de implementación que permitirán el correcto desarrollo de las historias de usuarios.

3. CAPÍTULO 3: IMPLEMENTACIÓN Y PRUEBAS DE LA EXTENSIÓN PARA MODELAR SOLUCIONES DE AD.

3.1Introducción

En este capítulo se realizará la representación del código fuente de las principales funcionalidades codificadas para la extensión, atendiendo a estándares de codificación ya creados. Se obtendrán resultados del diseño implementando la extensión informática y se garantizará la calidad del software mediante la realización de pruebas de aceptación.

3.2 Implementación

Para el desarrollo satisfactorio de la implementación se definen un grupo de tareas de desarrollo, las cuales representan fracciones del conjunto de acciones a realizar para el progreso satisfactorio de cada una de las historias de usuarios. Las tareas ayudan a organizar la implementación y son solamente usadas por los programadores, pues pueden estar descritas en un lenguaje técnico y no serían necesariamente entendibles por el cliente.

De acuerdo a la planificación realizada en el capítulo anterior, se llevaron a cabo cuatro iteraciones, obteniéndose como finalidad un producto funcional con las características deseadas. A continuación se expone un ejemplo de la iteración número uno.

Iteración 1

Se desarrolla la historia de usuario número uno, la cual permite modelar las estructuras dimensionales en un diagrama de clase.

- **HU Crear diagrama de estructura dimensional.**

Tabla # 7 Tarea de la implementación # 1

Tarea	
Número Tarea: 1	**Número Historia de Usuario:** 1
Nombre Tarea: Diseño de la interfaz para las propiedades de la asociación	
Tipo de Tarea: Desarrollo	**Puntos Estimados:** 0.1

Fecha Inicio: 1/1/2012	**Fecha Fin:** 2/1/2012
Programador Responsable: Sergio Martín Torres – Alain Suárez Suárez	
Descripción: Se implementa el diseño de la interfaz para las propiedades de la asociación, el cual permite visualizar y manejar los nuevos atributos correspondientes a la asociación.	

Para la correcta codificación del código fuente de la extensión en el lenguaje de programación Java, se hace indispensable la utilización de un estándar de codificación.

3.3 Estándar de codificación

Los estándares de codificación o convenciones de código, también son conocidos por el nombre común de estilos de programación. Un estándar de codificación constituye un convenio entre los programadores para escribir código fuente en un determinado lenguaje de programación, elevando la mantenibilidad del código, pues constituyen un punto de referencia para los programadores. Un estándar permite mantener un estilo de programación, pues ayuda a mejorar el proceso de codificación, haciéndolo, entre otras cosas, mucho más eficiente.

Las convenciones de código son importantes para los programadores por un gran número de razones:

- El 80% del coste del código de un programa va a su mantenimiento.
- Casi ningún software lo mantiene toda su vida el autor original.
- Las convenciones de código mejoran la lectura del software, permitiendo entender código nuevo mucho más rápidamente y más a fondo.
- Si se distribuye el código fuente como un producto, se debe asegurar que está bien hecho y presentado como cualquier otro producto[38].

Algunas de estas convenciones se pueden ver a continuación:

- **Identación**

Se deben emplear cuatro espacios como unidad de identación.

- **Longitud de la línea**

Evitar las líneas de más de 80 caracteres, ya que no son manejadas bien por muchas terminales y herramientas.

- **Comentarios**

Existen dos tipos de comentario, los de implementación y los de documentación, estos últimos existen solo en Java y se encuentran limitados por /**…*/.

```
/**
 * La clase Ejemplo ofrece ...
 */
public class Ejemplo { ...
```

- **Declaraciones**

Se realizará una declaración por línea, ya que facilita los comentarios. Es decir, preferiblemente

```
int nivel; // nivel de indentación
int tam;   // tamaño de la tabla
```

antes de

```
int level, size;
```

- **Inicialización**

Se inicializarán las variables locales donde se declaran. La única razón para no inicializar una variable donde se declara es si el valor inicial depende de algunos cálculos que deben ocurrir.

- **Declaraciones de clases**

Al codificar clases e interfaces de Java, se siguen las siguientes reglas de formato:

- No debe existir ningún espacio en blanco entre el nombre de un método y el paréntesis "(" que abre su lista de parámetros. Los nombres de las clases comienzan siempre por mayúsculas y si son nombres compuestos se separan por un guion bajo "_". La llave de apertura "{" aparece al final de la misma línea de la sentencia declaración.
- La llave de cierre "}" empieza una nueva línea indentada para ajustarse a su sentencia de apertura correspondiente, excepto cuando no existen sentencias entre ambas, que debe aparecer inmediatamente después de la de apertura "{"

- Los métodos se separan con una línea en blanco.

```
class Ejemplo extends Object {
    int ivar1;
    int ivar2;

    Ejemplo(int i, int j) {
        ivar1 = i;
        ivar2 = j;
    }

    int metodoVacio() {}

    ...
}
```

- **Sentencia if, if-else, if else-if else**

La clase de sentencias if-else debe tener la siguiente forma y deben usar siempre llaves {}.

```
if (condicion) {
    sentencias;
}

if (condicion) {
    sentencias;
} else {
    sentencias;
}

if (condicion) {
    sentencia;
} else if (condicion) {
    sentencia;
} else{
    sentencia;
}
```

- **Sentencias for**

Una sentencia for debe tener la siguiente forma:

```
for (inicializacion; condicion; actualizacion) {
    sentencias;
}
```

- **Sentencias while**

Una sentencia while debe tener la siguiente forma:

```
while (condicion) {
    sentencias;
}
```

- **Variables**

```
class Ejemplo extends Object {
    int ivar1;
    int ivar2;

    Ejemplo(int i, int j) {
        ivar1 = i;
        ivar2 = j;
    }

    int metodoVacio() {}

    ...
}
```

Todas las instancias y variables de clase o método empezarán con minúscula. Las palabras internas que lo forman (si son compuestas) empiezan con su primera letra en mayúsculas. Los nombres de variables no deben empezar con los caracteres Guión bajo "_" o signo de dólar "$", aunque ambos están permitidos por el lenguaje.

- **Métodos**

```
class Ejemplo extends Object {
    int ivar1;
    int ivar2;

    Ejemplo(int i, int j) {
        ivar1 = i;
        ivar2 = j;
    }

    int metodoVacio() {}

    ...
}
```

Los métodos cuando son compuestos tendrán la primera letra en minúscula, y la primera letra de las siguientes palabras que lo forma en mayúscula.

Descrito el estándar de codificación a utilizar, abordaremos una pequeña descripción de cómo se realizó la transformación del modelo de clases a un modelo Entidad-Relación en "Visual Paradigm for UML".

3.4 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.

Para la transformación del Diagrama de Clases a Diagrama Entidad Relación se capturan todos los elementos IClass e IAssociation presentes en el diagrama activo. El proceso es sencillo se crea una instancia de la clase DiagramManager mediante la interfaz ApplicationManager, la cual permite obtener el diagrama activo mediante el método getActiveDiagram(), posteriormente se capturan todos los componentes, en este caso se refiere a la interfaz más genérica asociada a los componentes, el IModelElement del cual heredan cada uno de los componentes antes mencionados. Luego se verifica si el componente es de tipo Modelo de Diagrama de Clase en este caso una clase o asociación, de esta forma tenemos el objeto clase del cual podemos extraer información de sus atributos que serán convertidos a columnas en la tabla asociada a la clase.

Para crear la tabla se hace mediante el uso de la interfaz IModelElementFactory la cual implementa un patrón Singleton, creamos una instancia y mediante el método de Fabricación createDBTable(), es creada la tabla. La tabla está compuesta por un nombre y columnas que son incluidas a la misma. Para dar nombre a la tabla basta con cambiar mediante el método setName() asociado a la IDBTable, pasando como nombre el de la clase mediante el método getName() asociado al IClass. Para la creación de columnas se utiliza la misma interfaz IModelElementFactory definida por el openapi.jar para la creación de objetos IModelElement. Luego se itera sobre la clase obteniendo una colección de campos para agregarlos como columnas a la tabla mediante los métodos addDBColumn().Si las clases o atributos presentan estereotipos se les define mediante el método addStereotype() del objeto IAttribute o IClass.

Para definir las relaciones entre tablas, se hace mediante el uso de la interfaz IModelElementFactory, la cual presenta un método de Fabricación createDBForeignKey() que permite instanciar un objeto de

llave foránea. El objeto de llave foránea IDBForeignKey presenta dos métodos setFrom() y setTo() que permiten definir la tabla de donde parte la relación y hacia qué tabla está dirigida.

Una vez creada las tablas y las relaciones entre ellas solo queda visualizar el diagrama entidad relación, para ello es necesario haber creado un IDiagramUIModel mediante la interfaz ApplicationManager. Esta permite crear un diagrama, de tipo "Diagrama de Entidad-Relación" adicionando IDiagramElement al diagrama, convirtiendo los IDBTable o sea las tablas a elementos del diagrama mediante el método createDiagramElement() asociado a la interfaz ApplicationManager. Una vez convertido las tablas y adicionado al IDiagramUIModel es visualizado mediante la ApplicationManager a través del método openDiagram(), visualizando en el área de trabajo de VP el diagrama entidad relación.

Para que todo software sea liberado con calidad, debe ser sometido a una serie de pruebas que determinaran si el software cumple con el mínimo de funcionalidad esperado.

3.5 Pruebas.

Descrita la implementación de la solución se pasa a explicar la fase de prueba que validará la efectividad de la solución implementada. Las pruebas son fundamentales para asegurar la calidad final del producto. XP es una metodología "Test-driven programming", lo que significa que está dirigida por pruebas. En XP se planifican varios ciclos de iteraciones, donde pueden estar una o varias historias de usuario incluidas en cada ciclo. A diferencia de las metodologías tradicionales en las que la fase de prueba es al final del proyecto y cuando está implementada la solución, XP planifica y crea las pruebas antes de implementar cualquier solución del proyecto. La definición de estas pruebas al comienzo, condiciona o "dirige" el desarrollo del proyecto ya que al plantearse las pruebas con anterioridad el desarrollador solo deberá preocuparse por realizar el esfuerzo mínimo necesario para pasar las pruebas anteriormente planteadas, agilizando así el proceso de obtención de la solución. XP plantea una fase prueba para cada iteración, además, se deben realizar pruebas de aceptación para cada historia de usuario, al concluir cada ciclo se realizan las pruebas de aceptación, así como en cada ciclo consiguiente con el objetivo de verificar que las iteraciones posteriores no afecten a las anteriores. Las pruebas de aceptación que hayan sido fallidas en el ciclo anterior se analizan para evaluar su corrección y prevenir que no vuelvan a ocurrir. Al no tener problemas con la estimación del tiempo de las historias de usuario, no se hizo necesario la implementación de pruebas Spikes. Las pruebas de aceptación mencionadas anteriormente son consideradas como 'Pruebas de Caja Negra", las pruebas

de caja negra se centran en las funciones, entradas y salidas del sistema, por lo que son referidas como pruebas funcionales.

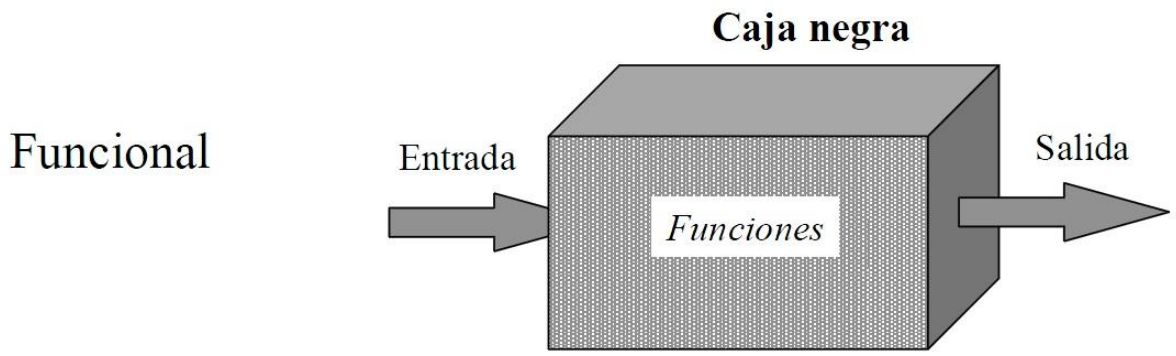

Figura # 14 Funcionamiento de las pruebas de caja negra.

Se definieron un conjunto de pruebas de aceptación con el cliente, a continuación se muestra la primera para la historia de usuario número uno, para las demás tablas dirigirse al anexo número 2.

Tabla # 8 Prueba de aceptación # 1

Caso de prueba de aceptación	
Código: HU1_p1	**Historia de Usuario:** 1
Nombre: Crear diagrama de estructura dimensional.	
Nombre de la persona que realiza la prueba: José Salvador Bermúdez Rodríguez	
Descripción de la Prueba: Probar que el sistema permita que se puedan realizar modelos de estructuras dimensionales con sus hechos y dimensiones así como sus atributos y relaciones.	
Condiciones de ejecución: El usuario debe ejecutar el Visual Paradigm for UML con la extensión previamente incorporada.	
Entrada/ pasos de ejecución: En Visual Paradigm for UML en la sección dedicada al modelado de estructuras dimensionales el usuario debe seleccionar para su	

representación los hechos y dimensiones y relacionarlos entre sí, así como definir sus atributos y jerarquías.
Resultado esperado: Que la aplicación permita representar los hechos y dimensiones como entidades con sus estereotipos correspondientes, la relación de los hechos con sus dimensiones, así como los atributos de estos.
Evaluación de la prueba: Satisfactoria

A continuación se expondrán algunos ejemplos de pruebas de caja negra por escenarios.

Tabla # 9 Escenario eliminar entidad, del caso de prueba basado en Historias de Usuario: Crear diagrama de estructura dimensional.

Escenario	Descripción	Respuesta del sistema	Flujo central
EC 1.1 Eliminar entidad	El usuario elimina la entidad que ha sido creada.	La dimensión ha sido eliminada correctamente.	1-El usuario presiona clic izquierdo en el botón Ext Vpmd de la barra de tareas del Visual Paradigm for UML, seguidamente en el botón insertar y después en la entidad deseada. 2-El usuario especifica el nombre de la entidad. 3-El usuario accede los campos especificados y adiciona los atributos. 4- El usuario da clic en el botón adicionar para agregar los atributos. 5-El usuario da clic en el botón aceptar para insertar la dimensión con sus atributos. 6- El usuario da clic izquierdo sobre la dimensión creada para seleccionarlo y presiona la tecla Delete o Ins.

En las siguientes imágenes se observan pruebas realizadas al funcionamiento de la aplicación con resultados satisfactorios.

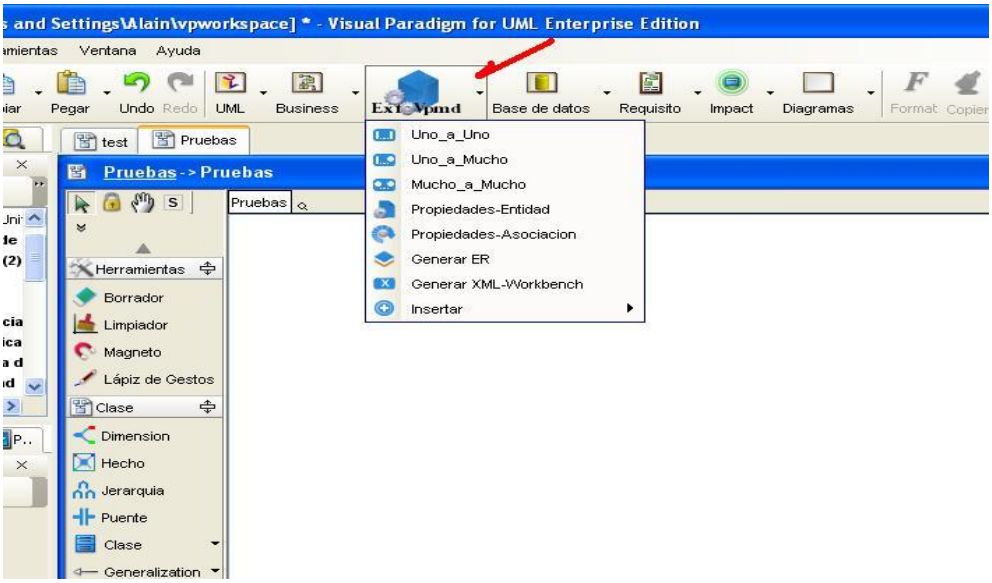

Figura # 15 Pruebas de funcionamiento realizadas a la aplicación 1.

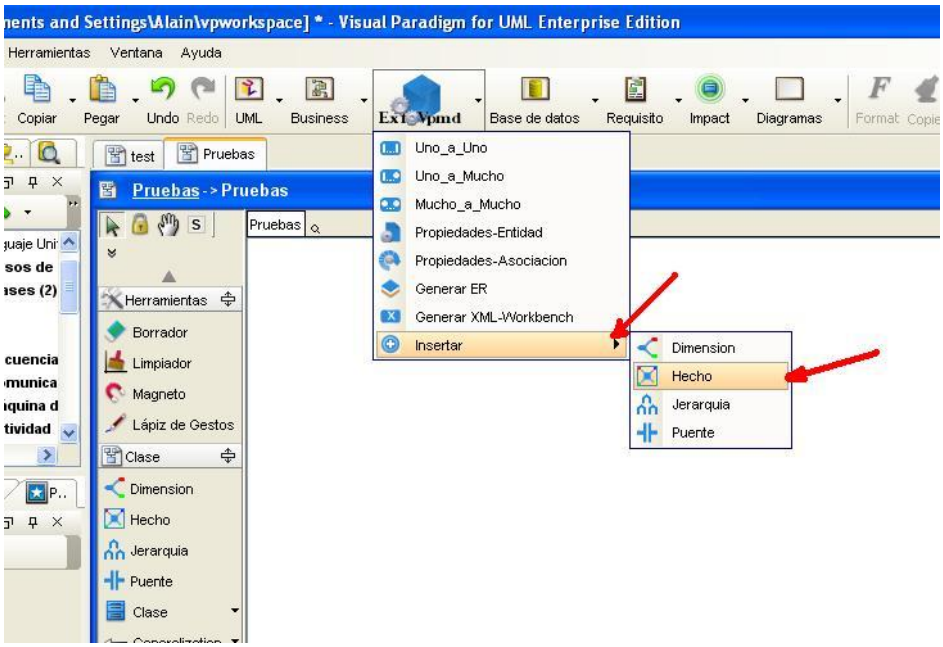

Figura # 16 Pruebas de funcionamiento realizadas a la aplicación 2.

En la siguiente imagen se observa la excepción lanzada por un error en la entrada de datos.

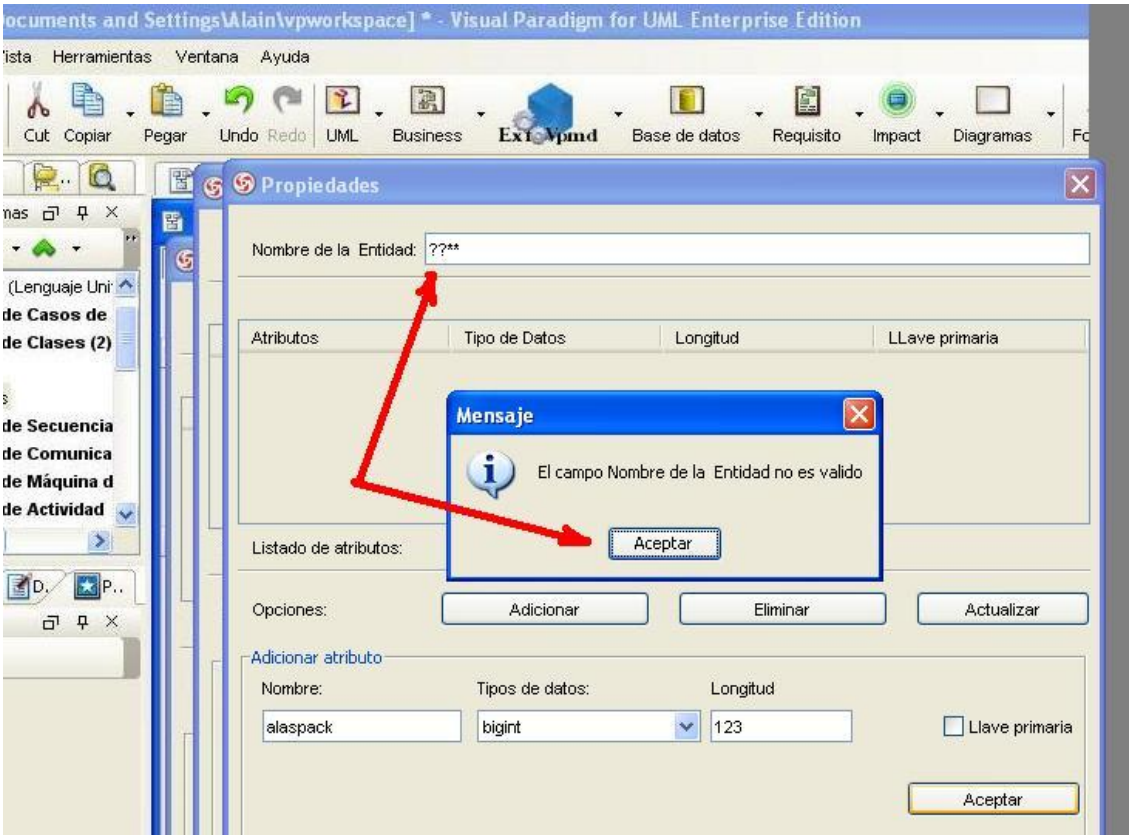

Figura # 17 Pruebas de funcionamiento realizadas a la aplicación 3.

Se realizaron las pruebas de funcionamiento del software para detectar no conformidades en la Extensión para modelar estructuras de datos dimensionales, las mismas fueron realizadas por el grupo de calidad interno del departamento Almacenes de Datos del Centro DATEC. Durante las pruebas se detectaron seis no conformidades que son descritas a continuación:

1. El icono de la extensión tiene un texto no entendible por lo que se debe cambiar el tipo de letra del icono y poner las siglas MDM en el mismo. En la descripción que se muestra cuando el cursor esta sobre el icono de la extensión se debe mostrar “Extensión para el modelado dimensional”. Esta se calificó de no significativa.

2. No se encuentra estandarizado el nombre en el componente para la creación de un hecho. Se debe cambiar la etiqueta (label) del componente para crear una entidad de tipo hecho, la etiqueta actual aparece como hecho y debe cambiarse por hech. Esta se calificó de no significativa.

3. Hay iconos en inglés y en español. El idioma debe ser homogenizado y en caso de ser español debe tener las tildes en las palabras. Esta se calificó de significativa.

4. Cuando se elimina una relación, las llaves que pasan de una entidad a otra, producto a la relación no se eliminan. Deben eliminarse estas llaves para evitar atributos duplicados. Esta se calificó de recomendación.

5. Los tipos de relaciones a crear están junto a los demás componentes. El menú de las relaciones debe estar agrupado en un componente que las reúna a todas. Esta se calificó de no significativa.

6. La opción del menú Insertar debe ponerse de primera según el orden lógico de las actividades que se realizan en la aplicación. Esta se calificó de no significativa.

Se le dio solución a las cinco no conformidades encontradas en la primera iteración, y la recomendación por la complejidad que requiere su desarrollo se deja para futuras versiones del producto. No se detectaron no conformidades en las pruebas de regresión, por lo que se le realizó una segunda iteración donde no se detectaron nuevas no conformidades, emitiéndose finalmente la carta de liberación por parte del jefe del grupo de calidad del departamento de Almacenes de Datos. Posteriormente se aplicaron las pruebas de aceptación para validar que el producto cumpla con los requisitos del cliente emitiéndose por parte de este la carta de aceptación, lo que constata que el producto está listo para su utilización.

3.6 Conclusiones del capítulo 3

En el proceso de construcción de la solución se le dio cumplimiento a 51 tareas de desarrollo de la implementación en cuatro iteraciones cumpliendo con lo identificado en las cuatro historias de usuario definidas. Se le realizaron las pruebas de liberación y aceptación al producto, lográndose el acta de aceptación por parte del cliente, calificando la solución obtenida como satisfactoria.

4. CONCLUSIONES

Luego de finalizada la investigación e implementada la solución, se arriba a las siguientes conclusiones:

- ✓ Se realizó un estado del arte que permitió la selección adecuada de la metodología de desarrollo XP y la herramienta NetBeans 7.0. a utilizar en el desarrollo de la extensión.
- ✓ Se decidió extender UML a partir de la creación de un Perfil UML permitiendo a Visual Paradigm for UML extenderse e incorporar el dominio de diseño de estructuras dimensionales.
- ✓ Se identificaron cuatro historias de usuarios, para describir los requisitos funcionales de la extensión y seis tarjetas CRC que permitieron una correcta planificación en aras de satisfacer al cliente. Se diseñó un Perfil UML para extender el dominio de UML permitiéndole el modelado de estructuras dimensionales.
- ✓ Se definieron un total de 51 tareas de implementación que permitieron el correcto desarrollo de la extensión. Se le realizaron las pruebas de liberación y aceptación a la aplicación, lográndose el acta de aceptación por parte del cliente, calificando la solución obtenida como satisfactoria.

5. RECOMENDACIONES

- ✓ Utilizar el contenido de la investigación como base de referencia para el desarrollo de futuras extensiones de herramientas CASE.
- ✓ Aplicar la extensión en todas las instituciones nacionales donde se modelen AD utilizando la herramienta CASE Visual Paradigm for UML.

6. REFERENCIAS BIBLIOGRÁFICAS

1- BARCHINI, G. E. Métodos "I + D" de la Informática. Universidad Nacional de Santiago del Estero, 2006 [citado 16/10/2011].

2- ARTURO GÁMEZ BLANCO, M. C., MENDÍVIL LEYVA. Informática I [online]. [Estado de Sonora, Mexico]: Colegio de Bachilleres del Estado de Sonora, 2007 [citado 20/10/2011]. [Disponible en: http://es.scribd.com/doc/5101923/Informatica-1-Libro-de-apoyo-docente-Mexico-DGB-SEP].

3- OSVLADOAVALOS. LA DIVERSIDAD DE LOS OBJETOS [online]. http://es.scribd.com, 2010 [citado 25/10/2011]. [Disponible en: http://es.scribd.com/doc/25171397/1-La-Diversidad-de-Los-Objetos-1-1-Caracteristicas].

4- ORGANIZATIVOS, P. Q. S. N. M., [citado 1/11/2011]. [Disponible en:http://www.adrformacion.com/cursos/metod5s/leccion1/tutorial2.html].

5- OMG. *OMG Mission Statement*, 2001. [citado 5/11/2011] [Disponible en: http://www.omg.org/gettingstarted/gettingstartedindex.htm]

6- ORALLO, E. H. El Lenguaje Unificado de Modelado (UML) [online]. 2003 [citado 8/11/2011]. [Disponible en: http://es.scribd.com/doc/54817695/Act-a-Uml].

7- OMG. OMG Unified Modeling LanguageTM (OMG UML), Superstructure [online]. 2009 [citado10/11/2011]. [Disponible en: http://www.omg.org/spec/UML/2.2/Superstructure].

8- YENISLEYDIS LEDESMA RODRÍGUEZ, Y. B. R. Extensión de la herramienta "Visual Paradigm for UML" para el soporte al Desarrollo Dirigido por Modelos con Ext JS. Universidad de las Ciencias Informáticas, 2011 [citado 12/11/2011].

9- GARCIA-BUSTELO, B. C. P. Desarrollo ágil de Software con Arquitectura Dirigida por Modelos [online]. Universidad de Oviedo, 2007 [citado 14/11/2011]. [Disponible en.

10- GIACHETTI, G., MARÍN, BEATRÍZ Y PASTOR, OSCAR. Perfiles UML y Desarrollo Dirigido por Modelos: Desafíos y Soluciones para Utilizar UML como Lenguaje de Modelado., [citado 16/11/2011]. [Disponible. ISSN 1988–3455.

11- INMON, B. Building the Data Warehouse [online]. [New York]: Wiley Publishing, 2005 [citado 17/112011]. [Disponible en.

12- CWM. Common Warehouse Metamodel (CWM) Specification [online]. March 2003, 2003 [citado 18/11/2011]. [Disponible en.

13- VALLECILLO, L. F. Y. A. Una Introducción a los Perfiles UML [online]. Universidad de Málaga, 2003 [citado 19/11/2011]. [Disponible en.

14- ER/STUDIO. ER/Studio, Modelado de datos Empresarial [online]. [San Francisco]: Embarcadaro Technologies, 2009 [citado 20/11/2011]. [Disponible en: www.embarcadero.com].

15- IBM. *IBM InfoSphere Data Architect*. [citado 21/11/2011] [Disponible en: www.ibm.com]

16- ORACLE. *Oracle SQL Developer Data Modelling*, Oracle Corporation, 2009.[citado 23/11/2011] [Disponible en: http://www.oracle.com/technology/products/database/datamodeler/index.html]

17- BARZANALLANA, R. *Informatica Aplicada a la Gestion Publica. IAGP, Metodologías de desarrollo de software*, Universidad de Murcia, 2005. [citado 24/11/2011] [Disponible en: http://www.um.es/docencia/barzana/IAGP/Iagp2.html]

18- INFORMÁTICOS, D. D. L. Y. S. *Rational Unified Process (Rup)* Edtion ed. Sevilla: Escuela Técnica Superior de Ingienería Informática, Universidad de Sevilla, 2006. 30 p [citado 25/11/2011].

19- TOROSSI, A. U. S. G. *El Proceso Unificado de Desarrollo de Software*. Edtion ed., 2003. 54 p [citado 26/11/2011].

20- MOLPECERES, A. *Proceso de desarrollo: RUP , XP y FDD*. Edtion ed., 2002. 11 p [citado 27/11/2011].

21- MARÍA A. MENDOZA SANCHEZ, I. I.-U. *Metodologías De Desarrollo De Software*. Edtion ed., 2004 [citado 28/11/2011].

22- ESCRIBANO, G. F. *Introducción a Extreme Programming*. Edtion ed., 2002 [citado 29/11/2011].

23- WIGAHLUK. Entre la XP y el RUP? 2007, [citado 30/11/2011]. [Disponible en:http://wigahluk.wordpress.com/2007/06/26/entre-la-xp-y-el-rup/].

24- FLORES, M. M. D. *RUP vs XP*. Edtion ed., 2006. 8 p [citado 2/12/2011].

25- GUERRERO, H. C. *El rol del Analista en RUP* 2007. 2011. [citado 4/12/2011] [Disponible en: http://hanzcocchi.net/el-rol-del-analista-en-rup/]

26- JOSÉ H. CANÓS, P. L. Y. C. P. Métodologías Ágiles en el Desarrollo de Software [online]. [Valencia]: DSIC -Universidad Politécnica de Valencia, 2008 [citado 8/12/2011]. [Disponible en.

27- HERNÁNDEZ, M. C. 2011. Historia de los lenguajes de programación. In *Proceedings of*, Universidad de las Ciencias Informáticas 2011 Historia de la Informática (HI) [citado 10/12/2011].

28- FERNÁNDEZ, O. B. Introducción al lenguaje de programación Java. [online]. 2004 [citado 12/12/2011]. [Disponible en.

29- *Definición de XML*, Definición.de, 2008-2011. [citado 13/12/2011] [Disponible en: http://definicion.de/categoria/tecnologia/Definición%de%XML%-%Qué%es,%Significado%y%Concepto.html]

30- GAYO, J. E. L. Lenguaje de XML. 2002, [citado 14/12/2011]. [Disponible en:http://www.topxml.com].

31- VISUAL-PARADIGM. 2002. [citado 15/12/2011] [Disponible en: http://www.visual-paradigm.com/product/vpsuite/]

32- NETBEANS.ORG. 2002. [citado 15/12/2011] [Disponible en: http://netbeans.org/index_es.html]

33- PARADIGM, V. *Visual Paradigm Suite Extensibility Plugin User´s Guide*. Edtion ed., 2007 [citado 16/12/2011].

34- RUBIA, J. M. Introducción a los almacenes de datos [online]. Marzo,2009 [citado 5/1/2012]. [Disponible en.

35- ERICH GAMMA, R. H., RALPH JOHNSON, JOHN VLISSIDES. Design Pattens-Elements of Reusable Object Oriented Software [online]. [citado 10/1/2012]. [Disponible en.

36- ANGELFIRE. *Patrones GOF*, 2011. [citado 8/2/2012] [Disponible en: http://geektheplanet.net/tag/gof-patterns]

37- Patrones del “Grang of Four ”. [online]. [España-Madrid]: Facultad de Informática-Universidad Politécnica de Madrid-Unidad, [citado 12/2/2012]. [Disponible en.

38- HOMMEL, S. Convenciones de Código para el lenguaje de programación,JAVA™ [online]. Sun Microsystems Inc, 2001 [citado 10/3/2012]. [Disponible en: http://www.javahispano.com].

7. BIBLIOGRAFÍA

1- BARCHINI, G. E. Métodos "I + D" de la Informática. Universidad Nacional de Santiago del Estero, 2006 [citado 16/10/2011].

2- ARTURO GÁMEZ BLANCO, M. C., MENDÍVIL LEYVA. Informática I [online]. [Estado de Sonora, Mexico]: Colegio de Bachilleres del Estado de Sonora, 2007 [citado 20/10/2011]. [Disponible en: http://es.scribd.com/doc/5101923/Informatica-1-Libro-de-apoyo-docente-Mexico-DGB-SEP].

3- OSVLADOAVALOS. LA DIVERSIDAD DE LOS OBJETOS [online]. http://es.scribd.com, 2010 [citado 25/10/2011]. [Disponible en: http://es.scribd.com/doc/25171397/1-La-Diversidad-de-Los-Objetos-1-1-Caracteristicas].

4- ORGANIZATIVOS, P. Q. S. N. M., [citado 1/11/2011]. [Disponible en:http://www.adrformacion.com/cursos/metod5s/leccion1/tutorial2.html].

5- OMG. *OMG Mission Statement*, 2001. [citado 5/11/2011] [Disponible en: http://www.omg.org/gettingstarted/gettingstartedindex.htm]

6- ORALLO, E. H. El Lenguaje Unificado de Modelado (UML) [online]. 2003 [citado 8/11/2011]. [Disponible en: http://es.scribd.com/doc/54817695/Act-a-Uml].

7- OMG. OMG Unified Modeling LanguageTM (OMG UML), Superstructure [online]. 2009 [citado10/11/2011]. [Disponible en: http://www.omg.org/spec/UML/2.2/Superstructure].

8- YENISLEYDIS LEDESMA RODRÍGUEZ, Y. B. R. Extensión de la herramienta "Visual Paradigm for UML" para el soporte al Desarrollo Dirigido por Modelos con Ext JS. Universidad de las Ciencias Informáticas, 2011 [citado 12/11/2011].

9- GARCIA-BUSTELO, B. C. P. Desarrollo ágil de Software con Arquitectura Dirigida por Modelos [online]. Universidad de Oviedo, 2007 [citado 14/11/2011]. [Disponible en.

10- GIACHETTI, G., MARÍN, BEATRÍZ Y PASTOR, OSCAR. Perfiles UML y Desarrollo Dirigido por Modelos: Desafíos y Soluciones para Utilizar UML como Lenguaje de Modelado., [citado 16/11/2011]. [Disponible. ISSN 1988–3455.

11- INMON, B. Building the Data Warehouse [online]. [New York]: Wiley Publishing, 2005 [citado 17/112011]. [Disponible en.

12- CWM. Common Warehouse Metamodel (CWM) Specification [online]. March 2003, 2003 [citado 18/11/2011]. [Disponible en.

13- VALLECILLO, L. F. Y. A. Una Introducción a los Perfiles UML [online]. Universidad de Málaga, 2003 [citado 19/11/2011]. [Disponible en.

14- ER/STUDIO. ER/Studio, Modelado de datos Empresarial [online]. [San Francisco]: Embarcadaro Technologies, 2009 [citado 20/11/2011]. [Disponible en: www.embarcadero.com].

15- IBM. *IBM InfoSphere Data Architect*. [citado 21/11/2011] [Disponible en: www.ibm.com]

16- ORACLE. *Oracle SQL Developer Data Modelling*, Oracle Corporation, 2009.[citado 23/11/2011] [Disponible en: http://www.oracle.com/technology/products/database/datamodeler/index.html]

17- BARZANALLANA, R. *Informatica Aplicada a la Gestion Publica. IAGP, Metodologías de desarrollo de software*, Universidad de Murcia, 2005. [citado 24/11/2011] [Disponible en: http://www.um.es/docencia/barzana/IAGP/Iagp2.html]

18- INFORMÁTICOS, D. D. L. Y. S. *Rational Unified Process (Rup)* Edtion ed. Sevilla: Escuela Técnica Superior de Ingienería Informática, Universidad de Sevilla, 2006. 30 p [citado 25/11/2011].

19- TOROSSI, A. U. S. G. *El Proceso Unificado de Desarrollo de Software*. Edtion ed., 2003. 54 p [citado 26/11/2011].

20- MOLPECERES, A. *Proceso de desarrollo: RUP , XP y FDD*. Edtion ed., 2002. 11 p [citado 27/11/2011].

21- MARÍA A. MENDOZA SANCHEZ, I. I.-U. *Metodologías De Desarrollo De Software*. Edtion ed., 2004 [citado 28/11/2011].

22- ESCRIBANO, G. F. *Introducción a Extreme Programming*. Edtion ed., 2002 [citado 29/11/2011].

23- WIGAHLUK. Entre la XP y el RUP? 2007, [citado 30/11/2011]. [Disponible en:http://wigahluk.wordpress.com/2007/06/26/entre-la-xp-y-el-rup/].

24- FLORES, M. M. D. *RUP vs XP*. Edtion ed., 2006. 8 p [citado 2/12/2011].

25- GUERRERO, H. C. *El rol del Analista en RUP* 2007. 2011. [citado 4/12/2011] [Disponible en: http://hanzcocchi.net/el-rol-del-analista-en-rup/]

26- JOSÉ H. CANÓS, P. L. Y. C. P. Métodologías Ágiles en el Desarrollo de Software [online]. [Valencia]: DSIC -Universidad Politécnica de Valencia, 2008 [citado 8/12/2011]. [Disponible en.

27- HERNÁNDEZ, M. C. 2011. Historia de los lenguajes de programación. In *Proceedings of*, Universidad de las Ciencias Informáticas 2011 Historia de la Informática (HI) [citado 10/12/2011].

28- FERNÁNDEZ, O. B. Introducción al lenguaje de programación Java. [online]. 2004 [citado 12/12/2011]. [Disponible en.

29- *Definición de XML*, Definición.de, 2008-2011. [citado 13/12/2011] [Disponible en: http://definicion.de/categoria/tecnologia/Definición%de%XML%-%Qué%es,%Significado%y%Concepto.html]

30- GAYO, J. E. L. Lenguaje de XML. 2002, [citado 14/12/2011]. [Disponible en:http://www.topxml.com].

31- VISUAL-PARADIGM. 2002. [citado 15/12/2011] [Disponible en: http://www.visual-paradigm.com/product/vpsuite/]

32- NETBEANS.ORG. 2002. [citado 15/12/2011] [Disponible en: http://netbeans.org/index_es.html]

33- PARADIGM, V. *Visual Paradigm Suite Extensibility Plugin User´s Guide*. Edtion ed., 2007 [citado 16/12/2011].

34- RUBIA, J. M. Introducción a los almacenes de datos [online]. Marzo,2009 [citado 5/1/2012]. [Disponible en.

35- ERICH GAMMA, R. H., RALPH JOHNSON, JOHN VLISSIDES. Design Pattens-Elements of Reusable Object Oriented Software [online]. [citado 10/1/2012]. [Disponible en.

36- ANGELFIRE. *Patrones GOF*, 2011. [citado 8/2/2012] [Disponible en: http://geektheplanet.net/tag/gof-patterns]

37- Patrones del "Grang of Four ". [online]. [España-Madrid]: Facultad de Informática-Universidad Politécnica de Madrid-Unidad, [citado 12/2/2012]. [Disponible en.

38- HOMMEL, S. Convenciones de Código para el lenguaje de programación,JAVA™ [online]. Sun Microsystems Inc, 2001 [citado 10/3/2012]. [Disponible en: http://www.javahispano.com].

39- MANRUBIA DÍEZ, C. L., JUAN MANUEL. Desarrollo de Software Dirigido por Modelos [online]. FICYT, Diciembre 2006 [citado 16/10/2011]. [Disponible en.

40- DRA. CLAUDIA AND GIANDINI, D. R. Desarrollo de Software Dirigido por Modelos, conceptos teóricos y su aplicación práctica [online]. 2007 [citado 22/11/2011]. [Disponible en.

41- GROUP, O. M. Meta Object Facility (MOF) Specification [online]. 2002 [citado 16/10/2011]. [Disponible en.

42- GROUP, O. M. MDA Guide [online]. 2003 [citado 18/11/2011]. [Disponible en.

43- GROUP, O. M. UML 2.0 Infrastructure Specification [online]. 2003 [citado 13/10/2011]. [Disponible en.

44- GROUP, O. M. Object Constraint Language Specification [online]. 2009 [citado 15/10/2011]. [Disponible en.

45- GROUP, O. M. OMG Unified Modeling LanguageTM (OMG UML),Infrastructure [online]. 2009 [citado 17/11/2011]. [Disponible en: http://www.omg.org/spec/UML/2.2/Infrastructure.

46- GROUP, O. M. Common Warehouse Metamodel (CWM) Specification [online]. Marzo 2003 [citado 8/10/2011]. [Disponible en.

47- L. FUENTES, A. V. Y. J. M. T. Using UML Profiles for Documenting Web- Based Application Frameworks [online]. Annals of Software Engineering, 2002 [citado 10/10/2011]. [Disponible en.

48- CASTRO, M. M. A. Procedimiento para la realizacion de pruebas de unidad de software orientado por objetos a nivel de clases [online]. Departamento de Ciencias Computacionales, Universidad Autonoma de Manizales, Colombia Junio 2011 [citado 10/3/2012]. [Disponible en.

49- JOSKOWICZ, I. J. Reglas y Prácticas en eXtreme Programming [online]. Universidad de Vigo, España., Febrero 2008 [citado 12/3/2012]. [Disponible en.

8. GLOSARIO DE TÉRMINOS

Almacenes de Datos: Es una copia de las transacciones de datos específicamente estructurada para la consulta y el análisis, que es orientado a temas, variante en el tiempo, no volátil e integrado.

Casuística: Consideración de los diversos casos particulares que se pueden prever en determinada materia.

IDE: Es un entorno de desarrollo integrado, programa compuesto por un conjunto de herramientas para un programador desde el que se pueden editar programas, compilarlos y depurarlo.

Inteligencia de negocio: Conjunto de herramientas y estrategias enfocadas a la administración y creación de conocimiento mediante el análisis de datos existentes en una organización o empresa.

Metamodelos: o metamodelado en ingeniería de software y sistemas entre otras disciplinas, es el análisis, construcción y desarrollo de marcos, reglas, restricciones, modelos y teorías aplicables y útiles para el modelado de tipos de problemas específicos.

Metodología de Desarrollo: Marco de trabajo usado para estructurar, planificar y controlar el proceso de desarrollo en sistemas de información.

Modelo dimensional: Modelo de almacenes de datos o data warehouse.

Perfil UML: Mecanismo de extensión del dominio de UML, permite que UML modele objetos que no comprende en su estructura original.

Pruebas Spikes: pruebas que sirven para validar las decisiones arquitectónicas del software.

Refactorización: (del inglés Refactoring) es una técnica de la ingeniería de software para reestructurar un código fuente, alterando su estructura interna sin cambiar su comportamiento externo.

Script: archivo de órdenes o archivo de procesamiento por lotes.

Software: Equipamiento lógico o soporte lógico de un ordenador. Comprende el conjunto de los componentes lógicos necesarios para hacer posible la realización de una tarea específica, en

contraposición a los componentes físicos del sistema (hardware). Suele ser utilizado para referirse a los programas y aplicaciones informáticas.

UML: De sus siglas en inglés Unified Modelling Language, Lenguaje Unificado de Modelado es un lenguaje de modelado visual que permite visualizar, especificar, construir y documentar artefactos de un sistema de software, permite realizar modelos para la guía del proceso de desarrollo del software.

9. ANEXOS

Anexo 1. Ejemplo de la implementación del archivo Plugin.xml

```
<plugin
id="sample.plugin"
name="Sample Plugin"
description="Sample Plugin"
provider="Visual Paradigm"
class="sample.plugin.SamplePlugin">
	<runtime>
		<library path="lib/sampleplugin.jar" relativePath="true"/>
		<library path="lib/AbsoluteLayout.jar" relativePath= "true"></library>
	</runtime>
	<actionSets>
		<actionSet id="sample.plugin.actions.ActionSet1">
			<toolbar
				id="sample.plugin.actions.Toolbar1"
				orientation="north"
				index="last"/>
			<menu
				id="sample.plugin.actions.Menu1"
				label="Sample Menu 1"
				mnemonic="M"
				menuPath="Tools/Report"/>
			<action
				id="sample.plugin.actions.Action1"
				actionType="generalAction"
				label="Sample Action 1"
				tooltip="Sample Action 1"
				icon="icons/red.png"
				style="normal"
				menuPath="Tools/Report"
				toolbarPath="sample.plugin.actions.Toolbar1/#"
				actionController class="sample.plugin.actions.ActionController"/>
			</action>
```

```
                <separator
                        id="sample.plugin.actions.Separator1"
                        menuPath="Tools/sample.plugin.actions.Action1"
                        toolbarPath="sample.plugin.actions.Toolbar1/sample.plugin.action.Action1"/>
                </actionSet>
        <contextSensitiveActionSet id="sample.plugin.actions.ActionSet2">
                <contextTypes all="false">
                <include type="Class"/>
                <exclude type="Package"/>
                </contextTypes>
                        <action
                                id="sample.plugin.actions.ContextAction1"
                                label="Sample Action [1]"
                                icon="icons/blue.png"
                                style="toggle"
                                menuPath="OpenSpecification">
                                actionController class="sample.plugin.actions.ContextActionController"/>
                        </action>
                </contextSensitiveActionSet>
        </actionSets>
</plugin>
```

Anexo 2. Casos de pruebas de aceptación.

Tabla 1: Prueba 2 de la HU "Crear diagrama de estructura dimensional."

Caso de prueba de aceptación	
Código: HU1_p2	**Historia de Usuario:** 1
Nombre: Crear diagrama de estructura dimensional.	
Nombre de la persona que realiza la prueba: José Salvador Bermúdez Rodríguez.	
Descripción de la Prueba: Probar que el sistema permita advertir al usuario de malas prácticas de diseño en el modelado de estructuras dimensionales.	

Condiciones de ejecución: El usuario debe ejecutar el Visual Paradigm for UML con la extensión previamente incorporada.
Entrada/ pasos de ejecución: En Visual Paradigm for UML en la sección dedicada al modelado de estructuras dimensionales el usuario debe seleccionar para su representación 2 o más hechos y relacionarlos entre sí, así como definir sus atributos y jerarquías.
Resultado esperado: El sistema muestra al usuario que la relación entre un hecho y otro hecho es un ejemplo de mala práctica de diseño, y le dará la opción al usuario de mantener la relación como está o crear una tabla puente entre los dos hechos .
Evaluación de la prueba: Satisfactoria

Tabla 2: Prueba 3 de la HU "Crear diagrama de estructura dimensional."

Caso de prueba de aceptación	
Código: HU1_p3	**Historia de Usuario:** 1
Nombre: Crear diagrama de estructura dimensional.	
Nombre de la persona que realiza la prueba: José Salvador Bermúdez Rodríguez.	
Descripción de la Prueba: Probar que el sistema permita la persistencia de los datos, guardar y cargar el modelo dimensional para futuros usos o modificaciones.	
Condiciones de ejecución: El usuario debe ejecutar el Visual Paradigm for UML con la extensión previamente incorporada.	
Entrada/ pasos de ejecución: En Visual Paradigm for UML en la sección dedicada al modelado de estructuras dimensionales el usuario debe seleccionar para su representación 2 o más hechos y relacionarlos entre sí, así como definir sus atributos y jerarquías. Posteriormente debe guardar el proyecto y cargarlo	

nuevamente.
Resultado esperado: El sistema guarda y carga el proyecto correctamente sin modificación ni perdida de datos.
Evaluación de la prueba: Satisfactoria.

Tabla 3: Prueba 1 de la HU "Transformar las estructuras dimensionales en modelos Entidad-Relación."

Caso de prueba de aceptación	
Código: HU2_p1	**Historia de Usuario:** 2
Nombre: Transformar las estructuras dimensionales en modelos Entidad-Relación.	
Nombre de la persona que realiza la prueba: José Salvador Bermúdez Rodríguez.	
Descripción de la Prueba: Probar que el sistema permita transformar las estructuras dimensionales en modelos Entidad-Relación aplicando las reglas de transformación.	
Condiciones de ejecución: El usuario debe ejecutar el Visual Paradigm for UML con la extensión previamente incorporada. Y haber modelado una estructura dimensional.	
Entrada/ pasos de ejecución: En Visual Paradigm for UML en la sección Ext Vpmd estructuras dimensionales el usuario debe seleccionar Generar ER.	
Resultado esperado: El sistema muestra al usuario un modelo Entidad-Relación con las reglas de transformación correctamente aplicadas.	
Evaluación de la prueba: Satisfactoria	

Tabla 4: Prueba 2 de la HU "Transformar las estructuras dimensionales en modelos Entidad-Relación."

Caso de prueba de aceptación	
Código: HU2_p2	**Historia de Usuario:** 2
Nombre: Transformar las estructuras dimensionales en modelos Entidad-Relación.	
Nombre de la persona que realiza la prueba: José Salvador Bermúdez Rodríguez.	
Descripción de la Prueba: Comprobar que las jerarquías no pasen al modelo Entidad-Relación como tablas independientes, ya que en el modelo, por restricciones de UML se utilizan entidades para representar las jerarquías.	
Condiciones de ejecución: El usuario debe ejecutar el Visual Paradigm for UML con la extensión previamente incorporada. Y haber modelado una estructura dimensional.	
Entrada/ pasos de ejecución: En Visual Paradigm for UML en la sección Ext Vpmd estructuras dimensionales el usuario debe seleccionar Generar ER.	
Resultado esperado: En el modelo Entidad-Relación las jerarquías deben estar como atributos de la tabla hecho con la que estaban relacionados en el modelo multidimensional.	
Evaluación de la prueba: Satisfactoria	

Tabla 5: Prueba 1 de la HU "Generar fichero XML para el Pentaho Schema Workbench."

Caso de prueba de aceptación	
Código: HU3_p1	**Historia de Usuario:** 3
Nombre: Generar fichero XML para el Pentaho Schema Workbench.	
Nombre de la persona que realiza la prueba: José Salvador Bermúdez Rodríguez.	
Descripción de la Prueba: Probar que el sistema es capaz de una vez diseñado y obtenido el modelo dimensional, así como el modelo Entidad-Relación se pueda	

exportar el mismo para ser utilizado por el área de BI con el Pentaho Schema Workbench a través de un fichero XML.
Condiciones de ejecución: El usuario debe ejecutar el Visual Paradigm for UML con la extensión previamente incorporada. Haber modelado una estructura dimensional y transformado las estructuras dimensionales al modelo Entidad-Relación.
Entrada/ pasos de ejecución: En Visual Paradigm for UML en la sección Ext Vpmd estructuras dimensionales el usuario debe seleccionar Generar XML-Workbench. Abrir el Pentaho Schema Workbench y seleccionar Open e importar el fichero XML previamente generado.
Resultado esperado: El sistema debe generar un fichero XML compatible con Pentaho Schema Workbench. Al importar el fichero no debe lanzar ningún error el Pentaho Schema Workbench.
Evaluación de la prueba: Satisfactoria

Tabla 6: Prueba 1 de la HU "Crear script para exportar a la base de datos."

Caso de prueba de aceptación	
Código: HU4_p1	**Historia de Usuario:** 4
Nombre: Generar fichero XML para el Pentaho Schema Workbench.	
Nombre de la persona que realiza la prueba: [Nombre y apellidos.]	
Descripción de la Prueba: Probar que el sistema es capaz de una vez diseñado y obtenido el modelo dimensional, así como el modelo Entidad-Relación se pueda exportar el mismo para ser utilizado por el área de BI con el Pentaho Schema Workbench a través de un fichero XML.	
Condiciones de ejecución: El usuario debe ejecutar el Visual Paradigm for UML con la extensión previamente incorporada. Haber modelado una estructura	

dimensional y transformado las estructuras dimensionales al modelo Entidad-Relación.
Entrada/ pasos de ejecución: En Visual Paradigm for UML en la sección Ext Vpmd estructuras dimensionales el usuario debe seleccionar Generar XML-Workbench. Abrir el Pentaho Schema Workbench y seleccionar Open e importar el fichero XML previamente generado.
Resultado esperado: El sistema debe generar un fichero XML compatible con Pentaho Schema Workbench. Al importar el fichero no debe lanzar ningún error el Pentaho Schema Workbench.
Evaluación de la prueba: Satisfactoria