UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

FACULTAD 6

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

EXTENSIÓN PARA MODELAR ALMACENES DE DATOS EN LA HERRAMIENTA "VISUAL PARADIGM FOR UML", VERSIÓN 2.0.

Autores:

Glennis Emilia Díaz del Toro.

Jessie Cruz Pérez.

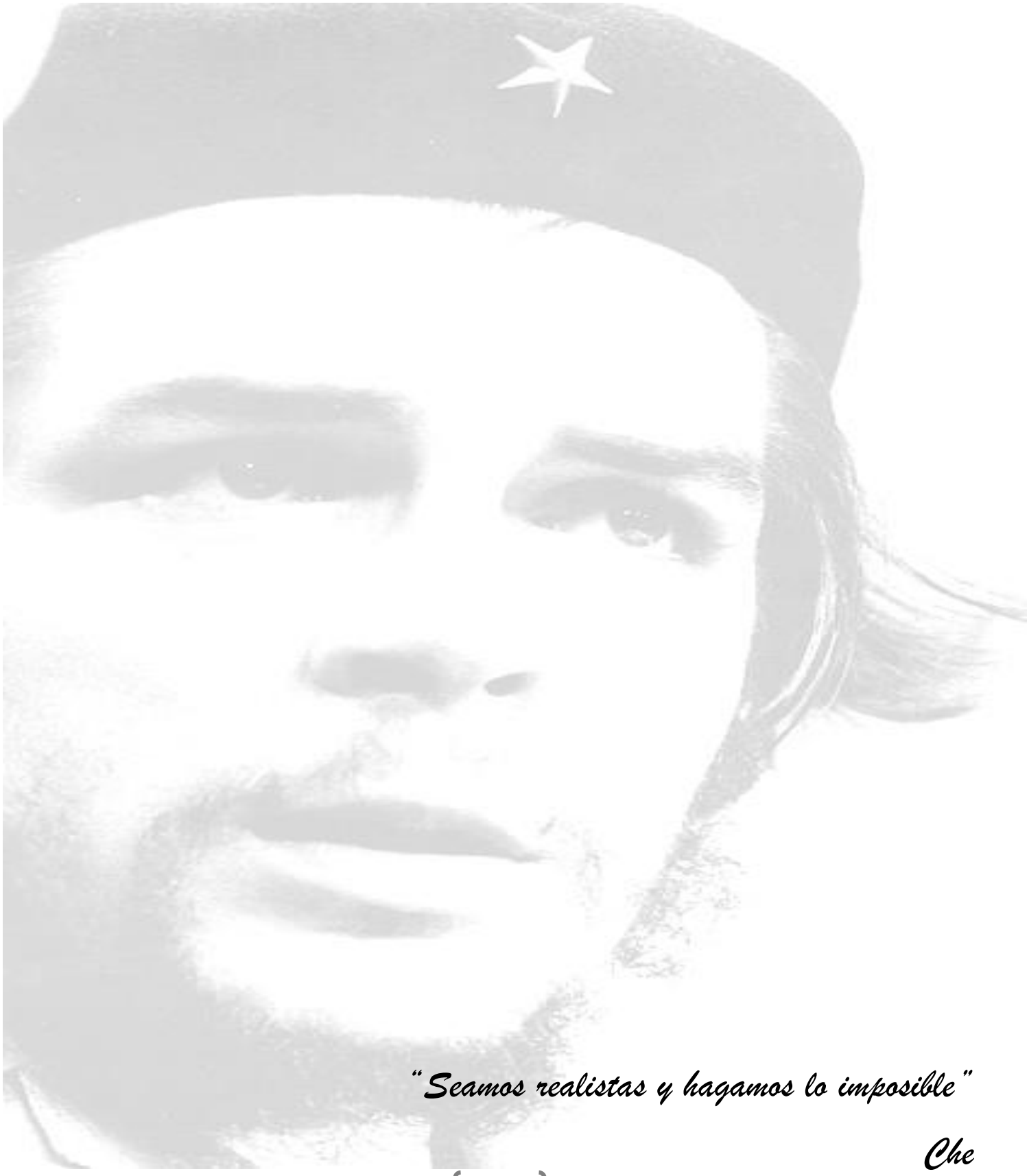
Tutores:

Ing. Idalmys Maza Capote.

Ing. Marisel Santana Rodríguez.

La Habana, junio 2015.

"Año 57 de la Revolución".



"Seamos realistas y hagamos lo imposible"

Che

Declaramos ser los autores de la presente tesis Extensión para modelar soluciones de Almacenes de Datos para la herramienta Visual Paradigm for UML, versión 2.0 y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

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

Glennis Emilia Díaz del Toro	Jessie Cruz Pérez
_____	_____
Firma del autor	Firma del autor
Ing. Idalmys Maza Capote	Ing. Marisel Santana Rodríguez
_____	_____
Firma del tutor	Firma del tutor

El éxito no significa nada si no tienes con quien compartirlo, por eso quiero agradecerle a todo aquel que hizo posible este sueño transformado éxito.

A mis abuelas por ser mi razón de ser, por darme todo su amor y comprensión a lo largo de mi vida, por enseñarme a crecer y darme un voto de confianza cada día.

A mis padres por estar ahí siempre para mí, por escucharme y quererme como una BB todavía.

A mis hadas madrinas mis tías Rosita y Tereza por el cariño y apoyó en todo momento de forma incondicional.

A mi tío Rubén y tía Onylis por quererme y tratarme como una hija para ellos.

A mis hermanos Charly Brawm, Ríos Pérez y mi gordito rico Kenny por ser unas personitas muy importantes para mí, por quererme y esperarme siempre aunque lleve tanto tiempo lejos de ellos.

A toda mi familia en general, mi tío Miguel y su esposa Lidia por sus consejos, sin ellos no estaría aquí, mi tía Teresa la vieja por su apoyo, a todos los que de una manera u otra me ayudaron.

A mi novio Rubensín por esperar por mí todo este tiempo, por ayudarme a resistir en los momentos difíciles, por demostrarme su querer cada día.

A mi familia UCI Yurima, Rosi, Nara, Yudelkis, Luis, Arodys por cada momento feliz juntos y por la ayuda bridada a lo largo de la carrera ya que han sido un apoyo muy especial para mí.

A todas las niñas de la casa Sonia, Yailin, Laro, Aryanna, Yanet incluyendo a Carlitos por la convivencia y los recuerdos inolvidables.

A mis amigos Jito y Yisell por estar siempre presente en las buenas y malas, siempre que necesité consejos y abrirme los ojos, siempre que necesité despejar y salir de la rutina.

A mi amiga y dúo de tesis Emilia por escuchar cada una de mis reflexiones y no juzgarme o asombrarse, por aconsejarme (aunque la escuche poco), por hacerme sentir persona sin importar lo distinta que fuera, por enseñarme a andar en el camino de los grandes (aunque todavía gateo) y por estar siempre ahí.

A todas amistades UCI adquiridas en el transcurso de la carrera, que de alguna manera me ayudaron a ser mejor persona, siempre los recordaré por haberme hecho más amena la estancia en la universidad pues no hay nada más agradable que tener en quien confiar y desahogar en momentos difíciles y de estrés: Osiel, Humberto, Morgado, Franki, Quinta, Fumita, Yera, Hanser, Ernesto, Joakin, Daniel (mon ami), Johan, a todos los varones del aula, a Rolyen y a muchos más amistades.

A mi amistades viejas Idelys, las negras (Celia y Irene), Yuli, Anabel, la gorda (Liset) que a pesar de la distancia me tienen vigente.

A mis tutoras y a la oponente por su apoyo, su comprensión, y paciencia.

De Jessie Cruz Pérez.

El éxito no le sale al paso por suerte ni por casualidad, siempre hay una mano amiga y en mi camino por la UCI, hay muchas que han hecho posible este sueño.

Agradezco a mi mamá, porque siempre me aconsejó y me mostró el camino a seguir, porque sé que cuando la vida sea muy dura y me sea difícil continuar, tengo en ella un refugio al que puedo regresar. Gracias mamá por ser mi ángel y por darme tu amor infinito, por detenerme cuando debías y por empujarme cuando tenía miedo de seguir mis sueños.

Agradezco a mi papá porque siempre me permitió contar con él, a pesar de que estos últimos años fueron difíciles en su vida personal, siempre me demostró que en la vida todo lo que queremos se logra, gracias por estar aquí conmigo disfrutando de este éxito.

Agradezco a mi familia por parte de madre que siempre conté con su apoyo: a mis primos Osmany, Osmel y Dunieksy que han sido un ejemplo a seguir de abnegación y sacrificio, pero en especial a mi tía Rosy que supo crecerse ante las dificultades y gracias a ella pude terminar mi quinto año con su apoyo infinito y sin lamentarse hacia mi mami preciosa.

Agradezco a mi familia por parte de padre por el apoyo que en un momento determinado me brindaron, en especial a mi tía Esperanza que siempre pude contar con ella y me dio muchos consejos para la vida, a mi hermanita Lia por su cariño infinito.

Agradezco a esa familia UCI que siempre me apoyó y me brindaron amor en todo momento, a la madre Yurima y las tías Rosy, Yudelkis, Nara y al hermano Luis que siempre nos brindó su ayuda incondicional en todo momento.

Agradezco a mi amiga fiel y dúo de tesis Jessie, por siempre tener una palabra de aliento, por siempre tenderme una mano y por tener una sonrisa para mí cuando estaba triste.

Agradezco a tres personitas que siempre estuvieron ahí cuando las necesitaba, me brindaron amor y ese abrazo de aliento para seguir adelante Yisell, Humberto y el Guajiro.

Agradezco a las niñas del apartamento Laro, Sonia, Yailín, Aryanna, Yanet y a Carlito que era parte de nosotras.

Agradezco a todos mis amistades que aunque esté lejos, mi corazón nunca olvidará que somos amigos y que estamos unidos por miles de aventuras y desafíos que supimos vencer juntos, en especial a Felo Dashiel, Jorgito (el bicho), Gabriel, Hanser, Joaquín, Morgado, el pollo, Karel, Jorgito, el Fuma, el Yera, Alain, Frank, Yainel, Inza, el Yasel, Ernesto, Gretel, Claudia, Zayli, Franki, Daniel (el flaquito bello) y muchos más.

Agradezco a mis tutoras y a la oponente Amarelys en especial por su ayuda, paciencia y dedicación.

A todos muchas gracias

De Glennis Emilia.

El éxito se adquiere a través del sacrificio persiguiéndolo con el apoyo de aquellas personas que confían en ti y que esperan que lo logres. Este sueño transformado en éxito, quiero dedicárselo:

A mi abuelo Rigo, aunque no esté presente, que Dios me lo tenga en la Gloria, nunca lo olvidaré pues fue un padre más para mí, supo siempre que llegaría a cumplir este mérito.

A mis hermanos y a mi primo por ser ahora los más pequeños de la casa y motivarlos a que sigan adelante, que si se puede que me tendrán siempre de apoyo y ayuda para ellos.

A una personita muy especial, que sin ella nada de esto hubiese salido, a mi dúo de tesis y amiga Emilia por ser muy grande a pesar de su tamaño.

De Jessie Cruz

El éxito no le sale al paso por suerte ni por casualidad. Esto se concibe, se prepara, se ejercita, y después recién se realiza.

Dedico este éxito a mi mamá Yudania, que cuando no tenía a quien acudir, sabía que podía contar con ella, que cuando todos los caminos se cerraban, su puerta era la única que siempre estaba abierta. Y cuando todo se ponía difícil ahí estaba a mi lado diciéndome que todo saldría bien.

A mi tía Nilva, que Dios la tenga en la gloria por confiar en mí y porque aún enferma me dijo que si se podía.

A mi hermana Lia exhortándola, para que siga mis pasos, pues todo sueño se puede convertir en realidad siempre y cuando creas que tú puedes.

A la persona que hizo más que nadie mi estancia placentera aquí en la UCI mi gran amiga Jessie Cruz, mi dúo de tesis.

De Glennis Emilia

Resumen

El uso de los Almacenes de Datos ha alcanzado un gran auge en las empresas o instituciones, ya que constituyen uno de los soportes fundamentales para el proceso de toma de decisiones. Para el modelado y creación de los almacenes se utilizan un conjunto de herramientas que son independientes y no se integran, por lo que es necesario que los especialistas manejen diferentes tecnologías para desarrollar su trabajo. En busca de una alternativa, se desarrolló una extensión para la herramienta Visual Paradigm, que permite el trabajo con las estructuras dimensionales y su publicación en el Pentaho BI Server. A partir de la necesidad de incorporarle a la extensión nuevas funcionalidades para realizar un modelo dimensional completo; el presente trabajo tiene como objetivo general, desarrollar la versión 2.0 de la extensión para modelar soluciones de Almacenes de Datos, que permita la virtualización de los cubos y los miembros calculables en el modelo de datos dimensional. El ciclo de vida de la extensión estuvo guiado por la metodología de desarrollo OpenUp. Para la implementación se utilizó el lenguaje de programación Java, en el Entorno de Desarrollo Integrado NetBeans 8.0, haciendo uso de las librerías OpenAPI y MondrianGUI. La calidad de la extensión se garantizó mediante pruebas funcionales y de aceptación. Con las nuevas funcionalidades incorporadas se provee a los especialistas de una solución donde puedan realizar el modelo lógico y el modelo dimensional de un Almacén de Datos, con el uso de la herramienta Visual Paradigm.

Palabras Claves: Almacenes de Datos, modelado, modelo dimensional, virtualización

Abstract

The use of data warehouses has reached a great boom in companies or institutions as they are one of the fundamental supports for the making decisions process. For the modeling and creation of the warehouses a set of tools which are independents are used and there is not any integration among them, so that is necessary the specialists hanlde different technologies to develop their work. In order to find a choice for that, an extension for the Visual Paradigm tool was developed. It allows working with the dimensional structures and its publishing on the Pentaho BI Server. From the need of adding new functionalities to the extension in order to carry out a complete dimensional model, the current paper has as a general objective, to develop the version 2.0 of the Extension to model data warehouses solutions. Besides it could be able to allow the cubes virtualization and the calculable members in the dimensional data model. The extension life cycle was guided for the OpenUp development methodology. To the implementation the Java

programming language was used, NetBeans 8.0 in the Integrated Development Enviroment. For this, the OpenAPI and MondrianGUI libraries were used. The extension quality was guaranteed by means of functional and acceptance tests. The new added functionalities provides a solution to the specialists where they can make the logic and dimensional model of a Data Warehouse using the Visual Paradigm tool.

***Keywords**: Data warehouse, modeling, dimensional model, virtualization*

Introducción

En los últimos años, ha existido un gran crecimiento en la capacidad del ser humano para generar y agrupar datos, debido básicamente al enorme poder de procesamiento de las máquinas, así como a su bajo costo de almacenamiento. Como resultado de este proceso, el almacenamiento de datos ha pasado a un nivel superior en la sociedad actual, trayendo consigo el surgimiento de los Almacenes de Datos (AD).

Un AD es un repositorio de datos de fácil acceso, que se caracteriza por integrar y depurar información de una o más fuentes para luego procesarla, permitiendo su análisis desde múltiples perspectivas. En la actualidad las empresas e instituciones apuestan por esta tecnología para el almacenamiento de grandes volúmenes de información. Ejemplo de esto lo constituye la Universidad de las Ciencias Informáticas (UCI), centro creado para producir aplicaciones y servicios informáticos, a partir de la vinculación estudio-trabajo como modelo de formación; así como servir de soporte a la industria cubana de la informática. El modelo productivo de la UCI está estructurado en centros de desarrollo, vinculados directamente a las facultades, con temáticas afines y especializados en diferentes áreas de investigación y desarrollo, entre los que se encuentra el Centro de Tecnologías de Gestión de Datos (DATEC) perteneciente a la Facultad 6, donde se desarrollan soluciones integrales de almacenamiento y análisis de datos.

En este centro, para confeccionar el modelo lógico de un AD se utiliza la herramienta CASE[1] Visual Paradigm for UML[2] (VP) *Community Edition*; herramienta profesional que guía el análisis y diseño de las soluciones, permitiendo la obtención de las vistas de las entidades con sus respectivos atributos y las relaciones existentes entre ellas, para mayor claridad y entendimiento. La herramienta Schema Workbench es una herramienta gráfica de diseño que admite la construcción visual de los esquemas de cubos OLAP[3], además los publica en el servidor de inteligencia del negocio para que puedan ser utilizados

[1] **CASE** ingeniería de software asistida por computadora (acrónimo CASE por sus siglas en inglés ***C****omputer* ***A****ided* ***S****oftware* ***E****ngineering*).

[2] **UML**: Lenguaje Unificado de Modelado (acrónimo UML por sus siglas en inglés *Unified Modeling Language*); es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema.

[3] **OLAP** es el acrónimo en inglés de procesamiento analítico en línea (On-Line *Analytical Processing*). Es una solución utilizada en el campo de la llamada Inteligencia empresarial (o *Business Intelligence*) cuyo objetivo es agilizar la consulta de grandes cantidades de datos.

en los análisis por los usuarios de la plataforma para decisiones futuras; en dicho centro es utilizada para la elaboración del modelo dimensional.

Estas dos herramientas de trabajo son independientes, no existe una integración entre ellas, por lo que el analista del AD después de crear el modelo lógico, crea manualmente el modelo dimensional siendo engorrosa dicha tarea. En busca de mejorar esta situación existente, se crea como alternativa una extensión para modelar soluciones de AD en la herramienta de modelado VP, la cual apoya el proceso de análisis y diseño de los mismos, teniendo en cuenta las estructuras de datos dimensionales, el diseño de los cubos (hechos), dimensiones y medidas. Dicha extensión permite publicar en la herramienta Pentaho BI Server el esquema con el cubo generado, como resultado del trabajo de diploma (Mariño, 2014).

Hoy en día a pesar de la creación de la extensión para el VP, los especialistas del centro no hacen uso de la misma ya que aún no se puede modelar en ella un modelo dimensional completo, pues no se tiene en cuenta la virtualización de los cubos. Esto trae consigo que no se realice el análisis de la información que no está relacionada, aunque para el cliente resulte de vital importancia su estudio de forma conjunta desde diferentes perspectivas. Además, no permite el trabajo con los miembros calculables, imposibilitando el manejo de los datos derivados. En cuanto a los roles, carece de la jerarquía de usuario por lo que no es posible definir los niveles de seguridad por los miembros del esquema.

A partir de la situación descrita se puede identificar el siguiente **problema de la investigación:** ¿Cómo lograr la virtualización del modelo de datos dimensional y los miembros calculables para soluciones de AD en la herramienta VP?

El problema descrito centra el **objeto de estudio** en los modelos dimensionales para el proceso de análisis y diseño de soluciones de AD, enmarcado en el **campo de acción**: los modelos dimensionales para el proceso de análisis y diseño de soluciones de AD utilizando la herramienta de modelado VP.

Con la finalidad de darle cumplimiento al problema de la investigación se precisa el siguiente **objetivo general**: Desarrollar la versión 2.0 de la extensión para modelar AD en la herramienta VP, que permita la virtualización de los cubos en el modelo de datos dimensional y los miembros calculables.

Para darle cumplimiento al objetivo general planteado se realizarán las siguientes tareas:

1. Fundamentación teórica y metodológica de la extensión para modelar soluciones de AD en la herramienta de modelado VP, en su versión 2.0, para el mejor entendimiento de la problemática que se plantea.
2. Selección de las herramientas, tecnologías y metodología a utilizar en el desarrollo de la extensión para modelar soluciones de AD en la herramienta de modelado VP, en su versión 2.0, para el desarrollo de la extensión.
3. Definición de las principales funcionalidades a incluir en la extensión para modelar soluciones de AD en la herramienta de modelado VP, en su versión 2.0, para la virtualización del modelo de datos dimensional y los miembros calculables.
4. Diseño de las funcionalidades a incluir en la extensión para modelar soluciones de AD en la herramienta de modelado VP, en su versión 2.0, para una correcta implementación.
5. Descripción de la estructura base de la extensión para modelar soluciones de AD en la herramienta VP, en su versión 2.0, para guiar la construcción del mismo.
6. Selección de los patrones de diseño a emplear en el desarrollo de la extensión para modelar soluciones de AD en la herramienta de modelado VP, en su versión 2.0, para el correcto diseño de objetos.
7. Implementación de la extensión para modelar soluciones de AD en la herramienta de modelado VP, en su versión 2.0, para lograr un correcto funcionamiento de los elementos definidos a incluir en la extensión.
8. Realización de pruebas funcionales a la extensión para modelar soluciones de AD en la herramienta de modelado VP, en su versión 2.0, para comprobar el correcto funcionamiento de los elementos definidos a incluir en la extensión.

Para guiar las respuestas que se buscan con la presente investigación se identificaron las siguientes **preguntas científicas**:

- ¿Cuáles son las bases teóricas y metodológicas que fundamentan el diseño del modelo dimensional?
- ¿Cuáles son las características y capacidades que debe tener la versión 2.0 de la extensión para modelar soluciones de AD en la herramienta de modelado VP?
- ¿Cómo lograr la implementación de la extensión para modelar soluciones de AD en la herramienta de modelado VP en su versión 2.0?

- ¿Cómo lograr la integración de la extensión con la herramienta de modelado VP?
- ¿Cómo validar el correcto funcionamiento de la extensión para modelar soluciones de AD en la herramienta de modelado VP?

Durante el desarrollo de la investigación se utilizan métodos científicos que guíen la misma. A continuación se especifica el motivo de la selección.

Métodos teóricos:

- **Método Analítico – Sintético:** utilizado para realizar un estudio del estado del arte del modelado de AD en la herramienta VP. Además permite definir las tecnologías y metodología que serán utilizadas y arribar a las conclusiones de la investigación, así como determinar las características de los requisitos a implementar.

Métodos empíricos:

- **Entrevista no estructurada**: se aplica para obtener información sobre el desarrollo de los AD en DATEC, además de tener en cuenta las extensiones desarrolladas para el VP y para así definir los requisitos de la extensión a desarrollar. Es realizada de forma individual a profesionales que están vinculados directamente con el desarrollo y evaluación de proyectos de este tipo, así como para caracterizar el objeto de estudio.
- **Revisión documental:** se utiliza en la revisión de la documentación oficial del expediente de proyecto de la extensión para modelar soluciones de AD en la herramienta de modelado VP. Permite obtener el conocimiento necesario en el desarrollo de la herramienta en su versión 2.0.
- **Análisis Estático**: permite el examen minucioso de la estructura de la extensión para realizar la inspección de la misma mediante pruebas con la finalidad de descubrir errores.

Estructura capitular

Capítulo I: Fundamentos Teóricos para el desarrollo de la extensión para modelar AD en la herramienta VP, versión 2.0.

En este capítulo se hará referencia al marco teórico y metodológico de la investigación donde se realiza un estudio del estado del arte. Además, se describen los principales aspectos de las herramientas a utilizar para el análisis, diseño e implementación de la extensión.

Capítulo II: Análisis y diseño de la extensión para modelar AD de la en la herramienta VP, versión 2.0.

En este capítulo se describe la propuesta de solución para la situación problemática anteriormente planteada. Se definen los requisitos funcionales y los no funcionales a tener en cuenta. Se exponen los artefactos necesarios que guían el proceso de desarrollo, siguiendo la metodología de software seleccionada.

Capítulo III: Implementación y pruebas de la extensión para modelar AD en la herramienta VP, versión 2.0.

En este capítulo se realiza la descripción de las principales funcionalidades desarrolladas para la extensión, las cuales estarán regidas por un conjunto de estándares de codificación, y se valida la solución propuesta mediante la aplicación de pruebas de funcionalidad y de aceptación.

Capítulo I: Fundamentos teóricos para el desarrollo de la extensión para modelar AD en la herramienta VP, versión 2.0.

En el presente capítulo se profundiza en los aspectos teóricos de los AD; así como las herramientas, metodologías de desarrollo y tecnologías que serán objeto de análisis y estudio, con el fin de fundamentar su uso. También se precisará el lenguaje de programación y el entorno de desarrollo donde se desarrollará la solución que se propone.

1.1 Conceptos asociados al dominio del problema

Con el fin de alcanzar una mejor comprensión de los temas que serán tratados en este capítulo, directamente relacionados con el objeto de estudio de la investigación, se describen a continuación un conjunto de conceptos asociados al dominio del problema.

Almacén de Datos

Un AD es un repositorio de datos de muy fácil acceso, alimentado de numerosas fuentes, transformadas en grupos de información sobre temas específicos de negocios, para permitir nuevas consultas y análisis (Leon, 2012).

La creación de un AD representa en la totalidad de las ocasiones el primer paso, desde el punto de vista técnico, para implantar una solución completa de inteligencia de negocios. En la actualidad se puede afirmar que los avances alcanzados en el desarrollo de los AD, confirman que ya es una tecnología madura y estable, lo que no impide su constante evolución (Villanueva, 2012).

En términos de repositorio de datos los AD reúnen características especiales (Inmon, 2008):

- Orientado a temas: los datos en la base de datos están organizados de manera tal, que todos los elementos de datos relativos al mismo evento u objeto del mundo real queden unidos entre sí.
- Integrado: la base de datos contiene los datos de todos los sistemas operacionales de la organización, y dichos datos deben ser consistentes.
- No volátil: la información no se modifica, ni se elimina una vez almacenado un dato, éste se convierte en información de sólo lectura, y se mantiene para futuras consultas.

- Variante en el tiempo: los cambios producidos en los datos a lo largo del tiempo quedan registrados para que los informes que se puedan generar reflejen esas variaciones.

Basado en los conceptos definidos anteriormente se expone, que un AD es una gran colección de datos que recoge información de múltiples fuentes u operacionales dispersos. En la sociedad actual se han convertido en pilares fundamentales para el proceso de toma de decisiones de las instituciones o empresas donde sean utilizados.

Extensión

Comúnmente se asocia la palabra extensión con *plugin* o componente. Una extensión es un módulo de *hardware* o *software* que añade una característica o un servicio específico a un sistema más grande. (Fernández, 2011).

Basado en el concepto anterior una extensión consiste en agregarle características o servicios específicos a un sistema, para solucionar una necesidad de los que trabajan con la herramienta. Un ejemplo de lo anteriormente expuesto lo constituye la herramienta VP a la cual se le agregaron nuevas funcionalidades para el modelado de soluciones de AD que permite el trabajo con los elementos de un modelo entidad - relación para lograr que este sea trasformado a un modelo dimensional.

1.2 Modelo dimensional

El concepto modelo dimensional ha sido abordado por diferentes autores, atribuyéndole disímiles definiciones, donde la idea fundamental es que el usuario visualice fácilmente la relación que existe entre los distintos componentes del modelo. A continuación se exponen algunos conceptos referentes al modelo dimensional.

Es una forma de acercar los datos al usuario final. En el mismo los datos se presentan de manera organizada, para garantizar que las respuestas sean rápidas, simples y entendibles, ofreciendo una visión clara respecto a la operación del negocio (Kimball, 2002).

El modelado dimensional es una técnica de modelización de información. Un modelo dimensional organiza la información a través de un formato simétrico, cuyo objetivo es facilitar la comprensión de los usuarios, optimizar el rendimiento de las consultas y facilitar las modificaciones en el modelo para permitir una rápida adaptación ante cambios en las necesidades de información (Espinosa, 2010).

El modelado dimensional es una técnica de diseño lógico que presenta los datos de un modo estandarizado que es intuitivo para los usuarios y proporciona un acceso eficiente a la información. La idea principal del modelado dimensional es que prácticamente toda la información de una organización puede ser representada como un hipercubo de datos de n dimensiones, dónde cada celda contiene una medición y cada eje del cubo determina una dimensión de estudio de los datos (Urquizu, 2012).

El modelo dimensional busca presentar la información de una manera estándar, sencilla y sobre todo intuitiva para los usuarios (López, 2012).

Pasos básicos del modelamiento dimensional (Basurto, 2007):

1. Decidir cuáles serán los procesos de negocios a modelar, basándose en el conocimiento de éstos y de los datos disponibles.
2. Decidir la tabla principal de cada proceso de negocio (tabla de hecho).
3. Decidir las características que representarán la tabla principal (tabla de dimensiones).
4. Elegir las mediciones (medidas) del negocio para la tabla principal.

Teniendo en cuenta los pasos del modelamiento dimensional en los que se hace referencia a las tablas de hechos y tablas de dimensiones se pasan a definir para mejor entendimiento de los términos.

Tablas de hechos

La tabla de hechos es donde las mediciones numéricas del negocio son almacenadas. Cada una de las mediciones es tomada como la intersección de todas las dimensiones (Kimball, 2002).

Existen dos tipos de hechos, los básicos y los derivados (Bernabeu, 2009):

- Hechos básicos: son los que se encuentran representados por un campo de una tabla de hechos.
- Hechos derivados: son los que se forman al combinar uno o más hechos con alguna operación matemática o lógica y que también residen en una tabla de hechos. Estos poseen la ventaja de almacenarse previamente calculados, por lo cual pueden ser accedidos a través de consultas SQL[4] sencillas y devolver resultados rápidamente, pero requieren más espacio físico en el AD.

[4] El lenguaje de consulta estructurado o SQL (por sus siglas en inglés *Structured Query Language*) es un lenguaje declarativo de acceso a bases de datos relacionales que permite especificar diversos tipos de operaciones en ellas.

Tablas de dimensiones

Las tablas dimensionales son aquellas donde las descripciones textuales de las dimensiones del negocio son almacenadas. Cada una de las descripciones textuales ayuda a describir un miembro de la dimensión respectiva (Kimball, 2002).

Cada tabla de dimensión podrá contener los siguientes campos (Bernabeu, 2009):

- Llave principal o identificador único.
- Llave foránea.
- Datos de referencia primarios: datos que identifican la dimensión.
- Datos de referencia secundarios: datos que complementan la descripción de la dimensión.

Medida

Son las mediciones numéricas que representa una actividad específica de un negocio, es decir, es una propiedad de una tabla de hecho usada para su análisis que brinda información relevante.

Existen tres clases de medidas:

- Medidas aditivas: pueden ser combinadas a lo largo de cualquier dimensión
- Medidas semi-aditivas: pueden ser combinadas a lo largo de una o más dimensiones.
- Medidas no aditivas: no pueden ser combinadas a lo largo de ninguna dimensión (Bernabeu, 2009).

Estas tablas según las técnicas de modelado utilizadas pueden adoptar forma de estrella, copo de nieve o constelación de hechos.

Esquema en estrella

Este esquema consta de una tabla de hechos central que se relaciona con varias tablas de dimensiones. Debe estar totalmente desnormalizado, siendo innecesarias las uniones entre las tablas cuando se realiza una consulta, esto favorece a una mayor rapidez de las consultas, aunque tiene la desventaja de generar cierto grado de redundancia.

Entre sus ventajas más significativas está que es el esquema más simple de interpretar y optimiza los tiempos de respuestas ante las consultas de los usuarios. Sin embargo, es el menos robusto para la

carga. Un esquema de estrellas puede tener cualquier número de tablas de dimensiones (IBM, 2013). En la figura 1 se puede apreciar un esquema estrella.

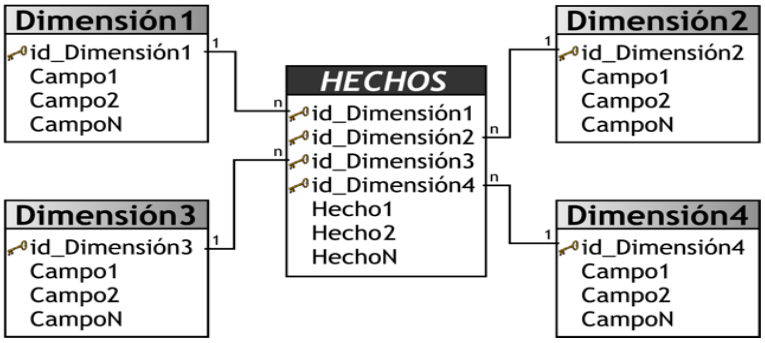


Fig. 1 Esquema estrella (Bernabeu, 2009).

Esquema copo de nieve

Constituye una ampliación del esquema en estrella cuando las tablas de dimensiones se organizan en jerarquías de dimensiones. Existe una tabla de hechos central que está relacionada con una o más tablas de dimensiones, las cuales a su vez pueden relacionarse con nuevas dimensiones. Posibilita la segregación de los datos de las tablas de dimensiones, es muy flexible y puede ser implementado luego de haber desarrollado un esquema en estrella, siendo muy útil en las tablas de dimensiones de muchas tuplas.

Tiene como desventaja que de existir muchas tablas de dimensiones, cada una de ellas con varias jerarquías, pueden crearse abundantes tablas llegando a ser inmanejables. Además su desempeño puede verse reducido si existen muchas uniones y relaciones entre tablas (IBM, 2013). En la figura 2 se puede apreciar un esquema copo de nieve.

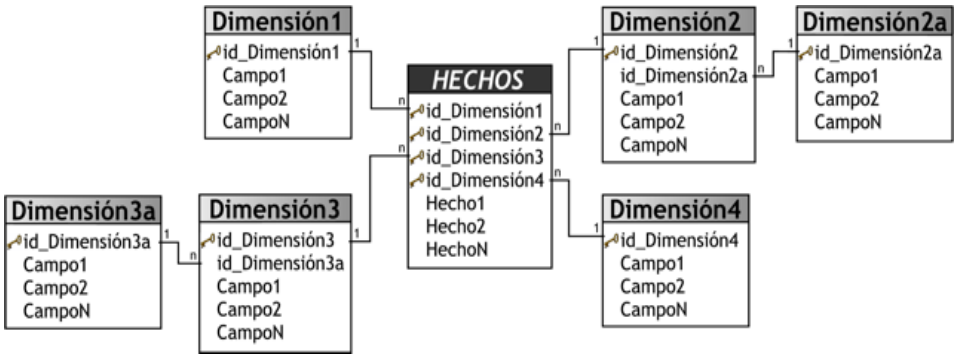


Fig. 2 Esquema copo de nieve (Bernabeu, 2009).

Esquema constelación de hechos

Este modelo está compuesto por una serie de esquemas en estrella que lo conforman una tabla de hechos principal y más tablas de hechos auxiliares. Estas tablas yacen en el centro del modelo y están relacionadas con sus respectivas tablas de dimensiones, vinculándose las tablas de hechos auxiliares con algunas dimensiones asignadas a la tabla de hecho principal y también con nuevas tablas de dimensiones (IBM, 2013). En la figura 3 se puede apreciar un esquema contalación de hechos.

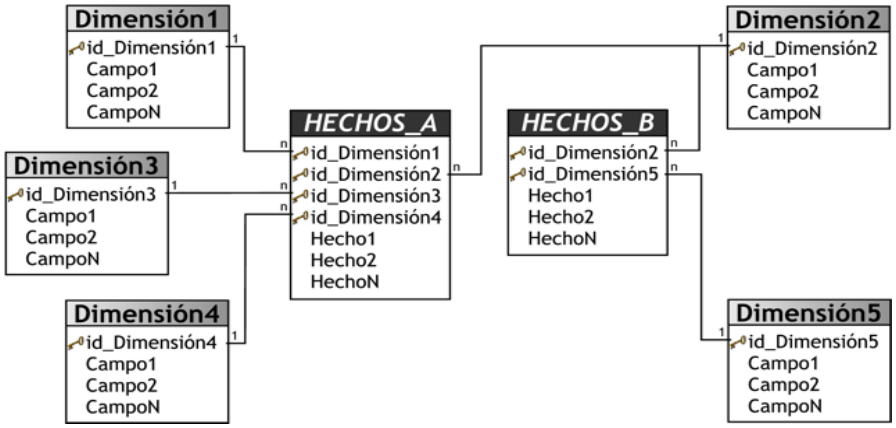


Fig. 3 Esquema constelación de hechos (Bernabeu, 2009).

Después de haber analizado los principales conceptos referentes al modelo dimensional, en cuanto a su composición y forma de organización, se define que para la investigación se haga uso del esquema constelación de hechos. Dicho esquema sirve de base para la virtualización que es objeto de análisis de la investigación.

1.3 Cubos multidimensionales

Un cubo multidimensional es un conjunto formado por las tablas de dimensiones y las tablas de hechos. Como resultado muestra una vista en forma de cubo cuyas celdas están compuestas por las medidas de las tablas hechos. Permiten que los reportes sean obtenidos con un bajo tiempo de respuesta y que el análisis de los datos sea diverso. Cada cara del cubo se refiere a un análisis distinto de las medidas almacenadas (Gallardo, 2012).

Los cubos son elementos claves en OLAP, tecnología que provee rápido acceso a datos en un AD. Los mismos proveen un mecanismo para buscar datos con rapidez y tiempo de respuesta uniforme independientemente de la cantidad de datos en el cubo o la complejidad del procedimiento de búsqueda.

Son el componente más poderoso de los AD, ya que es el motor de consultas especializado del AD. En estos modelos, los datos son vistos como cubos los cuales consisten en categorías descriptivas (tablas de dimensiones) y valores cuantitativos (medidas) (Kimball, 2002).

Las principales características de OLAP son (Sinexus, 2012):

- **Rápido:** proporciona la información al usuario a una velocidad constante (mayormente en cinco segundos o menos).
- **Análisis:** realiza análisis estadísticos y numéricos de los datos de forma básica.
- **Compartid**a: permite compartir los datos potencialmente confidenciales a través de una gran cantidad de usuarios, implementando para esto los requerimientos de seguridad necesarios.
- **Multidimensional:** permite ver la información en determinadas vistas o dimensiones.
- **Información:** accede a todos los datos necesarios, donde quiera que estos residan, y mientras no esté limitada por el volumen.

Entre los objetos más importantes a incluir en el cubo están los siguientes (Gallardo, 2012):

- **Indicadores:** son sumas efectuadas sobre hechos o expresiones que posibilitan analizar los datos almacenados, pertenecen a las tablas de hechos también conocidos como miembros calculables.
- **Atributos:** constituyen los criterios de análisis que se utilizarán para analizar los indicadores, pertenecen a las tablas de dimensiones. Dentro de un cubo multidimensional, los atributos son los ejes del mismo.
- **Jerarquías:** representa una relación lógica entre dos o más atributos.

En muchas de las ocasiones el usuario analiza información que está contenida en más de un cubo, siendo esta de vital importancia para una decisión de la empresa o institución, por lo que los cubos virtuales resultan imprescindibles para realizar dicha función. Para el mejor entendimiento del término cubo virtual se tiene en cuenta primeramente el término virtual que se define a continuación.

Virtual

Es la existencia aparente y no real (Medina, 2014). En computación se utiliza para designar a todo aquello que tiene existencia dentro de una simulación informática (Fallas, 2009).

Se define que un **cubo virtual** es la unión de dos tablas de hechos (cubos) que se deseen analizar desde perspectivas diferentes teniendo en cuenta que el mismo se crea virtualmente pero sus datos no subsisten en la base de datos. Los cubos virtuales contienen medidas y dimensiones que al igual que el cubo al que pertenecen, son virtuales.

Medida virtual

Las medidas virtuales son datos numéricos que representan una actividad específica del hecho virtual, conocida también como una propiedad del hecho casi siempre numérica utilizada para el análisis del mismo.

Dimensión virtual

Se conoce como dimensión virtual a las descripciones del hecho virtual que permiten que pueda ser analizado desde distintos puntos de vista, su funcionamiento es similar a las dimensiones reales. Estas son transparentes para el usuario ya que son evaluadas cuando se realizan las consultas, pues no necesitan de almacenamiento físico.

1.4 Schema Workbench

Es la herramienta gráfica de diseño que permite la construcción visual de los esquemas de cubos OLAP, además permite publicarlos en el servidor de inteligencia del negocio para que puedan ser utilizados en los análisis por los usuarios para decisiones futuras.

Este programa publicado en el año 2007, entrega todas las facilidades para poder realizar el modelo lógico del cubo OLAP al cual se le realizarán las consultas. Este se conecta directamente con la base de dato, para así poder diseñar los cubos OLAP que se requieren para que el usuario final pueda visualizar los indicadores (Espinosa, 2010).

Schema Workbench ofrece las siguientes funcionalidades (Díaz, 2012):

- Editor de esquemas integrados con un origen de datos subyacente para su validación.

- Prueba de consultas MDX [5] contra el esquema y la BD.
- El motor de Mondrian procesa peticiones de MDX que permiten configurar el cubo en dependencia de los intereses. Genera un archivo XML[6] que registra todas las acciones realizadas con la herramienta.

El motor Mondrian es una de las aplicaciones más importantes de la plataforma Pentaho, es utilizado por el Schema Worbench. Pues Mondrian es un servidor OLAP "Open Source" que gestiona la comunicación entre una aplicación OLAP y la base de datos con los datos fuente.

1.5 Metodología de desarrollo del software

Dentro del proceso de desarrollo de software, la metodología de desarrollo de software se define como un conjunto de procedimientos, técnicas, herramientas y un soporte documental que ayuda a los desarrolladores a realizar un nuevo software (Erazo, 2013). Actualmente existen fundamentalmente dos tipos de metodologías para contribuir al desarrollo de software: las metodologías tradicionales, también conocidas como metodologías robustas y las metodologías ágiles o ligeras.

Metodologías tradicionales

Se centran en la definición detallada de los procesos, tareas a realizar y herramientas a utilizar; además requiere una extensa documentación, ya que pretende prever todo de antemano. Este tipo de metodologías son más eficaces y necesarias cuanto mayor es el proyecto que se pretende realizar respecto (respecto a tiempo y recursos), donde el cliente interactúa con el equipo de desarrollo mediante reuniones, debiendo existir un contrato prefijado entre éstos que especifique el alcance del proyecto. Estas no resultan ser adecuadas para muchos de los proyectos actuales donde el entorno del sistema es muy cambiante y se exige reducir drásticamente los tiempos de desarrollo manteniendo una alta calidad (Ruiz, y otros, 2011).

Ante las dificultades de restricciones de tiempo y flexibilidad que poseen estas metodologías que no se adecuan a la presente investigación, se procede al estudio de las metodologías ágiles.

[5] MDX (acrónimo MDX por sus siglas en inglés ***M**ulti**D**imensional e**X**pressions*) es un lenguaje de consulta para bases de datos multidimensionales sobre cubos OLAP, se utiliza en Business Intelligence para generar reportes para la toma de decisiones basados en datos históricos, con la posibilidad de cambiar la estructura o rotación del cubo

[6] XML: lenguaje de marcado extensible (*e**X**tensible **M**arkup **L**anguage,* por sus siglas en inglés), que permite definir la gramática de lenguajes específicos, siendo útil cuando varias aplicaciones se deben comunicar entre sí.

Metodologías ágiles

Las metodologías ágiles dan mayor valor al individuo, el cliente colabora en todo momento; se basa en desarrollar un software iterativo e incremental, sencillo y adaptable (Ruiz, y otros, 2011). Se fundamentan en la entrega temprana del software con el uso de métodos no formales, ya que enfocan su mayor esfuerzo en la elaboración y entrega del producto. Se nombran ágiles por la capacidad de responder rápida y efectivamente ante los cambios. Se apoyan en las habilidades y experiencias personales del equipo, evitando los extenuantes caminos de las metodologías tradicionales, destacándose Programación Extrema (XP) y OpenUp.

Programación Extrema (XP)

Se utiliza en proyectos con pequeños equipos de desarrollo y con corto plazo de entrega. Se basa en la retroalimentación entre el cliente y el equipo de desarrollo, buena comunicación entre los participantes y simplicidad en las soluciones implementadas. Consiste en una programación rápida, cuya particularidad es tener como miembro del equipo al usuario final. Su diseño hace que apenas se le dé importancia al análisis como fase independiente, debido a que se trabaja exclusivamente en función de las necesidades del momento. Es muy costosa en caso de fallar (Beck, 1999). Esta metodología fue la utilizada en la versión 1.0 de la extensión.

OpenUp

La metodología Proceso Unificado Abierto (acrónimo OpenUp por sus siglas en inglés *Open Unified Process*) preserva las características fundamentales del Proceso Unificado de Desarrollo (acrónimo RUP por sus siglas en inglés *Rational Unified Process*). La misma es un proceso de desarrollo de software de código abierto diseñado para pequeños equipos organizados, quienes quieren tomar una aproximación ágil del desarrollo. Es un proceso iterativo mínimo, completo y extensible (Gomez, 2014). OpenUp está organizado dentro de cuatro áreas principales de contenido: comunicación y colaboración, intención, solución y administración. Se caracteriza por ser iterativo e incremental, estar centrado en la arquitectura y guiado por los casos de uso (Delgado, 2010).

Open UP es un proceso iterativo con iteraciones distribuidas en las siguientes cuatro fases. En la figura 4 se puede observar las fases de vida de OpenUp:

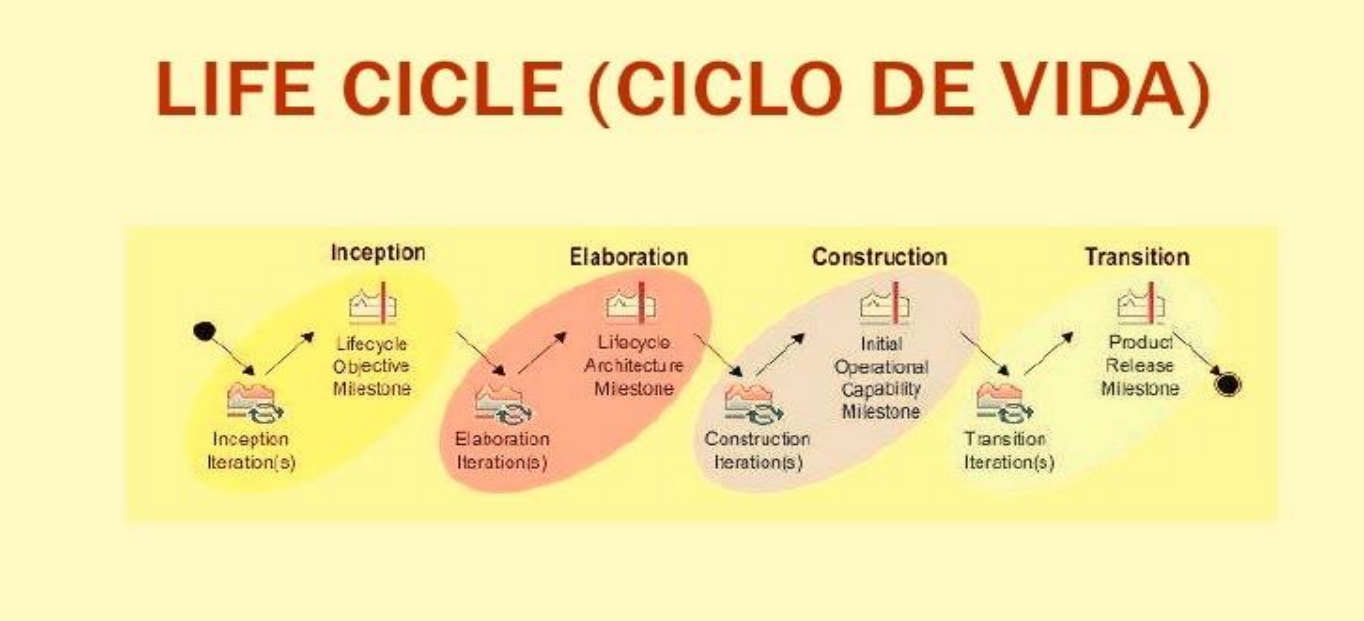


Fig. 4 Fases de OpenUp (Balduino, 2007).

A continuación se describen cada una de las fases de OpenUp (Balduino, 2007):

Fase de inicio: en esta fase, las necesidades de cada participante del proyecto son tomadas en cuenta y plasmadas en objetivos del proyecto. Se definen para el proyecto: el ámbito, los límites, el criterio de aceptación, los casos de uso críticos, una estimación inicial del coste y un boceto de la planificación.

Fase de elaboración: en esta fase se realizan tareas de análisis del dominio, negocio y definición de la arquitectura del sistema. Se debe elaborar un plan de proyecto, estableciendo los requisitos y una arquitectura estable. Por otro lado, el proceso de desarrollo, las herramientas, la infraestructura a utilizar y el entorno de desarrollo también se especifican en detalle en esta fase.

Fase de construcción: todos los componentes y funcionalidades del sistema que falten por implementar son realizados, probados e integrados en esta fase. Los resultados obtenidos en forma de incrementos ejecutables deben ser desarrollados de la forma más rápida posible sin dejar de lado la calidad de lo desarrollado.

Fase de transición: esta fase corresponde a la introducción del producto en la comunidad de usuarios, cuando el producto está lo suficientemente maduro. La fase de la transición consta de las sub-fases de pruebas de versiones beta, pilotaje y capacitación de los usuarios finales y de los encargados del

mantenimiento del sistema. En función de la respuesta obtenida por los usuarios puede ser necesario realizar cambios en las entregas finales o implementar alguna funcionalidad más.

Principios de OpenUp

OpenUp define como principios básicos los que a continuación se describen (Balduino, 2007):

- Colaborar para sincronizar intereses y compartir conocimiento. Este principio promueve prácticas que impulsan un ambiente de equipo saludable, facilitan la colaboración y desarrollan un conocimiento compartido del proyecto.
- Equilibrar las prioridades para maximizar el beneficio obtenido por los interesados en el proyecto. Este principio promueve prácticas que permiten a los participantes de los proyectos desarrollar una solución que maximice los beneficios obtenidos por los participantes y que cumpla con los requisitos y restricciones del proyecto.
- Centrarse en la arquitectura de forma temprana para minimizar el riesgo y organizar el desarrollo.
- Desarrollo evolutivo para obtener retroalimentación y mejoramiento continuo. Este principio promueve prácticas que permiten a los equipos de desarrollo obtener retroalimentación temprana y continua de los participantes del proyecto, permitiendo demostrarles incrementos progresivos en la funcionalidad.

En la presente investigación se decide optar por la metodología OpenUp para el desarrollo de la versión 2.0 de la extensión, debido a que XP es adecuada para proyectos con requisitos imprecisos, muy cambiantes, y donde existe un alto riesgo técnico. Además OpenUp provee un conjunto simplificado de artefactos, roles, tareas y guías de trabajo. Esta evita la elaboración de documentación, diagramas e iteraciones innecesarios, y permite disminuir las probabilidades de fracaso e incrementar las probabilidades de éxito. Es una metodología de desarrollo de software de código abierto diseñada para pequeños equipos. Además es preciso centrarse en la arquitectura para lograr una eficiente integración de la solución que se propone y que esta sea escalable y flexible; su desarrollo es iterativo e incremental lo que es reducir el tiempo de desarrollo, dividiendo el proyecto en intervalos incrementales superpuestos; todos los requisitos se analizan antes de empezar a desarrollar.

1.6 Herramientas a utilizar

Para el diseño e implementación de la extensión de VP en su versión 2.0 se hace necesario tener en cuenta un conjunto de herramientas que sean efectivas para el desarrollo de la solución y que en la medida de lo posible, respondan a las políticas de la migración a software libre de la universidad.

Herramienta CASE

Son un conjunto de programas y ayudas que dan asistencia a los analistas, ingenieros de *software* y desarrolladores, durante todos los pasos del ciclo de vida de desarrollo de un *software*. Permite Modelar los Procesos de Negocios (acrónimo BPM por sus siglas en inglés *Business Process Management*) de las empresas y desarrollar los Sistemas de Información Gerenciales (Reyna, 2013). Existen múltiples herramientas con estos fines, tales como *Enterprise Architect*, *VP, ArgoUML, Rational Rose*, entre otras.

Visual Paradigm 8.0 *Community Edition*

Es una herramienta CASE profesional que da soporte al análisis y diseño orientado a objetos, construcción, pruebas y despliegue del ciclo de vida completo del desarrollo de software, tiene licencia gratuita. Soporta las últimas versiones de UML así como la Notación y Modelado de Procesos de Negocios (acrónimo BPMN por sus siglas en inglés *Business Process Modeling Notation*). Brinda la posibilidad de crear todos los tipos de diagramas de clases, código inverso, generación de código a partir de diagramas y generar documentación en varios formatos como *MS, Word* y *PDF*. Permite también la comparación entre diagramas resaltando las diferencias, diseño de patrones y trazabilidad en el desarrollo de los modelos, soporta el proceso de ingeniería de varios gestores de base de datos (Paradigm, 2010).

Entre sus principales ventajas están:

- Multiplataforma: soportada en plataforma Java para Sistemas Operativos Windows, Linux que será donde se podrá ejecutar la extensión.
- Interoperabilidad: intercambia diagramas UML y modelos con otras herramientas. Soporta la importación y exportación a formatos XML necesario en la investigación.
- 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.

Su versión 8.0 incluye la funcionalidad de crear y especificar perfiles UML, la cual resulta de vital importancia para la implementación y ejecución de extensiones para la herramienta. Debido a todas las características mencionadas y los beneficios que brinda para el desarrollo de software, especialmente referentes al modelado, se decidió utilizar VP 8.0 *Community* Edition para el modelado de la extensión, además de ser la herramienta de modelado escogida por la Universidad para el desarrollo de software, asumida por el centro DATEC en sus producciones.

Extensión para modelar soluciones de AD en la herramienta de modelado VP

Es una extensión desarrollada como parte de una investigación precedente, que permite el modelado de soluciones de AD en la herramienta VP, la misma logra:

- Modelar lógicamente la estructura de un AD (hechos, medidas y dimensiones).
- Representar las jerarquías en el modelado del AD.
- Generar un XML con el diseño preliminar de los cubos OLAP a partir del diseño del AD.
- Generar el *script* para cargar el diseño en el gestor de base de datos.

Entorno de Desarrollo Integrado (IDE)

Es un software, cuyo principal objetivo es el desarrollo de otros *softwares* (Flores, 2005). Es un entorno de programación que ha sido empaquetado como un programa de aplicación. Puede ser exclusivo para un lenguaje de programación o bien, para varios. Suele consistir en un editor de código, con facilidades como completamiento de código y navegación entre clases, un compilador y herramientas de automatización de la compilación, un depurador y en algunos casos un constructor de interfaz gráfica (Carreño, 2013).Una vez seleccionado el lenguaje para implementar y el tipo de aplicación a desarrollar, se valoró el uso de varios IDE como fueron: Aptana Studio, Eclipse, Spket y Netbeans. Para el desarrollo del sistema se asume el NetBeans, pues sus características ofrece comodidades como herramienta para el desarrollo de la extensión para el VP además de ser el IDE utilizado en la versión 1.0; por lo que se profundiza en el análisis de sus características.

NetBeans 8.0

NetBeans es un entorno integrado de desarrollo disponible para Windows, Mac, Linux y Solaris. Consiste en un IDE de código abierto y una plataforma de aplicaciones que permiten a los desarrolladores crear rápidamente aplicaciones web, escritorio y aplicaciones móviles utilizando la plataforma Java, así como

JavaFX, PHP, JavaScript y Ajax, Ruby y Ruby onRails, Groovy y Grails, y C/C++ (Salter, 2014). Se escoge como IDE de desarrollo por ser el que propone la API[7] de VP para el desarrollo de extensiones.

1.7 Lenguaje de programación

Es un conjunto de reglas, notaciones, símbolos y/o caracteres que permiten a un programador expresar el procesamiento de datos y sus estructuras en la computadora. Cada lenguaje posee sus propias sintaxis. También se puede decir que un programa es un conjunto de órdenes o instrucciones que resuelven un problema específico basado en un lenguaje de programación (Vásquez, 2014). El lenguaje de programación permite a los programadores especificar sobre qué datos la computadora debe operar, cómo deben ser almacenados, transmitidos y cuáles acciones ejecutar ante determinadas circunstancias.

Java

Es un lenguaje de programación de plataforma independiente desarrollado por Sun Microsystems en los años 90. Emplea el paradigma de programación orientado a objeto para propósito general. El lenguaje toma sintaxis de lenguajes como C y C++, aunque tiene un modelo de objeto más simple y elimina herramientas de bajo nivel (Vásquez, 2014). Este lenguaje proporciona numerosas comprobaciones en compilación, tiempo de ejecución e interpretación, lo que lo ha llevado a ubicarse entre los lenguajes de preferencias entre la comunidad de desarrolladores a nivel internacional.

Entre sus principales ventajas están (Fernández, 2011):

- Es un lenguaje multiplataforma: este lenguaje de programación funciona en cualquier sistema operativo en el cual esté instalada la máquina virtual de Java.
- Posee capacidad, gran rendimiento y permite la creación de aplicaciones distribuidas.
- Orientado a objetos: al ser un lenguaje orientado a objeto, permite el encapsulamiento de los métodos o funciones para su manipulación.
- Dinámico: su sistema de ejecución en tiempo es dinámico en la fase de enlazado, lo que permite que las clases sólo se enlacen en la medida que sean necesarias.

[7] API: interfaz de programación de aplicaciones (del inglés: *Application Programming Interface*), es el conjunto de subrutinas, funciones y procedimientos (o métodos, en la programación orientada a objetos) que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción. Son usadas generalmente en las bibliotecas.

- Multihilo: permite la ejecución de varias tareas a la vez.

Para la implementación de la presente investigación se utilizará como lenguaje de programación Java. Además de los otros elementos mencionados en el epígrafe anterior, es el lenguaje propuesto por la API de desarrollo de la herramienta VP y es el utilizado en la versión precedente.

1.8 Conclusiones del capítulo

Durante el desarrollo de este capítulo se analizan los principales conceptos teóricos que dan soporte a la presente investigación permitiendo una mejor compresión de la misma. Como metodología de desarrollo se adopta OpenUp por ser una metodología ágil centrada en la arquitectura, con un desarrollo iterativo incremental; además de ser considerada la más adecuada para la situación que se plantea. La herramienta CASE utilizada es el VP, pues es una herramienta UML profesional que soporta el ciclo de vida completo del desarrollo de software y es la utilizada en el centro DATEC. Se optó como NetBeans IDE 8.0; acompañado del lenguaje de programación Java, un lenguaje multiplataforma que puede ser utilizado en diferentes sistemas operativos. El uso de esta metodología, tecnologías y herramientas permitirá el desarrollo de la extensión para modelar soluciones de AD en la herramienta VP en su versión 2.0.

Capítulo II: Análisis y diseño de la extensión para modelar AD en la herramienta VP, versión 2.0.

En el presente capítulo se realiza una descripción detallada de la versión 2.0 de la extensión para la herramienta de modelado VP. En el mismo se especifican los requisitos funcionales y no funcionales a tener en cuenta siguiendo la metodología de desarrollo de software seleccionada (OpenUp).

2.1 Propuesta de solución

Se propone la realización de la versión 2.0 de la extensión para modelar soluciones de AD en la herramienta VP para que en dicha extensión se realice un completo diseño dimensional. Esta versión permitirá modelar los elementos virtuales del modelo dimensional de un AD y estará orientada a facilitar el diseño del mismo. Podrán crearse formularios diferentes a la versión anterior de la extensión, facilitando el análisis de hechos desde perspectivas diferentes. También generará un archivo XML permitiendo guardar los elementos virtuales conjuntamente con las funciones definidas por el usuario y los roles que sean creados en la extensión. El archivo XML será utilizado por Pentaho BI Server para la generación de reportes y las vistas de análisis.

2.2 Modelo del dominio

El modelo de dominio es una representación visual de las clases conceptuales u objetos del mundo real en un dominio de interés (Larman, 2004). Permite que el usuario visualice los principales conceptos que se manejan para poder comprender el contexto en el cual se enmarca el sistema, donde se expone un marco conceptual y las relaciones entre estas definiciones. El modelo desarrollado no se trata de un conjunto de diagramas que describen clases de software u objetos de software con responsabilidades, sino que puede considerarse como un diccionario visual de las abstracciones relevantes, vocabulario e información del dominio (García, y otros, 2011). A continuación el diagrama del modelo del dominio:

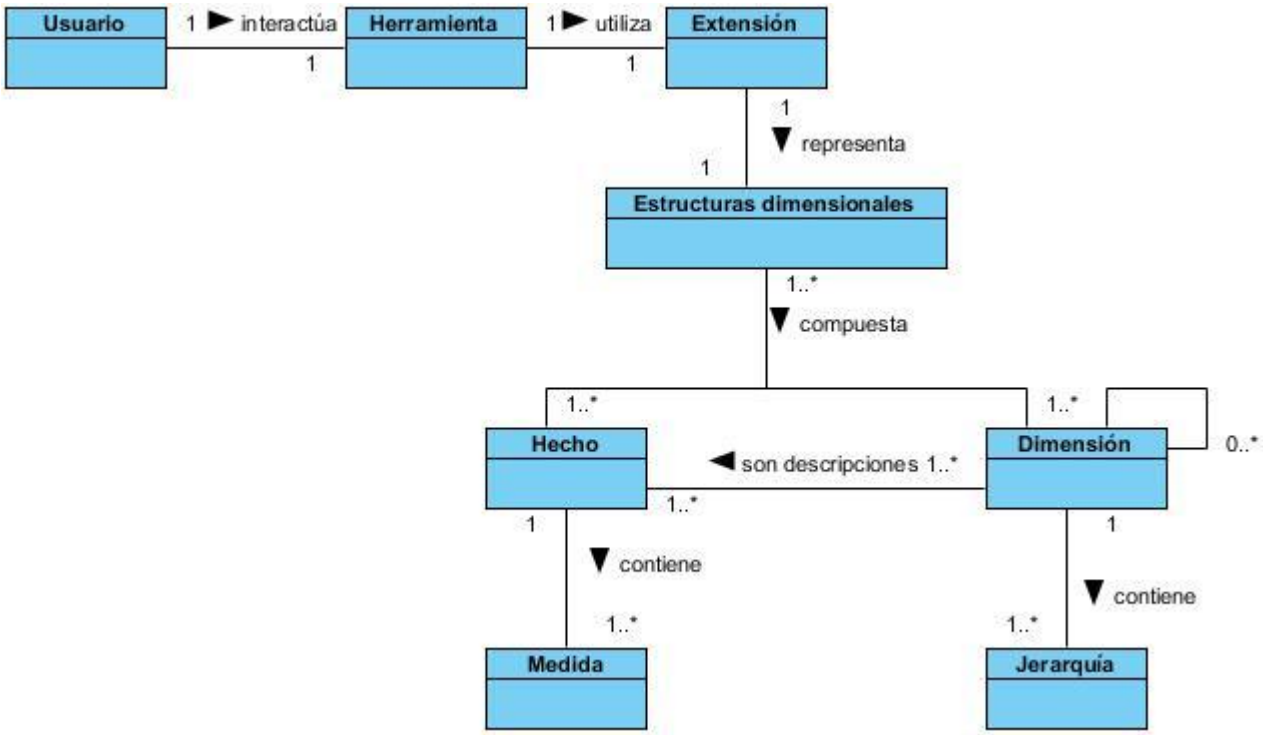


Fig. 5 Diagrama conceptual del dominio de la extensión.

Descripción de los conceptos del diagrama de dominio de la extensión

Usuario: persona facultada para utilizar la extensión asistida a la herramienta VP.

Herramienta: herramienta CASE VP.

Extensión: componente integrado a la herramienta VP.

Estructuras dimensionales: buscan presentar los datos en un estándar y facilitar una recuperación adecuada de estos. Se basa en la dualidad hecho-dimensión.

Hecho: se denomina hecho o cubo a la actividad que se realiza en el negocio en un tiempo determinado. También se puede definir como el objeto de análisis en el negocio para la toma de decisiones que están caracterizados por medidas numéricas. Constituye una tabla de hecho en el modelo físico.

Medida: datos numéricos que representan una actividad específica de un negocio.

Dimensión: se conoce como dimensión a las descripciones de un hecho que permite su análisis posterior en el proceso de toma de decisiones y brinda una perspectiva adicional a un hecho dado. Son

agrupaciones lógicas de atributos con un significado común y atómico. Son usadas para seleccionar y agregar datos a un cierto nivel deseado de detalle. Constituye una tabla de dimensión en el modelo físico.

Jerarquía: en las jerarquías 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 vendidos en el año Y, en el mes Z y en el día W.

2.3 Requisitos

Un requisito es simplemente una declaración abstracta de alto nivel de un servicio que debe proporcionar el sistema o una restricción de éste. También descrito como una definición detallada y formal de una función del sistema (Sommerville, 2006).

Uno de los pasos más importante en el proceso de desarrollo de un software es el levantamiento de requisitos del sistema. En la presente investigación se definieron un conjunto de requisitos funcionales y no funcionales los cuales son la base para obtener una solución que cumpla con las solicitudes del cliente y la calidad requerida en esta.

Requisitos Funcionales

Los Requisitos Funcionales (RF), son declaraciones de los servicios que debe proporcionar el sistema, de la manera en que éste debe reaccionar a entradas particulares y de cómo se debe comportar en situaciones particulares (Sommerville, 2006).

A continuación se muestran los RF identificados:

Tabla. 1 Requisitos funcionales.

#	Requisitos Funcionales	Descripción	Entrada	Salida
RF 1.	Adicionar cubo virtual.	La extensión debe mostrar una interfaz en la que le permita al especialista crear un cubo virtual y así poder analizar información desde distintos puntos de vistas.	Nombre, descripción, encabezado, visible, habilitado.	Cubo virtual.

RF 2.	Modificar cubo virtual.	La extensión debe ser capaz de modificar un cubo virtual.	Cubo virtual.	Cubo virtual modificado.
RF 3.	Eliminar cubo virtual.	La extensión debe ser capaz de eliminar un cubo virtual.	Cubo virtual.	
RF 4.	Visualizar cubos virtuales	La extensión debe ser capaz de mostrar todos los cubos virtuales creados.	Cubo virtual.	Lista de cubos virtuales creados.
RF 5.	Adicionar dimensión virtual.	La extensión debe mostrar una interfaz en la que permita al usuario crear una dimensión virtual. Para el mismo se debe tener en cuenta que, el usuario escogerá de los cubos existentes que no son virtuales al que pertenecerá dicha dimensión, además de la dimensión que será objeto en el análisis en el cubo virtual.	Nombre, encabezado, nombre del cubo, dimensión.	Dimensión virtual.
RF 6.	Modificar dimensión virtual.	La extensión debe ser capaz de modificar dimensión virtual.	Dimensión virtual.	Dimensión virtual actualizada.
RF 7.	Eliminar dimensión virtual.	La extensión debe ser capaz de eliminar una dimensión virtual.	Dimensión virtual.	
RF 8.	Visualizar dimensión virtual.	La extensión debe ser capaz de mostrar la dimensión virtual creada en una lista.	Dimensión virtual.	Lista de dimensiones virtuales.
RF 9.	Adicionar medida virtual.	La extensión debe mostrar una interfaz en la que le permita al usuario crear una medida virtual. Para el mismo se debe tener en cuenta que el usuario tendrá que escoger de los cubos existentes que no son virtuales al que pertenecerá dicha medida, además la medida que será objeto en	Nombre, nombre de la medida, nombre del cubo.	Medida virtual.

		análisis en el cubo virtual.		
RF 10.	Modificar medida virtual.	La extensión debe ser capaz de modificar una medida virtual.	Medida virtual.	Medida virtual modificada.
RF 11.	Eliminar medida virtual.	La extensión debe ser capaz de eliminar una medida virtual.	Medida virtual.	
RF 12.	Visualizar medida virtual.	La extensión debe ser capaz de mostrar la medida virtual creada en una lista.	Medida virtual.	Lista de medidas virtuales.
RF 13.	Adicionar rol.	La extensión debe mostrar una interfaz en la que le permita al usuario crear un rol al esquema. Dicho rol será creado con el fin de establecer la seguridad por los miembros del esquema.	Nombre del rol.	Rol.
RF 14.	Modificar rol.	La extensión debe ser capaz de modificar los roles del esquema.	Rol.	Rol actualizado.
RF 15.	Eliminar rol.	La extensión debe ser capaz de eliminar los roles del esquema.	Rol.	
RF 16.	Visualizar rol.	La extensión debe ser capaz de mostrar los roles del esquema creados.	Rol.	Lista de roles del esquema.
RF 17.	Adicionar permiso al esquema.	La extensión debe mostrar una interfaz en la que le permita al usuario crear permisos sobre el esquema. Para el mismo se debe tener en cuenta el haberse creado al menos un rol con anterioridad.	Acceso.	Permiso al esquema.
RF 18.	Modificar permiso del esquema.	La extensión debe ser capaz de modificar los permisos del esquema.	Permiso al esquema.	Permiso actualizado.
RF 19.	Eliminar permiso del	La extensión debe ser capaz de eliminar	Permiso de	

	esquema.	los permisos al esquema.	esquema.	
RF 20.	Visualizar permiso del esquema.	La extensión debe ser capaz de mostrar los permisos del esquema creados.	Permiso al esquema.	Lista de permisos del esquema.
RF 21.	Adicionar permisos al cubo.	La extensión debe mostrar una interfaz en la que le permita al usuario crear permisos sobre un determinado cubo. Para el mismo se debe tener en cuenta el haberse creado al menos un permiso al esquema con anterioridad.	Acceso, cubo.	Permiso del cubo.
RF 22.	Modificar permisos del cubo.	La extensión debe ser capaz de modificar los permisos del cubo.	Permiso al cubo.	Permiso actualizado.
RF 23.	Eliminar permiso del cubo.	La extensión debe ser capaz de eliminar los permisos del cubo.	Permiso del cubo.	
RF 24.	Visualizar permiso del cubo.	La extensión debe ser capaz de mostrar los permisos del cubo creados.	Permiso al cubo.	Lista de permisos del cubo.
RF 25.	Adicionar permiso a la dimensión.	Descripción: la extensión debe mostrar una interfaz en la que le permita al usuario crear permisos sobre una determinada dimensión. Para el mismo se debe tener en cuenta el haberse creado al menos un permiso al cubo con anterioridad.	Acceso, dimensión.	Permiso de dimensión.
RF 26.	Modificar permiso de la dimensión.	Descripción: la extensión debe ser capaz de modificar los permisos de la dimensión.	Permiso de la dimensión.	Permiso actualizado.
RF 27.	Eliminar permiso de la dimensión.	La extensión debe ser capaz de eliminar los permisos de la dimensión.	Permiso de la dimensión.	
RF 28.	Visualizar permiso de la dimensión.	La extensión debe ser capaz de mostrar los permisos de la dimensión creados.	Permiso de la dimensión.	Lista de permisos de la

				dimensión.
RF 29.	Adicionar permiso a la jerarquía.	La extensión debe mostrar una interfaz en la que le permita al usuario crear permisos sobre una determinada jerarquía. Para el mismo se debe tener en cuenta el haberse creado al menos un permiso a la dimensión con anterioridad.	Acceso, jerarquía, nivel máximo, nivel mínimo.	Permiso de jerarquía
RF 30.	Modificar permiso de la jerarquía.	La extensión debe ser capaz de modificar los permisos de la jerarquía.	Permiso de la jerarquía.	Permiso actualizado.
RF 31.	Eliminar permiso de la jerarquía.	La extensión debe ser capaz de eliminar los permisos de la jerarquía.	Permiso de la jerarquía.	
RF 32.	Visualizar permisos de la jerarquía.	La extensión debe ser capaz de mostrar los permisos de la jerarquía creados.	Permiso de la jerarquía.	Lista de permisos de la jerarquía.
RF 33.	Adicionar permiso del miembro.	La extensión debe mostrar una interfaz en la que le permita al usuario crear permisos sobre el miembro. Para el mismo se debe tener en cuenta el haberse creado al menos un permiso de jerarquía con anterioridad.	Acceso, miembro	Permiso del miembro.
RF 34.	Modificar permiso del miembro.	La extensión debe ser capaz de modificar los permisos al miembro.	Permiso al miembro.	Permiso del miembro actualizado.
RF 35.	Eliminar permiso del miembro.	La extensión debe ser capaz de eliminar los permisos del miembro.	Permiso del miembro.	
RF 36.	Visualizar permisos del miembro.	La extensión debe ser capaz de mostrar los permisos del miembro creados.	Permiso del miembro.	Lista de permisos del miembro.
RF 37.	Adicionar miembro	La extensión debe mostrar una interfaz que	Nombre,	Miembro

	calculable.	le permita al usuario crear un miembro calculable. Para ello se debe tener en cuenta la existencia de un cubo o de un cubo virtual.	descripción, encabezado, dimensión, jerarquía, padre, fórmula|elementos, formato string.	calculable.
RF 38.	Modificar miembros calculables.	La extensión debe ser de modificar un miembro calculable.	Miembro calculable.	Miembro calculable actualizado.
RF 39.	Eliminar miembros calculables.	La extensión debe ser capaz de eliminar un miembro calculable.	Miembro calculable.	
RF 40.	Visualizar miembro calculable.	La extensión debe ser capaz de mostrar el miembro calculable creado en una lista.	Miembro calculable.	Lista de miembros calculables.
RF 41.	Adicionar NS (acrónimo NS por sus siglas en inglés, *Named Set*).	La extensión debe mostrar una interfaz que le permita al usuario crear el NS. Esta funcionalidad debe permitir fijar un conjunto a través de una consulta MDX, para ser reutilizado en el Pentaho BI Server.	Nombre, descripción, fórmula y encabezado.	NS.
RF 42.	Modificar NS.	La extensión debe ser capaz de modificar el NS.	NS.	NS modificado.
RF 43.	Eliminar NS.	La extensión debe ser capaz de eliminar el NS.	NS.	
RF 44.	Visualizar NS.	La extensión debe ser capaz de mostrar NS creado en una lista.	NS.	Listas de NS.
RF 45.	Adicionar UDF (acrónimo UDF por sus siglas en	La extensión debe ser capaz de mostrar una interfaz que le permita al usuario crear	Nombre, nombre de la	UDF.

	inglés *User Defined Function*).	el UDF. Esta funcionalidad debe permitir cargar funciones creadas por el usuario, para ser utilizadas en la manipulación de los datos derivados.	clase.	
RF 46.	Modificar UDF.	La extensión debe ser capaz de modificar el UDF.	UDF.	UDF modificado.
RF 47.	Eliminar UDF.	La extensión debe ser capaz de eliminar el UDF.	UDF.	
RF 48.	Visualizar UDF.	La extensión debe ser capaz de mostrar el UDF creado en una lista.	UDF.	Lista de UDF.

Requisitos no funcionales

Los Requisitos no funcionales (RnF), son propiedades o cualidades que el producto debe tener. Estas propiedades o cualidades se refieren a las características que hacen al producto atractivo, usable, rápido o confiable (Sommerville, 2006). Por lo general los RnF son fundamentales en el éxito del producto; normalmente están vinculados a los RF, es decir, una vez que se conoce lo que el sistema debe hacer se puede determinar cómo ha de comportarse, qué cualidades o propiedades debe tener.

A continuación se describen los RnF necesarios para que sea utilizada la extensión.

Tabla. 2 Requisitos no funcionales y descripción.

Categoría	Descripción
Requisitos de Software	Para el uso de la extensión se deberá disponer del sistema operativo: Windows XP o superior, GNU/Linux preferentemente Ubuntu GNU/Linux 10.4 o superior. Máquina Virtual de Java JDK y JRE tanto para GNU/Linux como para Windows.
Requerimientos de Hardware	Para el uso de la extensión se deberá disponer de 1GB de RAM o superior, 300 MB libres de disco duro como mínimo.
Diseño e implementación	El análisis y diseño del componente será basado en la metodología OpenUp. Utilizando para la implementación el lenguaje de programación Java.

Requisitos de Soporte	La solución estará bien documentada de forma tal que el tiempo de mantenimiento sea mínimo en caso de necesitarlo, permitiendo implementar cambios, ya sea de corrección o mejora del software. Se proveerá un manual de usuario con todas las especificaciones de la extensión.
Requisitos de portabilidad	La extensión podrá ser integrada al VP en diferentes sistemas operativos, por ser el VP multiplataforma, entre los que se encuentra Windows XP o superior, GNU/Linux preferentemente Ubuntu GNU/Linux 10.4 o superior.
Apariencia o interfaz externa	La extensión al integrarse con el VP, tendrá una interfaz amigable, sencilla, legible y fácil de interactuar para hacer más factible el uso para los usuarios, manteniendo los estándares y características de la herramienta VP.

2.4 Modelo de casos de uso del sistema

Un Caso de Uso (CU) es una secuencia de interacciones que se desarrollarán entre un sistema y sus actores en respuesta a un evento que inicia un actor principal sobre el propio sistema. Es además la técnica para la captura de requisitos potenciales de un nuevo sistema o una actualización de software (Penagos, 2014).

Actores del sistema

Un actor del sistema especifica un rol jugado por un usuario o cualquier otro sistema como *hardware* externo u otros sujetos que interactúan con el sujeto (Bars, 2007) en este caso la extensión. Un actor no necesariamente representa una entidad física específica, sino simplemente una faceta particular (es decir, un "rol") de alguna actividad que es relevante a la especificación de sus casos de uso asociados. A continuación se especifican los actores del sistema.

Tabla. 3 Actores del sistema y descripción.

Actor	Descripción
Especialista	Profesional encargado de generar los cubos virtuales con sus respectivas características, también genera los miembros calculables y trabaja con los elementos relacionados a los permisos que puede gestionar

Diagrama de caso de uso del sistema

Los diagramas de casos de uso sirven para especificar la comunicación y el comportamiento de un sistema mediante su interacción con los usuarios y/u otros sistemas (Penagos, 2014). O lo que es igual, un diagrama que muestra la relación entre los actores y los casos de uso del sistema. Los RF del sistema definido anteriormente quedan agrupados en 12 CU presentados en la tabla 3.

Tabla. 4 CU del sistema.

Referencia a requisitos	Nombre del caso de uso
RF 1, RF 2, RF 3, RF 4.	CU 1: Gestionar cubo virtual
RF 5, RF 6, RF 7, RF 8.	CU 2: Gestionar dimensión virtual
RF 9, RF 10, RF 11, RF 12.	CU 3: Gestionar medida virtual
RF 13, RF 14, RF 15, RF 16.	CU 4: Gestionar rol
RF 17, RF 18, RF 19, RF 20.	CU 5: Gestionar permiso al esquema
RF 21, RF 22, RF 23, RF 24.	CU 6: Gestionar permiso al cubo
RF 25, RF 26, RF 27, RF 28.	CU 7: Gestionar permiso a la dimensión
RF 29,RF 30, RF 31,RF 32	CU 8: Gestionar permiso a la jerarquía
RF 33, RF 34, RF 35, RF 36	CU 9: Gestionar permiso al miembro
RF 37, RF 38, RF 39,RF 40	CU 10: Gestionar miembro calculable
RF 41, RF 42, RF 43,RF 44	CU 11: Gestionar NS
RF 45, RF 46, RF 47, RF 48	CU 12: Gestionar UDF

Ver figura 6 el diagrama de CU del sistema:

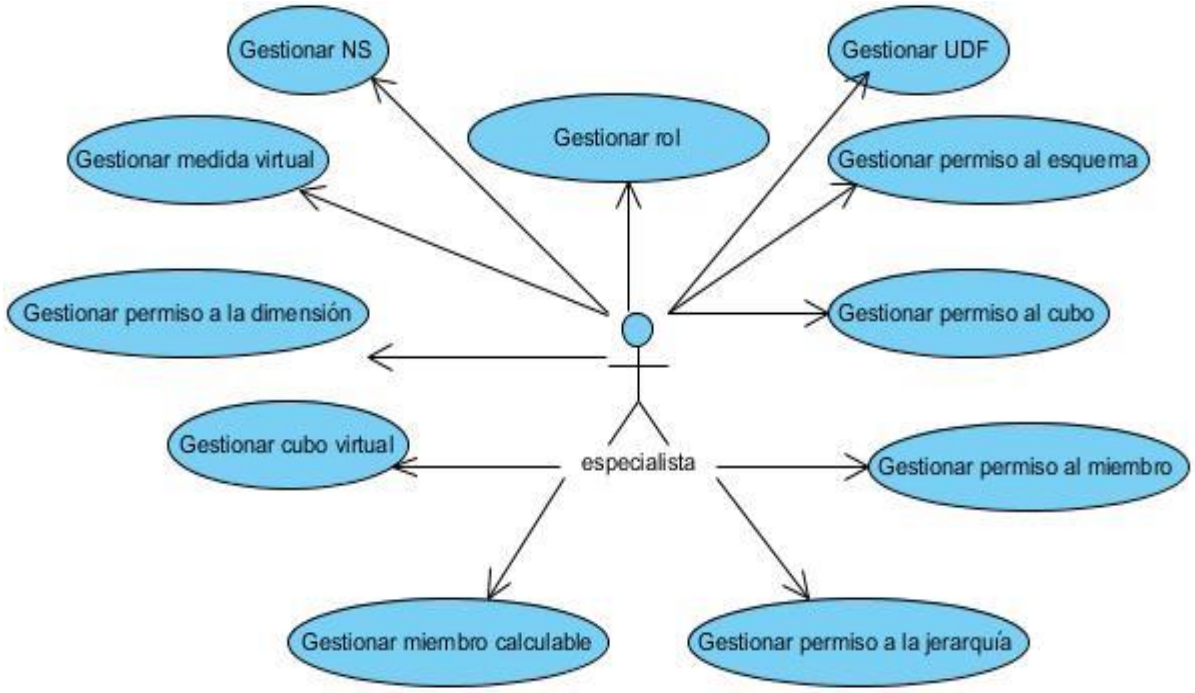


Fig. 6 Diagrama de caso de uso del sistema.

Patrones de CU

El comportamiento de estos CU son estructurados y organizados por patrones de CU que son a su vez comportamientos que deben existir en el sistema, ayudan a describir qué es lo que el sistema debe hacer, es decir, describen el uso del sistema y cómo este interactúa con los usuarios. Estos patrones se utilizan generalmente como plantillas que describen como debería ser estructurados y organizados los casos de uso. Dichos patrones capturan mejores prácticas para modelar CU (Adolp, y otros, 2002). Los RF definidos con anterioridad se agrupan a través del patrón CRUD (acrónimo de las palabras es inglés *Create, Read, Update, Delete* traducidas al español crear, leer, actualizar y eliminar); existen dos tipos de CRUD el completo y el parcial.

CRUD completo

Este patrón consta de un CU, llamado Información CRUD o Gestionar información modela todas las operaciones que pueden ser realizadas sobre una parte de la información de un tipo específico, tales como creación, lectura, actualización y eliminación. Suele ser utilizado cuando todos los flujos contribuyen al mismo valor del negocio, y a la vez son cortos y simples.

CRUD parcial

Algunas de las alternativas del CU pueden ser modeladas como caso de uso independiente. Aplicabilidad: este patrón es preferible cuando uno de los flujos alternativos del CU es más significativo, muy largo o mucho más complejo que el patrón completo.

Para la agrupación de los CU identificados anteriormente se utiliza el CRUD completo, pues todos los elementos son significativos que son el crear, el modificar, el eliminar y el visualizar presente en los CU 1 al CU 12.

Descripción de los casos de uso del sistema

Permiten conocer más sobre quién/qué espera algo del sistema (descripción de actores) y cómo se comporta el sistema para que ese actor logre el objetivo que desea (descripción del sistema al ser inicializado el CU). El comportamiento de un CU se puede especificar describiendo un flujo de eventos de forma textual, lo suficientemente claro que lo entienda fácilmente el cliente. Es importante especificar cómo y cuándo se empieza y acaba el caso de uso (Pinciroli, 2010).

A continuación un ejemplo de una descripción de un CU.

Tabla. 5 Descripción del CU. Gestionar cubo virtual.

Objetivo	Con este CU se pretende que el especialista realice las acciones de adicionar, visualizar, modificar y eliminar sobre un cubo virtual es decir gestionar el mismo.
Actores	Especialista.
Resumen	El CU inicia cuando el especialista va a realizar algunas de las siguientes operaciones ➢ Adicionar cubo virtual: cuando existen al menos dos cubos virtuales y una dimensión es común, el especialista adiciona un cubo virtual, finalizando así el CU. ➢ Visualizar cubo virtual: cuando el especialista adiciona un cubo virtual, el sistema permite ver el cubo virtual adicionado, y así dar la posibilidad al especialista de modificar los cubos, finalizando así el CU. ➢ Modificar cubo virtual: cuando el especialista necesita modificar los datos de un cubo virtual existente, modifica los datos e indica actualizar los mismos,

	finalizando así el CU. ➢ Eliminar cubo virtual: cuando el especialista necesita borrar un cubo virtual existente, finalizando así el CU.
Complejidad	Alta.
Prioridad	Crítico.
Precondiciones	De acuerdo a la acción elegida por el especialista a. Adicionar cubo virtual: antes de adicionar un cubo virtual tiene que existir dos cubos y al menos una dimensión en común. b. Visualizar cubo virtual: para poder visualizar el cubo virtual primero debe ser adicionado. c. Modificar cubo virtual: para poder ser modificado el cubo virtual primero debe ser marcado el cubo virtual para así poder seleccionar la opción **"Adicionar Cubo Virtual".** d. Eliminar cubo virtual: para eliminar un cubo virtual primero debe ser seleccionado el cubo virtual para así poder ser eliminado.
Postcondiciones	En dependencia de la acción del especialista ➢ Se adiciona un nuevo cubo virtual. ➢ Se visualiza el cubo virtual adicionado. ➢ Se modifica un cubo virtual existente. ➢ Se elimina un cubo virtual.

Flujo de eventos

Flujo básico <Gestionar cubo virtual>

	Actor	**Sistema**
	Selecciona cual opción desea realizar: a. Adicionar cubo virtual. b. Visualizar cubo virtual. c. Modificar cubo virtual.	En dependencia de la acción solicitada por el especialista, muestra la interfaz correspondiente: a. Adicionar cubo virtual: ir a la sección 1 **"Adicionar cubo virtual"**.

	d. Eliminar cubo virtual.	b. Visualizar cubo virtual: ir a la sección 2 **"Visualizar cubo virtual".** c. Modificar cubo virtual: ir a la sección 3 **"Modificar cubo virtual".** d. Eliminar cubo virtual: ir a la sección 4 **"Eliminar cubo virtual".**
Sección 1: "Adicionar cubo virtual"		
Flujo básico <Adicionar cubo virtual>		
	Actor	**Sistema**
1.	Selecciona la opción **"Adicionar Cubo Virtual"** si no existen dos hechos y una dimensión en común ver flujo alterno **<Dos hechos y una dimensión>.**	2. Muestra una interfaz para adicionar un cubo virtual con los parámetros Nombre, Descripción, Encabezado, Visible y Habilitado, donde estos últimos *JCheckBox* se encuentran activados y el campo Nombre es obligatorio.
3.	Ingresa los datos del cubo virtual y selecciona la opción aceptar "**Aceptar**".	4. Verifica los datos si existe un cubo virtual con el mismo nombre ver flujo alterno **<Cubo existe>,** o si este campo se encuentra vacío, ver flujo alterno **<Nombre vacío>.**
		5. Registra los datos y adiciona el cubo virtual. ***Finaliza el CU***
Flujos alternos al paso 1		
Nº Evento <Dos hechos y una dimensión>		
	Actor	**Sistema**
		1.1 Emite un mensaje de error **"No se puede crear cubo virtual. Deben existir dos hechos y una dimensión en común".** ***Se regresa al paso 1***
Flujos alternos al paso 4		

Nº Evento <Nombre vacío>		
	Actor	**Sistema**
		4.1 Emite un mensaje de error "**El campo nombre es obligatorio"**. ***Se regresa al paso 2***
Flujos alternos al paso 4		
Nº Evento <Cubo existe>		
	Actor	**Sistema**
		4.2 Emite un mensaje de error **"Ya existe un cubo virtual con ese nombre"**. ***Se regresa al paso 2***
Sección 2: "Visualizar cubo virtual"		
Flujo básico <Visualizar cubo virtual>		
	Actor	**Sistema**
1.	Adiciona un cubo virtual véase **Sección 1.**	2. Visualiza los cubos virtuales adicionados en el panel lateral izquierdo de la interfaz principal. ***Finaliza el CU***
Sección 3: "Modificar cubo virtual"		
Flujo básico <Modificar cubo virtual>		
	Actor	**Sistema**
1.	Marca el cubo que desea modificar de la lista de cubos virtuales almacenados y selecciona la opción **"Adicionar Cubo Virtual"**.	2. Muestra una interfaz para adicionar un cubo virtual con los parámetros Nombre, Descripción, Encabezado, Visible y Habilitado, donde estos últimos *JCheckBox* se encuentran activados y el campo Nombre es obligatorio.
3.	Realiza los cambios y selecciona la opción aceptar.	4. Verifica si el cubo ya existe de ser así ver flujos alternos **<Cubo existe>** o si este campo se encuentra vacío, ver flujo alterno **<Nombre vacío>** terminado así el CU**.**

		5. Registra los datos y adiciona el cubo virtual. ***Finaliza el CU***
Sección 4: "Eliminar cubo virtual"		
Flujo básico <Eliminar cubo virtual>		
	Actor	**Sistema**
1.	Marca el cubo virtual de la lista de cubos virtuales existente, el que desea eliminar y selecciona la opción **"Eliminar"**.	2. Muestra un mensaje de confirmación **"¿Esta seguro que desea eliminar el cubo virtual?".**
3.	Presiona la opción si, sino ver flujos alternos **<No cancelar>**	4. Actualiza y elimina el cubo virtual. ***Finaliza el CU***
Flujos alternos al paso 3		
Nº Evento <No cancelar >		
	Actor	**Sistema**
		3.1 Actualiza y no elimina el cubo virtual. ***Finaliza el CU***

Relaciones	CU Incluidos	No procede
	CU Extendidos	No procede
Requisitos no funcionales	No procede	
Asuntos pendientes	No procede	

Ver el resto de las descripciones de los CU en el expediente de proyecto de OpenUp ubicado en la carpeta requisitos llamado el artefacto Modelo de sistema v2.0.

2.5 Arquitectura de software propuesta

La arquitectura de software puede considerarse como el puente entre los requerimientos del sistema y la implementación (Hofmeister, y otros, 2000). En la que se especifica la estructura del sistema, entendida como la organización de componentes y relaciones entre ellos, reduce los riesgos asociados con la

construcción del software (Pressman, 2009). Es un conjunto de patrones que proporcionan un marco de referencia necesario para guiar la construcción de un software, permitiendo a los programadores, analistas y todo el conjunto de desarrolladores del software compartir una misma línea de trabajo y cubrir todos los objetivos y restricciones de la aplicación. Es considerada el nivel más alto en el diseño de la arquitectura de un sistema puesto que establecen la estructura, funcionamiento e interacción entre las partes del software (Morán, 2013).

Patrón arquitectónico

El patrón arquitectónico es quien define la estructura básica de la aplicación siendo una plantilla de construcción que incluye reglas y pautas para su organización (Larman, 2005). La herramienta VP provee una interfaz de programación de aplicaciones que permite a los desarrolladores implementar y reutilizar clases e interfaces para desarrollar funciones agregadas de software. Para definir la arquitectura de la extensión propuesta, resulta fundamental regirse por los elementos arquitectónicos definidos en el API de la herramienta. Esta propone una arquitectura basada en el patrón arquitectónico Modelo-Vista-Controlador (MVC). Este patrón separa los datos y la lógica de negocio de una aplicación de la interfaz de usuario y el módulo encargado de gestionar los eventos y las comunicaciones. Para ello MVC propone la construcción de tres componentes distintos que son el modelo, la vista y el controlador, es decir, por un lado define componentes para la representación de la información y por otro lado para la interacción del usuario. Este patrón de diseño se basa en las ideas de reutilización de código y la separación de conceptos, características que buscan facilitar la tarea de desarrollo de aplicaciones y su posterior mantenimiento.

- **El Modelo**: es la representación de la información con la cual el sistema opera, por lo tanto gestiona todos los accesos a dicha información, tanto consultas como actualizaciones, implementando también los privilegios de acceso que se hayan descrito en las especificaciones de la aplicación (lógica de negocio). Envía a la vista aquella parte de la información que en cada momento se le solicita para que sea mostrada (típicamente a un usuario). Las peticiones de acceso o manipulación de información llegan al modelo a través del controlador. Ver la figura 7 donde se representan las clases del modelo de la extensión.

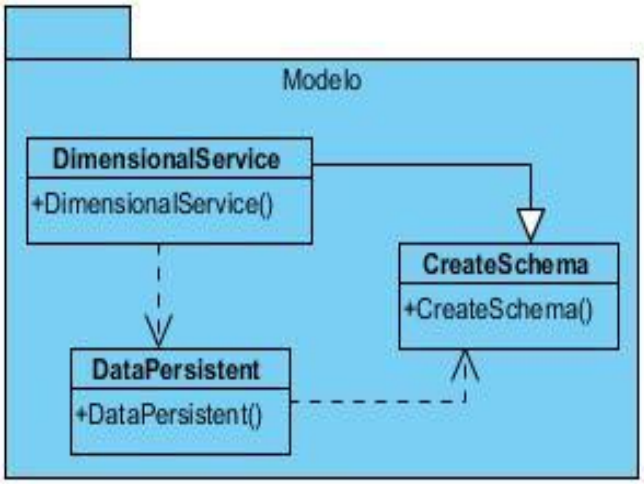


Fig. 7 Clases del modelo de la extensión.

- **El Controlador**: responde a eventos (usualmente acciones del usuario) e invoca peticiones al modelo cuando se hace alguna solicitud sobre la información (por ejemplo, editar un documento o un registro en una base de datos). También puede enviar comandos a su vista asociada si se solicita un cambio en la forma en que se presenta de modelo (por ejemplo, desplazamiento o *scroll* por un documento o por los diferentes registros de una base de datos), por tanto se podría decir que el 'controlador' hace de intermediario entre la vista y el modelo. Ver la figura 8 donde se representan las clases de las controladoras de la extensión.

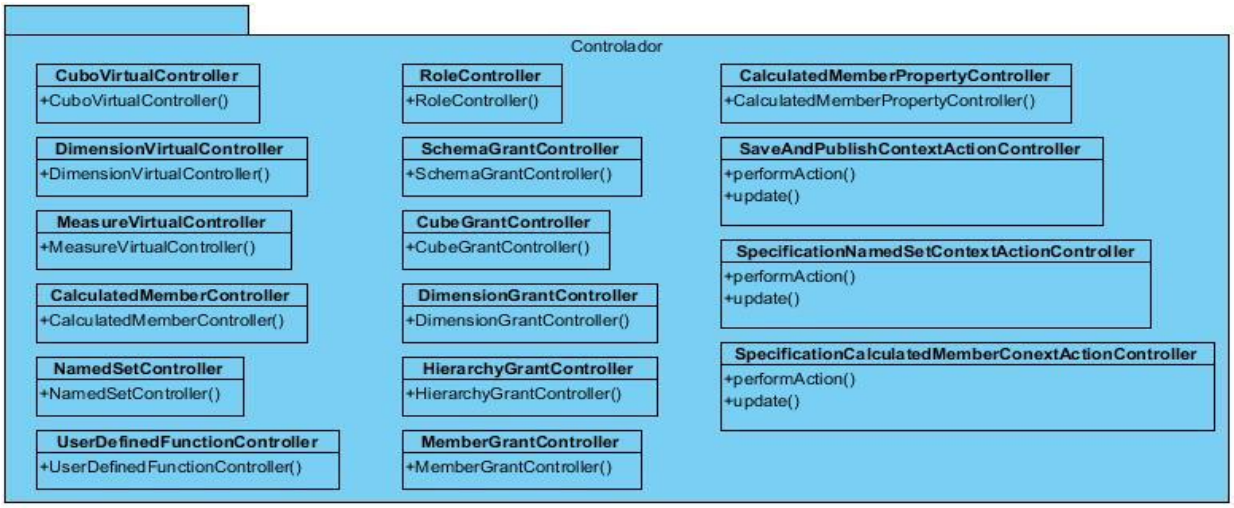


Fig. 8 Clases controladoras de la extensión.

- **La Vista**: presenta el modelo (información y lógica de negocio) en un formato adecuado para interactuar (usualmente la interfaz de usuario) por tanto requiere de dicho modelo la información que debe representar como salida. Ver figura 9 las clases de la vista de la extensión.

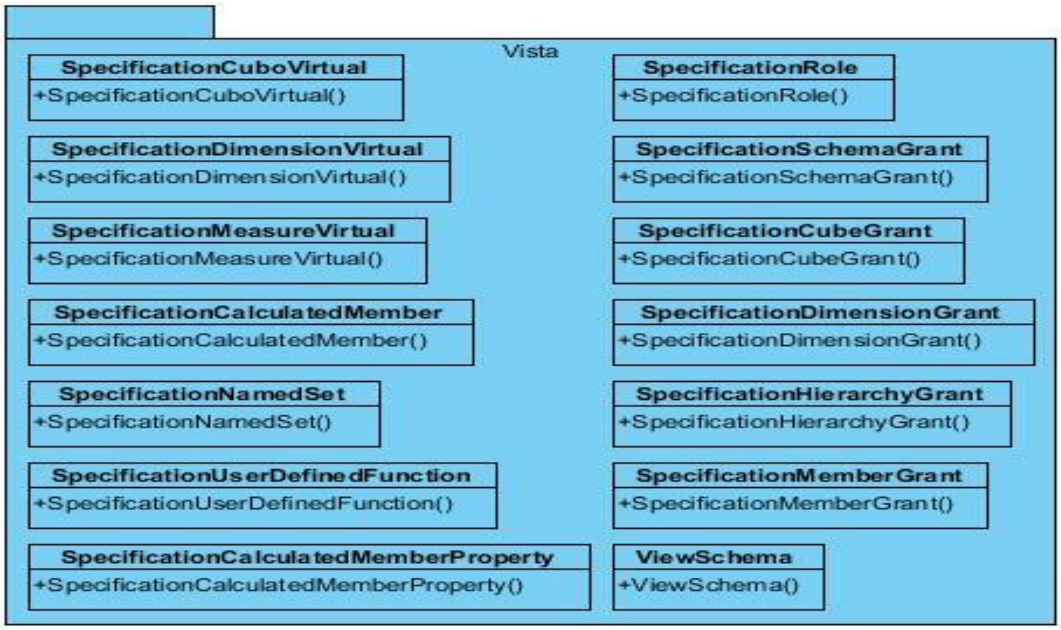


Fig. 9 Clases de la vista de la extensión.

2.6 Diagrama de clase del diseño

El Diagrama de clase es el diagrama principal de diseño y análisis para un sistema. En él, la estructura de clases del sistema se especifica, con relaciones entre clases y estructuras de herencia. Durante el análisis del sistema, el diagrama se desarrolla buscando una solución ideal. Durante el diseño, se usa el mismo diagrama y se modifica para satisfacer los detalles de las implementaciones.

El diagrama de clase del diseño describe gráficamente las especificaciones de las clases de *software* y de las interfaces en una aplicación. Normalmente contiene la siguiente información: clases, asociaciones, atributos, interfaces con sus operaciones y constantes, métodos, información sobre los tipos de los atributos, navegabilidad y dependencias (Larman, 2004). A continuación se muestra el diagrama de clase del CU Gestionar cubo virtual en la figura 10.

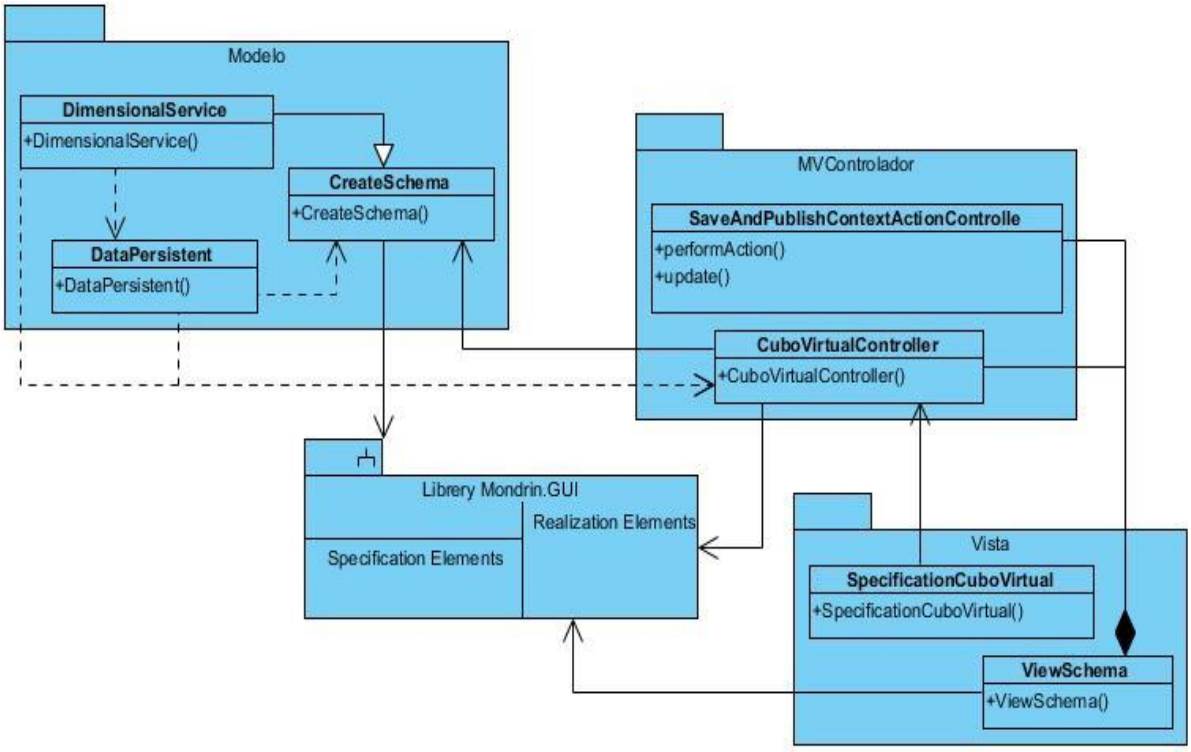


Fig. 10 Diagrama de clase del CU Gestionar cubo virtual.

El diagrama de clase que presenta la extensión contiene en el paquete Vista a las interfaces que interactúan con el usuario, cada una de ellas contiene un objeto de la clase controladora que le corresponde. Existe una controladora para cada prototipo de la interfaz principal, agrupadas en el paquete Controlador, donde estas envían los datos escritos por el usuario en las interfaces, a las clases del paquete Modelo, encargadas de guardar y procesar los datos.

Ver el resto de los diagramas de clase CU en el expediente de proyecto de OpenUp en la carpeta arquitectura y diseño documento llamado Modelo de diseño v2.0.

Patrones de diseño

Describen los principios fundamentales del diseño de objetos. Estos son agrupados fundamentalmente por dos grandes grupos conocidos como: Patrones de Software para la asignación General de Responsabilidad (acrónimo GRASP por sus siglas en inglés *General Responsability Assignment Software Patterns*) y La Banda de los cuatro (acrónimo GoF por sus siglas en inglés *Gang of Four*).

Patrones GRASP

Los patrones básicos de GRASP son utilizados para describir los principios fundamentales de diseño de objetos para la asignación de responsabilidades. El nombre se eligió para indicar la importancia de captar estos principios, si se quiere diseñar eficazmente el software orientado a objetos (Larman, 2005). Este es el principal objetivo del uso de los siguientes patrones GRASP empleados en el desarrollo de la extensión:

- **Experto**: la responsabilidad de asignar una labor a la clase que tiene o puede tener los datos necesarios para cumplir determinada responsabilidad, es la solución que pretende dar este patrón ante el problema de cómo realizar la asignación de la forma más eficiente posible. Si estas asignaciones de responsabilidades se hacen adecuadamente, los sistemas tienden a ser más fáciles de entender, mantener y ampliar, lo que nos ofrece la garantía de poder reutilizar los componentes en futuras aplicaciones (Larman, 2004). Este patrón se ve reflejado en la clase CreateSchema ya que esta le asigna una labor a la clase que contiene o puede tener los datos necesarios para cumplir determinada responsabilidad, esta clase es DataPersistent que es la que tiene la responsabilidad de registrar los datos del sistema. (Ver figura 11).

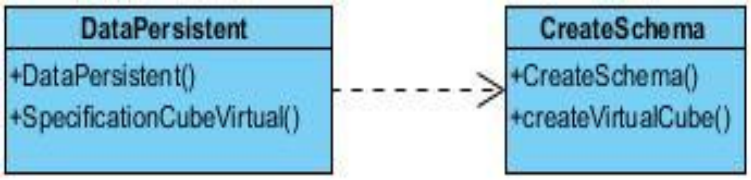


Fig. 11 Patrón experto.

- **Bajo acoplamiento**: es la idea de tener las clases lo menos relacionadas que se pueda. De tal forma que en caso de producirse una modificación en alguna de ellas, se tenga la mínima repercusión posible en el resto de clases, potenciando la reutilización y disminuyendo la dependencia entre las clases (Larman, 2004). El uso de este patrón se ve reflejado en la clase DataPersistent que es la clase donde se captura los datos de la interfaz permitiendo crear, modificar y eliminar los elementos dimensionales actualizando las funciones del CreateSchema sin tener en cuenta que si se realiza algún cambio en esta clase, no sea afectada DataPersistent. (Ver figura 11).
- **Controlador**: el patrón controlador es un patrón que sirve como intermediario entre una determinada interfaz y el algoritmo que la implementa, de tal forma que es la que recibe los datos del usuario y la que los envía a las distintas clases según el método llamado (Larman, 2004). Es el encargado de asignar la responsabilidad del manejo de uno de los eventos del sistema, a una clase facultada de

atender determinada funcionalidad, solucionando así el problema fundamental de este patrón, al saber quién debería ocuparse de dicho evento. En todos los casos, si se recurre a un diseño orientado a objetos, hay que elegir los controladores que manejen esos eventos de entrada. Este patrón se ve reflejado en las clases Controller ya que estas saben quién debería de ocuparse de guiar los eventos. Este patrón se ve reflejado en la clase CuboVirtualController. (Ver figura 12).

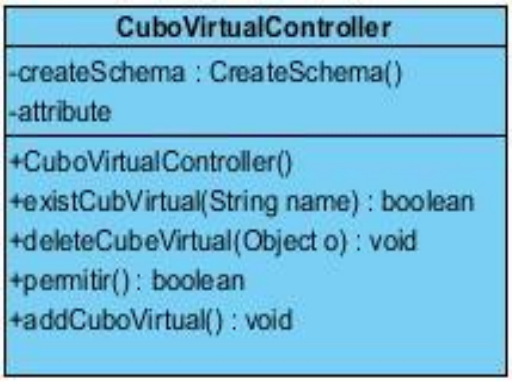


Fig. 12 Patrón controlador.

- **Alta cohesión**: una clase de alta cohesión posee un número relativamente pequeño, con una importante funcionalidad relacionada y poco trabajo por hacer. Colabora con otros objetos para compartir el esfuerzo si la tarea es grande (Larman, 2004). Este patrón se ve reflejado en la mayoría de las clases del diseño ya que estas implementan solo las funcionalidades que les corresponden. Un ejemplo específico de este patrón es la relación entre las clases SaveAndPublishContextActionController y ViewSchema, pues la controladora solo se encarga de crear la interfaz, y esta de implementar los métodos que le conciernen. (Ver figura 13).

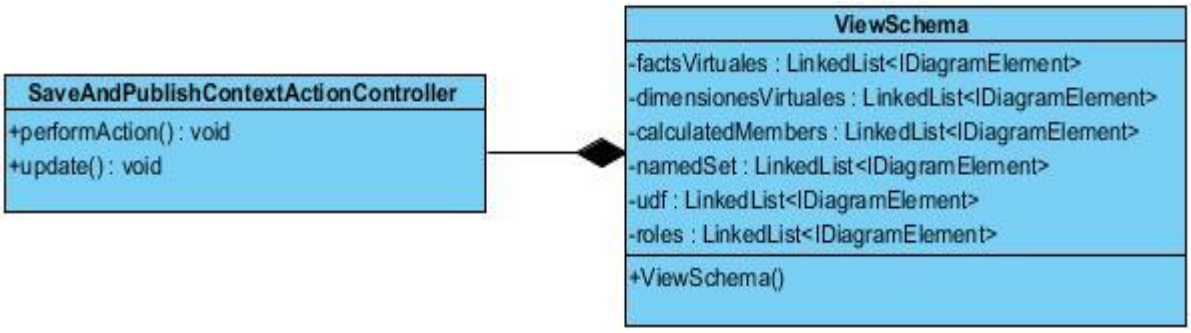


Fig. 13 Patrón alta cohesión.

- **Creador**: el patrón Creador guía la asignación de responsabilidades relacionadas con la creación de objetos, tarea muy frecuente en los sistemas orientados a objetos. El propósito fundamental de este patrón es encontrar un creador que se conecta con el objeto producido en cualquier evento (Larman, 2004). Este patrón se evidencia en las clases Controller ya que estas crean una instancia de la vista encargada de interactuar con el usuario según el evento que se genere.Ejemplo: SaveAndPublishContextActionController. (Ver figura 11).

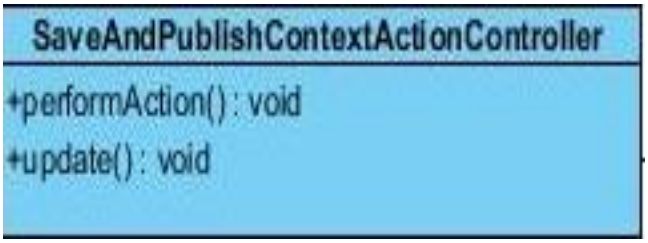


Fig. 14 Patrón creador.

Patrones GoF

Dentro de los patrones de diseño se encuentran los patrones GoF. Estos patrones definen el comportamiento entre las clases y los objetos. En el desarrollo del módulo se utilizaron los siguientes patrones GoF:

Entre los patrones utilizados para el desarrollo de la extensión se encuentran Iterador y el Solitario agrupado como patrones de comportamiento y de creación respectivamente. Estos se encuentran representados a través de una descripción detallada acerca del propósito que recoge cada uno y la aplicación que tienen durante el desarrollo de la extensión como se muestra a continuación:

- **Solitario:** 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) (Larman, 2004). En la extensión se utilizó el patrón en la implementación de la clase SaveAndPublishContextActionController creando una instancia única de la clase interfaz ViewSchema, evitando la duplicidad a través del punto de acceso global que es capaz de definir. (Ver figura 13).

- **Iterador**: el patrón iterador, es de tipo comportamiento a nivel de objetos. 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 (Larman, 2004). Este patrón es utilizado en el método permitir (), encargado en buscar la cantidad de hechos que contiene una dimensión para ser utilizado en la validación cuando se crea un cubo virtual, dicho método pertence a la clase CuboVirtualController. (Ver figura 12).

2.7 Conclusiones del capítulo

Se concluye, una vez culminado el capítulo que servirá de ayuda a los usuarios, desarrolladores, e interesados a entender el análisis y diseño para la implementación de la extensión. Un elemento importante que sirvió como punto de partida para el desarrollo, fue el modelo del dominio y la explicación detallada del mismo, para mejor entendimiento de la situación planteada. Las descripciones de las funcionalidades a implementar, quedaron establecidas en cuarenta y ocho RF agrupados en doce CU y cinco RnF para el correcto funcionamiento de la extensión. A través de la clase del diseño se hace referencia al patrón arquitectónico MVC utilizado para la implementación teniendo en cuenta los patrones de diseño GRASP: experto, bajo acoplamiento, alta cohesión, controlador y el creador; de la familia GoF el solitario y el iterador.

Capítulo III: Implementación y pruebas de la extensión para modelar AD en la herramienta VP, versión 2.0.

En el presente capítulo se organizan los componentes mediante el diagrama de componentes para definir la estructura en capas de la aplicación y se realiza además el diagrama de despliegue del sistema. Se define además la estrategia de prueba a seguir y los resultados de las mismas una vez aplicadas al sistema, contribuyendo a mejorar la calidad, usabilidad e identificar fallos en la aplicación.

3.1 Implementación

La implementación inicia con los resultados obtenidos de la fase de diseño. El modelo de implementación contiene el diagrama de componente; estos describen los componentes a construir, su organización y la dependencia entre nodos físicos en los que funcionará el sistema, los cuales comienzan a desarrollarse en el flujo de trabajo de diseño y que se perfeccionan en la implementación.

Diagrama de componente

Describen los elementos físicos del sistema y sus relaciones. Muestran las opciones de realización incluyendo código fuente, binario y ejecutable. Los componentes representan todos los tipos de elementos software que entran en la fabricación de aplicaciones informáticas. Pueden ser simples archivos, paquetes, bibliotecas cargadas dinámicamente, etc. En la vista de la figura siguiente se visualizan los componentes principales del sistema.

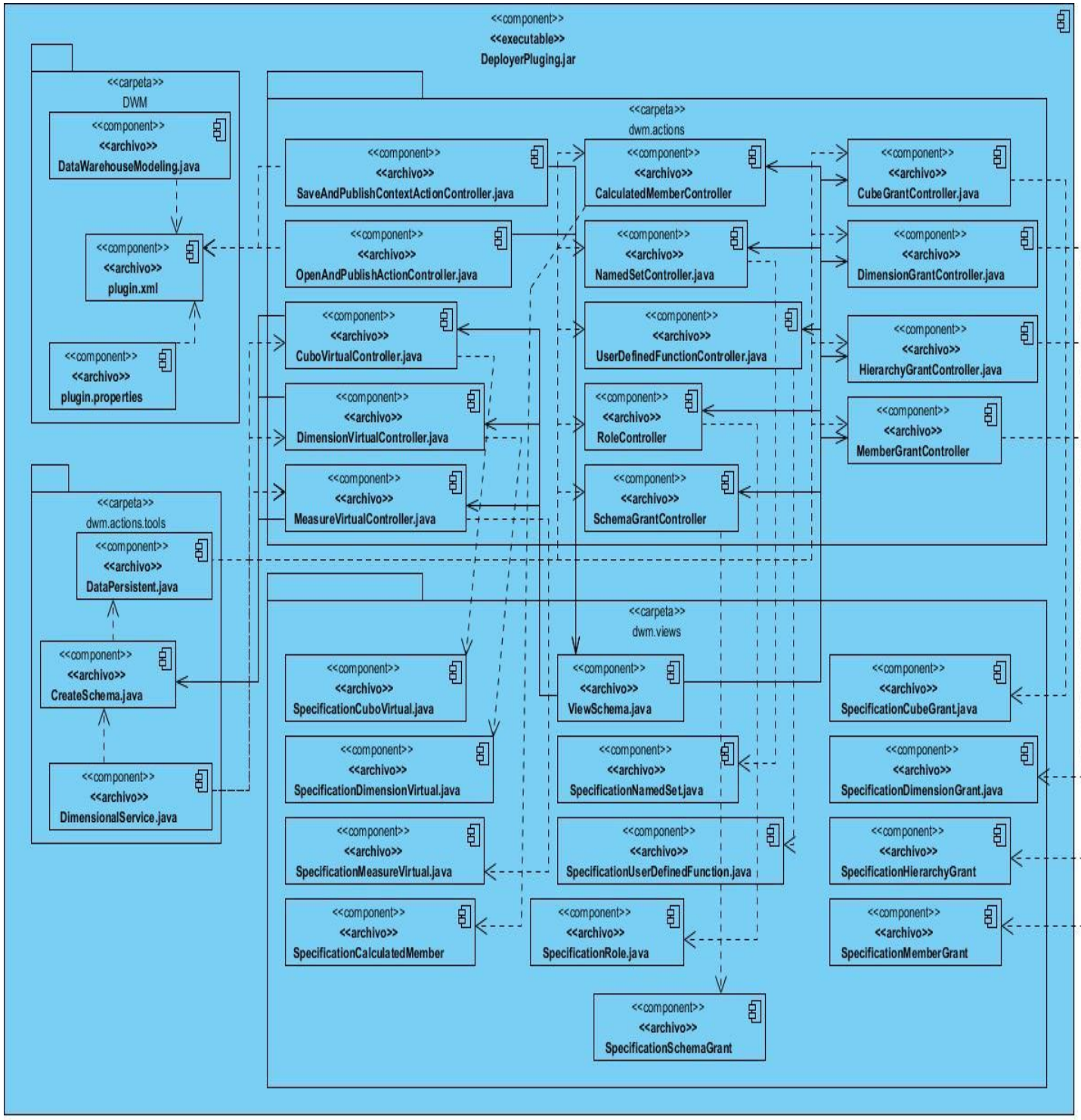


Fig. 15 Diagrama de componente de la extensión.

El diagrama de componente de la extensión está compuesto por los componentes:

- Carpeta: específica un paquete que contiene en su interior uno o varios componentes.
- Archivo: especifica un componente que representa un documento que contiene código fuente.
- Librería: especifica una biblioteca de objetos estática o dinámica.
- Ejecutable: especifica un componente que se puede ejecutar en un nodo.

El diagrama de componente de la extensión muestra un paquete llamado DWM que contiene las clases DataWarehouseModeling.java, plugin.xml y plugin.propeties que están relacionadas con la configuración de la extensión. El paquete dwm.actions agrupa las clases referentes a las acciones que son accesibles a través de la interfaz principal del VP o del menú de herramientas; estas clases tienen la terminación ActionController.java. El paquete dwm.actions.tools contiene las clases DataPersistent.java, Createschema.java y DimensionalService.java relacionadas con los datos que se utilizarán en la extensión. El paquete dwm.views contiene las interfaces de la extensión donde todas comienzan por Specification seguido del nombre y terminado en .java excepto ViewSchema.java que es la interfaz principal de la extensión. La clase DataWarehouseModeling.java carga y descarga de la extensión mediante la clase VPPluginInfo que provee a la librería openapi.jar que es proporcionada por la herramienta VP, capturando la información definida en el archivo plugin.xml, la función de este último consiste en la configuración definiendo el nombre, descripción, proveedor y las acciones a ejecutar de la extensión.

Estructura del desarrollo

La estructura de desarrollo para el IDE NetBeans está compuesta por un paquete principal que lleva el nombre de la extensión, además de poseer tres paquetes asociados a él. El primero llamado igual que la extensión (DWN.actions) contiene las acciones a realizar. El segundo contiene un conjunto de clases necesarias para su configuración (DWN.actions.tools) y el tercero contiene las interfaces y los formularios (DWN.views). Para la descripción de cada uno de estos paquetes que utiliza el IDE se emplea un ejemplo de un proyecto. A continuación se muestra el contenido por paquete:

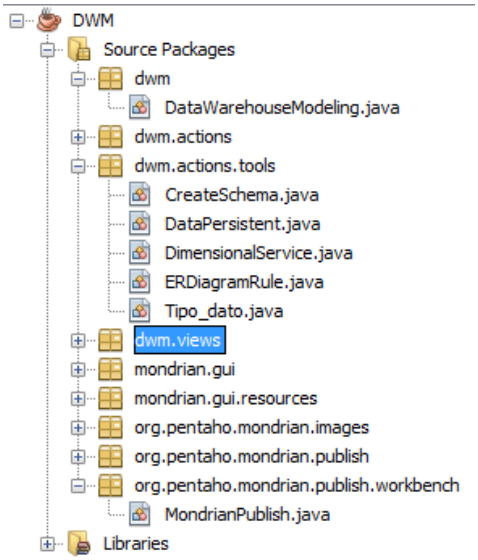


Fig. 16 Estructura general de los paquetes.

Integración con la herramienta

Para la integración de la extensión, VP propone una estructura de despliegue compuesta por una carpeta con el nombre de *plugins*, que se encuentra dentro del directorio de instalación de esta misma herramienta. La carpeta *plugins* se estructura según la siguiente forma:

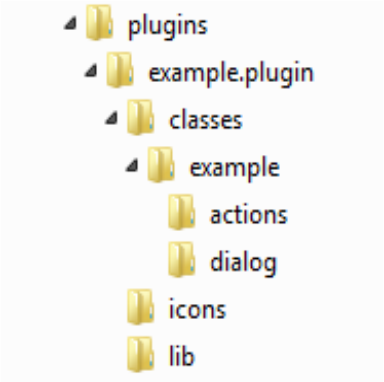


Fig. 17 Estructura de despliegue.

Si se observa la estructura de implementación de la extensión, es diferente a la de integración con la herramienta, lo que dificulta el proceso de pruebas al integrar la extensión implementada para el VP. La solución a este inconveniente es utilizar una herramienta de despliegue de extensiones que facilite la integración del mismo con VP; pasando los valores a la herramienta como es directorio de origen de la

implementación en el IDE y la dirección donde será desplegada la extensión. A continuación la interfaz para el despliegue de la extensión.

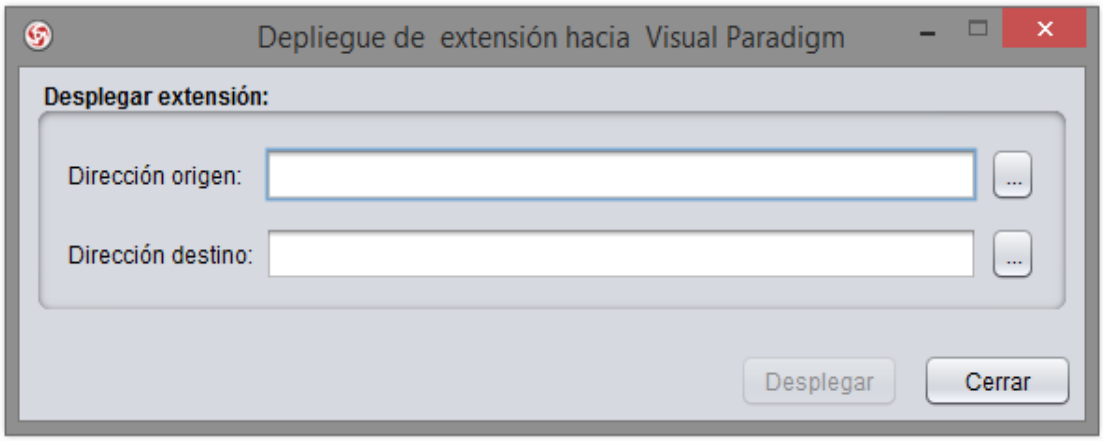


Fig. 18 Interfaz para el despliegue de la extensión.

3.2 Estándares de codificación

Un estándar de codificación completo comprende todos los aspectos de la generación de código. Si bien los programadores deben implementar un estándar de forma prudente, éste debe tender siempre a lo práctico. Un código fuente completo debe reflejar un estilo armonioso, como si un único programador hubiera escrito todo el código de una sola vez.

Usar técnicas de codificación sólidas y realizar buenas prácticas de programación con vistas a generar un código de alta calidad es de gran importancia para la calidad del software y para obtener un buen rendimiento. Además, si se aplica de forma continua un estándar de codificación bien definido, se utilizan técnicas de programación apropiadas y posteriormente se efectúan revisiones de rutinas, caben muchas posibilidades de que un proyecto de software se convierta en un sistema de software fácil de comprender y de mantener.

Aunque el propósito principal para llevar a cabo revisiones del código a lo largo de todo el desarrollo es localizar defectos en el mismo, las revisiones también pueden afianzar los estándares de codificación de manera uniforme. La adopción de un estándar de codificación sólo es viable si se sigue desde el principio hasta el final del proyecto de software. No es práctico, ni prudente, imponer un estándar de codificación una vez iniciado el trabajo (Studio, 2003).

En el desarrollo de la extensión para la herramienta VP sobre la plataforma Java, se pusieron en práctica los estándares de codificación de la versión 1.0 de la extensión para el trabajo de la implementación de una forma más organizada adaptada a dicha versión:

Organización de los ficheros: serán evitados los ficheros de gran tamaño que contengan más de mil líneas de código, pues en ocasiones el tamaño excesivo provoca que la clase no encapsule un comportamiento claramente definido, albergando una gran cantidad de métodos que realizan tareas funcional o conceptualmente heterogéneas.

Tamaño y organización de las líneas de código: la longitud de línea no debe superar los ochenta caracteres. En caso de que una expresión ocupe más de una línea, esta se podrá romper o dividir tras una coma o antes de un operador y la nueva línea debe estar alineada con el inicio de la expresión al mismo nivel que la línea anterior.

Declaraciones: toda variable local tendrá que ser inicializada en el momento de su declaración, a no ser que su valor inicial dependa de algún valor que tenga que ser calculado previamente. Las declaraciones deben situarse al principio de cada bloque principal en el que se utilicen y nunca en el momento de su uso.

En las declaraciones de los métodos, no debe incluirse ningún espacio entre el nombre del método y el paréntesis inicial del listado de parámetros. Los métodos se separarán entre sí mediante una línea en blanco.

Sentencias: el caracter inicio de bloque debe situarse al final de la línea que inicia el bloque. El caracter final de bloque debe situarse en una nueva línea tras la última línea del bloque y alineada con respecto al primer caracter de dicho bloque. Todas las sentencias de un bloque deben encerrarse entre llaves, aunque el bloque conste de una única sentencia.

Espacios en blanco: se utilizarán espacios en blanco entre una palabra clave y un paréntesis, tras cada coma en un listado de argumentos, para separar un operador binario de sus operandos, excepto en el caso del operador ("."), para separar las expresiones incluidas en la sentencia "for" y al realizar el moldeo o "casting" de clases.

Nomenclatura de identificadores

Paquetes: los nombres de los paquetes se escribirán siempre en letras minúsculas para evitar que entren en conflicto con los nombres de clases e interfaces.

Clases e interfaces: los nombres de clases deben ser sustantivos y deben tener la primera letra en mayúscula. Si el nombre es compuesto, cada palabra componente deberá comenzar con mayúscula. Debe evitarse el uso de acrónimos o abreviaturas. Toda interfaz se nombrará con el prefijo "I" para diferenciarla de la clase que la implementa (que tendrá el mismo nombre sin el prefijo "I").

Métodos: los métodos deben ser verbos escritos en minúscula. Cuando el método esté compuesto por varias palabras cada una de ellas tendrá la primera letra en mayúscula.

Variables: las variables se escribirán siempre en minúscula. Las variables compuestas tendrán la primera letra de cada palabra componente en mayúscula. Estas nunca podrán comenzar con el caracter "_" o "$". Los nombres de variables deben ser cortos y debe evitarse el uso de nombres de variables con un solo caracter, excepto para variables temporales.

Constantes: todos los nombres de constantes tendrán que escribirse en mayúscula. Cuando los nombres de constantes sean compuestos las palabras se separarán entre sí mediante el caracter de subrayado "_".

Buenas prácticas de programación

Visibilidad de atributos de instancia y de clase: los atributos de instancia y de clase serán siempre privados, excepto cuando tengan que ser visibles en subclases herederas; en tales casos serán declarados como protegidos. El acceso a los atributos de una clase se realizará por medio de los métodos "*get*"[8] y "*set*"[9] correspondientes.

Referencias a miembros de una clase: debe evitarse el uso de objetos para acceder a los miembros de una clase (atributos y métodos estáticos). Debe utilizarse en su lugar el nombre de la clase.

Asignación sobre variables: se deben evitar las asignaciones de un mismo valor sobre múltiples variables en una misma sentencia, pues dichas sentencias suelen ser difíciles de leer. No se deben utilizar asignaciones embebidas o anidadas.

[8] Un método get obtiene el valor de un atributo específico.

[9] Un método set modifica o configura el valor de un atributo específico.

Paréntesis: se deben utilizar paréntesis en expresiones que incluyan distintos tipos de operadores para evitar problemas de precedencia de operadores.

3.3 Pruebas del sistema

La prueba del software es un elemento crítico para garantizar la calidad del software y representa una revisión de las especificaciones, del diseño y de la codificación. Una vez generado el código fuente es necesario probar el software para descubrir y corregir la mayor cantidad de errores posibles antes de ser entregado. Su objetivo es diseñar una serie de casos de prueba que tengan una alta probabilidad de encontrar errores (Pressman, 2009).

Una prueba de software permite identificar y corregir errores, lo que va a permitir que el sistema entregado al usuario final tenga la calidad requerida. En la actualidad existen diferentes tipos de pruebas que se le pueden aplicar a un software para garantizar su calidad, estás se aplicarán teniendo en cuenta las características del sistema.

En la presente investigación se definen las pruebas que a continuación se especifican.

Pruebas funcionales

Se le denominan a las pruebas de software que tienen por objetivo probar que los sistemas desarrollados, cumplan con las funciones específicas para los cuales han sido creados. Este tipo de pruebas tiene dos métodos para ser aplicadas, el método caja blanca que está orientado directamente al código y el método caja negra orientado a las funcionalidades del sistema. Por las características de la solución se selecciona como método a aplicar caja negra, que se centran en lo que se espera de un módulo, es decir, intentan encontrar casos en que el módulo no se atiene a su especificación; donde el probador se limita a proveer datos como entrada y estudiar la salida, sin preocuparse de lo que pueda estar haciendo el módulo por dentro. Existen distintas técnicas de caja negra para confeccionar los casos de prueba de caja negra; según el estudio se realizará la técnica de: particiones de equivalencia.

La partición de equivalencia es una técnica de prueba de caja negra que divide el campo de entrada de un programa en clases de datos de los que se pueden derivar casos de prueba. Esta se dirige a una definición de casos de prueba que descubran clases de errores, reduciendo así el número total de casos de prueba que hay que desarrollar.

Caso de prueba

Un caso de prueba se diseña según las funcionalidades descritas en los casos de uso. El propósito que se persigue con este artefacto es lograr una comprensión común de las condiciones específicas que la solución debe cumplir. Se comienza con la descripción de los casos de uso del sistema, como apoyo para las revisiones. Cada planilla de caso de prueba recoge la especificación de un CU, dividido en secciones y escenarios, detallando las funcionalidades descritas en él y describiendo cada variable que recoge el CU en cuestión. Para la siguiente investigación se definieron doce casos de prueba en correspondencia con los CU. A continuación se presenta un ejemplo de un escenario del caso de prueba Gestionar cubo virtual.

Descripción general: caso de prueba Gestionar cubo virtual, tiene como finalidad gestionar todo lo referente a los cubos virtuales ya sea adicionar, modificar, eliminar y visualizar.

Condiciones de ejecución

1. Tener instalada la extensión en el VP.
2. Existir dos cubos y una dimensión virtual.

Tabla. 6 SC Adicionar cubo virtual

Escenario	Descripción	Variable 1	Variable 2	Variable 3	Respuesta del sistema	Flujo central
EC 1.1 Adicionar cubo virtual.	En este escenario se realiza la adición de un nuevo cubo virtual al sistema correctamente.	V	V	V	Visualiza el cubo virtual creado.	Se selecciona la opción crear cubo virtual ubicada en los íconos de la interfaz principal. Se llenan los datos del cubo virtual. Se da clic en la opción "**Aceptar**"
		hecho_tienda	Tienda	Este cubo contendrá varias características de una tienda en específico.		
EC 1.2	Este escenario no	V	N/A	N/A	Muestra una	Se selecciona la

Adicionar cubo virtual (no existe cubo y dimensión en común.	debe permitir crear un cubo virtual si no existen dos cubos y una dimensión en común.	hecho_tienda	Tienda	Este cubo contendrá varias características de una tienda en específico.	notificación **"No se puede adicionar cubo virtual si no tiene al menos 2 hechos y una dimensión en común".**	opción crear cubo virtual.
EC 1.3 Adicionar cubo virtual (campo nombre vacío).	Este escenario no debe permitir crear un cubo virtual si el campo nombre está vacío.	I (vacío)	V Tienda	V Este cubo contendrá varias características de una tienda en específico.	Muestra una Muestra una notificación **"El campo nombre es obligatorio".**	Se selecciona la opción de crear cubo virtual. Se llenan los datos del cubo virtual. Se selecciona la opción "**Aceptar**".
EC 1.4 Adicionar cubo virtual (cubo existente).	Este escenario no debe permitir crear un cubo virtual si ya existe otro con el mismo nombre.	V hecho_tienda	N/A	N/A	Muestra una notificación **"Ya existe un cubo virtual con ese nombre".**	Se selecciona la opción de crear cubo virtual. Se llenan los datos del cubo virtual. Se selecciona la opción "**Aceptar**".

Tabla. 7 Descripción de las variables.

No	Nombre de campo	Clasificación	Valor Nulo	Descripción
1	Nombre	N/A	No	En este campo se le permite al usuario ponerle

				nombre del cubo virtual.
2	Descripción	N/A	Si	En este campo se le permite al usuario hacer una descripción del cubo virtual a crear.
3	Encabezado	N/A	Si	En este campo se le permite al usuario ponerle un encabezado al cubo virtual creado.

Las celdas de la tabla contienen V, I, o N/A: V indica válido, I indica inválido y N/A que no es necesario proporcionar un valor del dato en este caso, ya que es irrelevante.

Ver el resto de los casos de prueba en el expediente de proyecto de OpenUp ubicado en la carpeta implementación y pruebas documento llamado Casos de prueba v2.0.

Pruebas de aceptación

Estas pruebas se realizan cuando el sistema está listo para ser utilizado por el cliente. El cliente debe ser quien las realice, ayudado por especialistas de las mismas, para que sea el mismo usuario quien aporte los casos de prueba. Estas pruebas se caracterizan por:

- Participación activa del usuario, que debe ejecutar los casos de prueba ayudado por miembros del equipo de pruebas.
- Están enfocadas a probar los requisitos de usuario, o mejor dicho a demostrar que no se cumplen los requisitos, los criterios de aceptación o el contrato. Si no se consigue demostrar esto el cliente deberá aceptar el producto.
- Corresponden a la fase final del proceso de desarrollo de software.

Es muy recomendable que las pruebas de aceptación se realicen en el entorno en que se va a explotar el sistema (incluido el personal que lo maneje). En caso de un producto de interés general, se realizan pruebas con varios usuarios que reportarán sus valoraciones sobre el producto.

Resultados de las pruebas realizadas

Pruebas funcionales

Al realizar las pruebas funcionales, con el uso del método de caja negra a través de la técnica partición equivalente, se verificó el correcto funcionamiento de la extensión al cumplir con todos sus requisitos

funcionales. La realización de las pruebas permitió identificar los errores no detectados que hasta entonces, poseía la extensión. Se identificaron un total de catorce no conformidades; de las cuales fueron significativas con impacto alto cuatro, significativas con impacto medio cinco, no significativa con impacto alto tres, no significativa con impacto medio uno y no significativa con impacto bajo una, en la primera iteración; en la segunda fueron detectada un total de tres no conformidades dos significativas con impacto alto y medio respectivamente y una no significativa con impacto bajo.

La siguiente tabla muestra el resultado obtenido durante las tres iteraciones de pruebas efectuadas. En ella se recogen la cantidad de requisitos que intervienen durante la ejecución de las pruebas así como el número de no conformidades, las cuales fueron resueltas al final de cada iteración.

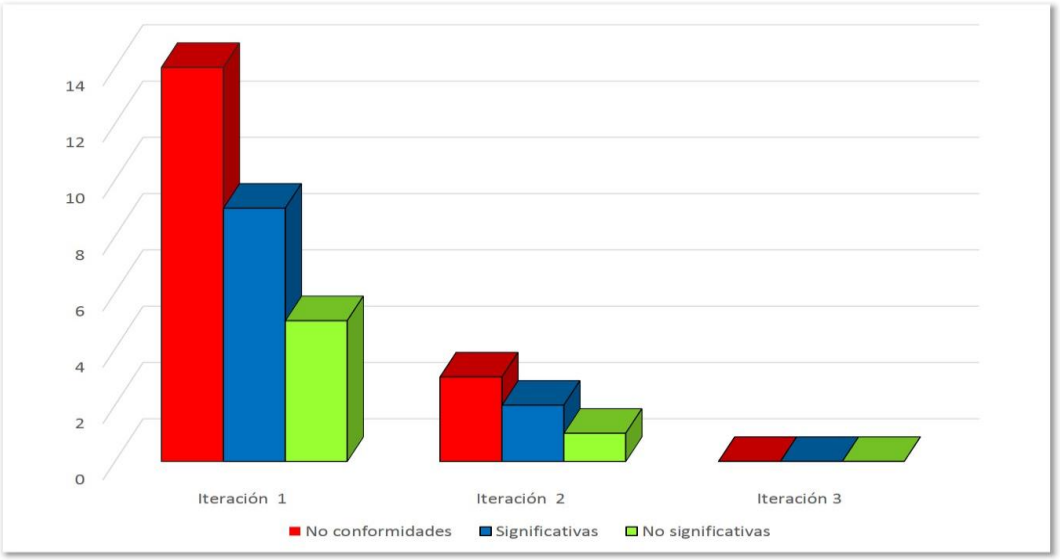


Fig. 19 Resultados de la prueba funcional.

Ver las no conformidades en el expediente de proyecto de OpenUp en la carpeta implementación y prueba documento llamado No Conformidades v2.0.

Prueba de aceptación

La prueba de aceptación fue realizada por parte del cliente, con el fin de validar que la extensión desarrollada realice las funciones para las que ha sido creada, en base a los requerimientos planteados por el usuario. Dicha prueba fue ejecutada por un especialista del área de inteligencia de negocio, en compañía de desarrolladores. La extensión fue probada en una estación de trabajo con las siguientes

prestaciones; pc de 1 GB de RAM con sistema operativo Linux, en esta ambiente de trabajo se pudieron probar todas las funcionalidades del sistema. Se realizaron dos iteraciones; en una primera iteración se detectaron tres no conformidades significativas, de las cuales una con impacto alto referente a la actualización de los cubos virtuales y las restantes con impacto bajo, referente a errores de concordancia entre los CU y la extensión.

En respuesta a las no conformidades detectadas, se trabajó en la identificación de la dificultad y su ajuste o solución, según el caso correspondiente. Las no conformidades detectadas fueron solucionadas correctamente, verificándose en una segunda iteración, donde se constata que el producto está listo para su utilización.

Ver el acta de aceptación en el expediente de proyecto de OpenUp en la carpeta implementación y prueba, en formato jpg llamado Acta de Aceptación v2.0.

3.4 Conclusiones del capítulo

En el presente capítulo se realizó diagrama de componente de la extensión, con el propósito de mostrar una mejor organización y las relaciones que se establecen en el mismo. Se definen los estándares de codificación para ser aplicados en la implementación basada en la versión 1.0 de la extensión. Como resultado de la implementación se logró realizar todas las funcionalidades de la extensión y fueron probadas mediante las pruebas funcionales y de aceptación suprimiendo las no conformidades detectadas en cada iteración.

Conclusiones generales

La investigación cumplió con el objetivo general del trabajo, en la que se desarrolló la versión 2.0 de la extensión para modelar soluciones de AD, que permite la virtualización de los cubos y los miembros calculables en el modelo de datos dimensional, para ello:

Se identificaron los principales elementos a tener en cuenta para el desarrollo y modelado de los hechos, las dimensiones y las medidas virtuales, además del trabajo con datos derivados y la jerarquía de usuario a partir del análisis de las estructuras dimensionales.

La metodología que se adecua a las características del proyecto y al equipo de desarrollo es OpenUp por lo que se escoge como metodología de desarrollo de software. Se emplea como herramienta CASE VP *Community Edition* y NetBeans IDE en su versión 8.0 respectivamente, el lenguaje de programación utilizado fue Java, ya que es el propuesto por la API de desarrollo de la herramienta VP.

La elaboración del modelo de dominio permitió un mejor entendimiento de la extensión, para así delimitar los elementos presentes y los que fueron implementados, con el objetivo de solucionar el problema de la investigación planteado.

Para definir las características y condiciones que debe proveer la solución se identificaron cuarenta y ocho RF y seis RnF, se elaboraron doce CU en los cuales se agrupan los RF que permitieron una correcta planificación en aras de satisfacer las necesidades del cliente.

En el proceso de construcción de la solución se le dio cumplimiento a un conjunto de tareas de programación que permitieron el correcto desarrollo de la extensión, además se le realizaron las pruebas funcionales y de aceptación al producto, lográndose probar la calidad de las funcionalidades del mismo, calificando la solución obtenida como satisfactoria.

Con la versión 2.0 de la extensión para modelar soluciones de AD en la herramienta VP, se provee a los especialistas de una solución, donde puedan realizar el modelo lógico y el modelo dimensional. De esta forma reducir el tiempo de desarrollo del diseño de un AD y garantizar un alto nivel en la calidad de los mismos; además de menor utilización de recursos humanos.

Recomendaciones

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

Utilizar el contenido de la investigación como base de referencia para el desarrollo de futuras extensiones para la herramienta CASE VP.

Referencias Bibliográficas

1. **Adolp, Steve, Cockburn, Alistair y Bramble, Paul. 2002.** Patterns for Effective Use Cases. 2002. ISBN 0201721848 .
2. **Bars, Change. 2007.** OMG Unified Modeling Language (OMG UML),Superstructure, V2.1.2. 2007.
3. **Basurto, Cristhian Kirs Herrera. 2007.** Tutoriales en adictos al trabajo/Datawarehouse. [En línea] 2007. http://www.adictosaltrabajo.com.
4. **Beck, Kent. 1999.** Extreme Programing Explained Embrace Change. 1999. ISBN 0201616416 .
5. **Bernabeu, Ing. Ricardo Dario Córdoba. 2009.** Data Warehousing: Investigación y Sistematización de Conceptos – Hefesto: Metodología propia para la Construcción de un Data Warehouse. 2009.
6. **Carreño, Lohanny. 2013.** Tópicos generales de Ingeniería de Software. [En línea] 2013. https://ingsoftwarei2014.wordpress.com/category/rapid-applicationdevelopment-rad-entorno-integrado-de-desarrollo-ide-ingenieria-de-software-asistida-por-computador-case/.
7. Configuración de la metodología Open UP. **Balduino, Ricardo. 2007.** 2007.
8. **Delgado, Ing. Cecilia Hinojosa y Ing. Ramiro. 2010.** Aplicación de la metodología OpenUp. 2010.
9. **Díaz, Josep Curto. 2012.** Introducción al Business Intelligence. s.l. : UOC, 2012.
10. **Erazo, Miguel. 2013.** PREZI. [En línea] 2013. https://prezi.com/0_tqriq_lh_p/untitled-prezi/.
11. **Espinosa, Itziar Angoitia. 2010.** DATA WAREHOUSE PARA LA GESTIÓN DE LISTA DE ESPERA SANITARIA. Universidad Politécnica de Madrid. 2010.
12. **Espinosa, Roberto. 2010.** Pentaho Shema Worbench. 2010.
13. **Fallas, Ing.Fernely Artavia. 2009.** www.Redestudiantil.com... Su mejor opción en internet. [En línea] 2009. http://www.redestudiantil.com...su/.
14. **Fernández, Oscar Belmonte. 2011.** Introducción al lenguaje de programación Java v1.0. 2011.
15. **Flores, Alberto Martínez Fernández y Israel Ortiz. 2005.** webDevStudio. [En línea] 2005. http://laurel.datsi.fi.upm.es/~ssoo/DAW/Trabajos/2004-2005/Febrero/19/index_es.
16. **Gallardo, Erith Eduardo Pérez. 2012.** Data Warehouse, Modelo, Conceptos e Implementación orientada a SQL Server. [En línea] 2012. http://www.monografias.com/trabajos57/data-warehouse-sql/data-warehouse-sql2.shtm.

17. **García, Jesús, y otros. 2011.** DESARROLLO DE SOFTWARE DIRIGIDO POR MODELOS: CONCEPTOS, MÉTODOS Y HERRAMIENTAS. 2011. ISBN 978-84-9964-215-4.

18. **Gomez, Andrés. 2014.** PREZI. Metodología Open-UP. [En línea] 2014. https://prezi.com/c5pgkh5cmfa4/metodologia-open-up/.

19. **Hofmeister, y otros. 2000.** Applied Software Architecture. [ed.] Addison Wesley. 2000.

20. **IBM. 2013.** Creación de modelos dimensionales lógicos y físicos. [En línea] 2013. http://pic.dhe.ibm.com/infocenter/idm/docv3/index.jsp?topic=%2Fcom.ibm.datatools.dimensional.ui.doc%2Ftopics%2Ft_ldm_pdm_dim_not_wizard.html.

21. **Inmon, William H. 2008 .** DW 2.0 - Architecture for the Next Generation of Data Warehousing. With Derek Strauss and Genia Neushloss, Elsevier Press. 2008 .

22. **Kimball, Margy Ross y Ralph. 2002.** The Data Warehouse Toolkit. 2002.

23. **Larman, Craig. 2005.** Applying UML and Patterns. An Introduction to Object-Oriented. Analysis and Design and Iterative Development. [ed.] Prentice-Hall. 2005. Vol. Tercera Edición .

24. **Larman, Craig . 2004.** UML y Patrones. 2004. Vol. segunda edición ISBN 8420534382 ISBN-13.

25. **León, Gerardo. 2012.** PREZI. [En línea] 2012. https://prezi.com/inxanorqezd-/data-warehouse/.

26. **López, Pedro. 2012.** Inparatodos. [En línea] 2012. http://inparatodos.blogspot.com/2010/09/que-es-un-modelo-dimensional.html.

27. **Mariño, Ediel Enrique García y Alejandro. 2014.** EXTENSIÓN PARA MODELAR SOLUCIONES DE ALMACENES DE DATOS EN LA HERRAMIENTA DE MODELADO VISUAL PARADIGM FOR UML. Centro de Tecnologías de Gestión de Datos Universidad de las Ciencias Informáticas . 2014.

28. **Medina, Cristian Cortez y Yoshvani. 2014.** Diccionario de la lengua española (DRAE). 23. 2014. ISBN 9788467041897.

29. **Morán, Edgar. 2013.** Arquitectura de Software. 2013.

30. **Paradigm, Visual. 2010.** UML and Business Process modeling tool for software development project - Visual Paradigm for UML. 2010.

31. **Penagos, Diblain Romero. 2014.** Senadsi2014. 2014.

32. **Pinciroli, Gustavo J. Sabio y Fernando. 2010.** ¿Cómo describir Casos de uso? 2010.

33. **Pressman, Roger. 2009 .** Ingeniería de Software: Un enfoque práctico. 2009 . págs. 226 - 240. Vol. Sesta Edición .

34. **Reyna, Rafael. 2013.** Scribd Capítulo I Herramientas CASE. [En línea] 2013. http://es.scribd.com/doc/3062020/Capitulo-I-HERRAMIENTAS-CASE..

35. **Ruiz, Javier Heredia, Almanza, Lilián Álvarez y Pons, Naryana Linares. 2011.** Comparación y tendencias entre metodologías ágiles y formales. Metodología utilizada en el Centro de Informatización para la Gestión de Entidades. 2011.

36. **Salter, David. 2014.** NETBeans IDE 8 Cookbook. [ed.] Packt Publishing. 2014. ISBN 1782167765 / 9781782167761.

37. **Sinexus. 2012.** [En línea] 2012. http://www.sinnexus.com/business_intelligence/olap_avanzado.aspx.

38. **Sommerville, Ian. 2006.** Ingeniería de Sofware. 2006. Vol. Séptima edición ISBN 84-7829-074-5.

39. **Studio, Visual. 2003.** Microsoft Developer Network. [ed.] Microsoft. 2003.

40. **Urquizu, Pau. 2012.** [En línea] 2012. http://www.businessintelligence.info/definiciones/que-es-modelo-dimensional.html.

41. **Vásquez, Alejandra. 2014.** PREZI. PRINCIPALES LENGUAJES DE PROGRAMACIÓN. [En línea] 2014. https://prezi.com/cuhbwkm4vjby/principales-lenguajes-de-programacion/.

42. **Villanueva, Wladimiro Díaz. 2012.** Data Warehouses. Valencia : s.n., 2012.

Bibliografía

1. **Adolp, Steve, Cockburn, Alistair y Bramble, Paul. 2002.** Patterns for Effective Use Cases. 2002. ISBN 0201721848.
2. **Balduino, Ricardo. 2007.** Configuración de la metodología Open UP. 2007.
3. **Bars, Change. 2007.** OMG Unified Modeling Language (OMG UML),Superstructure, V2.1.2. 2007.
4. **Basurto, Cristhian Kirs Herrera. 2007.** Tutoriales en adictos al trabajo/Datawarehouse. [En línea] 2007. http://www.adictosaltrabajo.com.
5. **Beck, Kent. 1999.** Extreme Programing Explained Embrace Change. 1999. ISBN 0201616416.
6. **Bernabeu, Ing. Ricardo Dario Córdoba. 2009.** Data Warehousing: Investigación y Sistematización de Conceptos – Hefesto: Metodología propia para la Construcción de un Data Warehouse. 2009.
7. **Bahit, Eugenia. 2002.** El paradigma de la Programación Orientada a Objetos en PHP y el patrón de arquitectura de Software MVC. s.l. : Creative Commons Atribución-NoComercial-SinDerivadas 3.0 Unported, 2002.
8. **Beck, Kent. 2009.** Planning Extreme Programming. 2009. ISBN 0-201-7109-9..
9. **Beltrá, Carlos Patricio López. 2007.** Análisis, Diseño e implementación de un Data Mart para la Dirección Financiera y Recursos Humanos de la Escuela Politécnica del Ejército para una toma de decisión efectiva. 2007.
10. **Carreño, Lohanny. 2013.** Tópicos generales de Ingeniería de Software. [En línea] 2013. https://ingsoftwarei2014.wordpress.com/category/rapid-applicationdevelopment-rad-entorno-integrado-de-desarrollo-ide-ingenieria-de-software-asistida-por-computador-case/.
11. **Canós, José H. 2010.** Metodologías Ágiles en el Desarrollo de Software. valencia : Universidad Politécnica de Valencia, 2010.
12. **Carreira, Mercedes Ruiz. 2008.** Estimación del Coste de la Calidad del Software a través de la Simulación de Procesos de Desarrollo. 2008.
13. **Decsai. 2013.** Modelado de datos. Fundamentos de diseño de bases de datos. [En línea] Departamento de Ciencias de la Computación. Universidad de Granada, 2013. [Citado el: 10 de Diciembre de 2013.] http://elvex.ugr.es/idbis/db/docs/intro/C%20Modelado%20de%20datos.pdf.

14. **Delgado, Ing. Cecilia Hinojosa y Ing. Ramiro. 2010.** Aplicación de la metodología OpenUp. 2010.

15. **Erazo, Miguel. 2013.** PREZI. [En línea] 2013. https://prezi.com/0_tqriq_lh_p/untitled-prezi/.

16. **Espinosa, Itziar Angoitia. 2010.** DATA WAREHOUSE PARA LA GESTIÓN DE LISTA DE ESPERA SANITARIA. Universidad Politécnica de Madrid. 2010.

17. **Espinosa, Roberto. 2010.** Pentaho Shema Worbench. 2010.

18. **Escribano, Gerardo Fernández. 2002.** Introducción a Extreme Programming. 2002.

19. **Espinosa, Roberto. 2010.** Pentaho Shema Worbench. [En línea] 2010. [Citado el: 29 de Octubre de 2013.] http://churriwifi.wordpress.com/2010/07/04/17-3-preparando-el-análisis-dimensional-definición-de-cubos-utilizando-schema-workbench.

20. **Fallas, Ing.Fernely Artavia. 2009.** www.Redestudiantil.com... Su mejor opción en internet. [En línea] 2009. http://www.redestudiantil.com...su/.

21. **Fernández, Oscar Belmonte. 2011.** Introducción al lenguaje de programación Java v1.0. 2011.

22. **Flores, Alberto Martínez Fernández y Israel Ortiz. 2005.** webDevStudio. [En línea] 2005. http://laurel.datsi.fi.upm.es/~ssoo/DAW/Trabajos/2004-2005/Febrero/19/index_es.

23. **Fernández, Carlos. 2013.** Dataprix. OLAP, MOLAP y ROLAP. [En línea] 2013. [Citado el: 1 de Diciembre de 2013.] http://www.dataprix.com/olap-rolap-molap.

24. **Flower. 2010.** Java Foundations. [En línea] 2 de Julio de 2010. [Citado el: 9 de Abril de 2014.] http://javafoundations.blogspot.com/2010/07/java-estandares-de-programacion.html.

25. **Gallardo, Erith Eduardo Pérez. 2012.** Data Warehouse, Modelo, Conceptos e Implementación orientada a SQL Server. [En línea] 2012. http://www.monografias.com/trabajos57/data-warehouse-sql/data-warehouse-sql2.shtm.

26. **García, Jesús, y otros. 2011.** DESARROLLO DE SOFTWARE DIRIGIDO POR MODELOS: CONCEPTOS, MÉTODOS Y HERRAMIENTAS. 2011. ISBN 978-84-9964-215-4.

27. **Genbetadev. 2013.** [En línea] 2013. http://www.genbetadev.com/bases-de-datos/fundamento-de-las-bases-de-datos-modelo-entidad-relacion.

28. **Gomez, Andrés. 2014.** PREZI. Metodología Open-UP. [En línea] 2014. https://prezi.com/c5pgkh5cmfa4/metodologia-open-up/.

29. **Gallardo, Erith Eduardo Pérez. 2012.** Data Warehouse, Modelo, Conceptos e Implementación orientada a SQL Server. [En línea] 2012. [Citado el: 15 de Noviembre de 2013.] http://www.monografias.com/trabajos57/data-warehouse-sql/data-warehouse-sql2.shtml.

30. **Gamma, Erick. 2008.** Patrones de Diseño: elementos de software orientados a objetos reutilizables. Traducido por: Acebal, C. F. Addison Wesley, 313 p. Traducido de: Design Patterns: Elements of Reusable Object-Oriented Software. 2008.

31. **Garlan, M SHAW - D. 1996.** Software Architecture Perspective on an Emerging Discipline. New Jersey : Prentice Hall, 1996.

32. **Grosso, Andrés. 2011.** Prácticas de Software: Patrones Grasp. [En línea] 2011. [Citado el: 17 de Marzo de 2014.] http://www.practicasdesoftware.com.ar/2011/03/patrones-grasp/.

33. **Hofmeister, y otros. 2000.** Applied Software Architecture. [ed.] Addison Wesley. 2000.

34. **IBM. 2013.** Creación de modelos dimensionales lógicos y físicos. [En línea] 2013. http://pic.dhe.ibm.com/infocenter/idm/docv3/index.jsp?topic=%2Fcom.ibm.datatools.dimensional.ui.doc%2Ftopics%2Ft_ldm_pdm_dim_not_wizard.html.

35. **Inmon, William H. 2008.** DW 2.0 - Architecture for the Next Generation of Data Warehousing. With Derek Strauss and Genia Neushloss, Elsevier Press. 2008.

36. **Kimball, Margy Ross y Ralph. 2002.** The Data Warehouse Toolkit. 2002.

37. **Larman, Craig. 2005.** Applying UML and Patterns. An Introduction to Object-Oriented. Analysis and Design and Iterative Development. [ed.] Prentice-Hall. 2005. Vol. Tercera Edición.

38. **Larman, Craig. 2004.** UML y Patrones. 2004. Vol. segunda edición, ISBN 8420534382 ISBN-13.

39. **León, Gerardo. 2012.** PREZI. [En línea] 2012. https://prezi.com/inxanorqezd-/data-warehouse/.

40. **López, Pedro. 2012.** Inparatodos. [En línea] 2012. http://inparatodos.blogspot.com/2010/09/que-es-un-modelo-dimensional.html.

41. **Mariño, Ediel Enrique García y Alejandro. 2014.** EXTENSIÓN PARA MODELAR SOLUCIONES DE ALMACENES DE DATOS EN LA HERRAMIENTA DE MODELADO VISUAL PARADIGM FOR UML. Centro de Tecnologías de Gestión de Datos, Universidad de las Ciencias Informáticas. 2014.

42. **Medina, Cristian Cortez y Yoshvani. 2014.** Diccionario de la lengua española (DRAE). 23. 2014. ISBN 9788467041897.

43. **Morán, Edgar. 2013.** Arquitectura de Software. 2013.
44. **Paradigm, Visual. 2010.** UML and Business Process modeling tool for software development project - Visual Paradigm for UML. 2010.
45. **Penagos, Diblain Romero. 2014.** Senadsi2014. 2014.
46. **Pinciroli, Gustavo J. Sabio y Fernando. 2010.** ¿Cómo describir Casos de uso? 2010.
47. **Pressman, Roger. 2009.** Ingeniería de Software: Un enfoque práctico. 2009. págs. 226 - 240. Vol. Sesta Edición.
48. **Reyna, Rafael. 2013.** Scribd Capítulo I Herramientas CASE. [En línea] 2013. http://es.scribd.com/doc/3062020/Capitulo-I-HERRAMIENTAS-CASE..
49. **Ricardo, Yaneisy Pedraza González y Edgar Rojas. 2012.** Almacén de datos sala situacional. 2012.
50. **Ruiz, Javier Heredia, Almanza, Lilián Álvarez y Pons, Naryana Linares. 2011.** Comparación y tendencias entre metodologías ágiles y formales. Metodología utilizada en el Centro de Informatización para la Gestión de Entidades. 2011.
51. **Salter, David. 2014.** NETBeans IDE 8 Cookbook. [ed.] Packt Publishing. 2014. ISBN 1782167765 / 9781782167761.
52. **Sinexus. 2012.** [En línea] 2012. http://www.sinnexus.com/business_intelligence/olap_avanzado.aspx.
53. **Sommerville, Ian. 2006.** Ingeniería de Sofware. 2006. Vol. Séptima edición, ISBN 84-7829-074-5.
54. **Studio, Visual. 2003.** Microsoft Developer Network. [ed.] Microsoft. 2003.
55. **Urquizu, Pau. 2012.** [En línea] 2012. http://www.businessintelligence.info/definiciones/que-es-modelo-dimensional.html.
56. **Vásquez, Alejandra. 2014.** PREZI. PRINCIPALES LENGUAJES DE PROGRAMACIÓN. [En línea] 2014. https://prezi.com/cuhbwkm4vjby/principales-lenguajes-de-programacion/.