Universidad de las Ciencias Informáticas

Facultad 6

Título: Plugin para generar esquemas dimensionales a partir del diagrama Entidad-Relación desde el Visual Paradigm.

Trabajo de diploma para optar por el título de

Ingeniero en Ciencias Informáticas

Autores: Antonio Barreto Sánchez

Julio Fuentes Gallardo

Tutores: Ing. Yamila Mateu Romero

Ing. Roberto Tellez Ibarra

La Habana, Julio de 2016

"Año 58 de la Revolución"

"Sólo podemos ver poco del futuro, pero lo suficiente para darnos cuenta de que hay mucho que hacer".

Alan Turing

Declaración de autoría

Declaramos ser los autores de la presente tesis Plugin para el Visual Paradigm que genere esquemas dimensionales a partir del diagrama Entidad-Relación 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 _____.

Antonio Barreto Sánchez

_____

Firma del autor

Julio Fuentes Gallardo

_____

Firma del autor

Ing. Yamila Mateu Romero

_____

Firma del tutor

Ing. Roberto Téllez Ibarra

_____

Firma del tutor

Datos del Contacto

Autores:

Antonio Barreto Sánchez

Universidad de las Ciencias Informáticas (UCI)

La Habana, Cuba

e-mail: abarreto@estudiantes.uci.cu

Julio Fuentes Gallardo

Universidad de las Ciencias Informáticas (UCI)

La Habana, Cuba

e-mail: jfgallardo@estudiantes.uci.cu

Tutores:

Ing. Yamila Mateu Romero

Universidad de las Ciencias Informáticas (UCI)

La Habana, Cuba

e-mail: ymateu@uci.cu

Ing. Roberto Tellez Ibarra

Universidad de las Ciencias Informáticas (UCI)

La Habana, Cuba

e-mail: rtibarra@uci.cu

Agradecimientos

De Julio:

Por encima de todo y de todos quisiera agradecer a mi madre por ser la mujer más luchadora que conozco, por el amor y la confianza que no me faltaron nunca, por creer y estar pendiente de mí. Por ser mi razón de ser. Te quiero mucho.

A mi abuela y a sus consejos, gracias por ser mi soporte en tiempos difíciles.

A mi familia por estar siempre al tanto de lo que sucede en mi vida y por estar pendientes de mi carrera universitaria y darme ánimos para seguir adelante.

A mi abuelo Guido, mis tíos Osmín y Orlando, a mi pequeña hermana.

Al dúo de las Yanelis, Benítez y Hernández, a Manolo, Irina, Arodys e Irela quienes sin su ayuda hoy no hubiese cumplido mi sueño, a Irina en especial por hacer de los momentos más amargos los más divertidos.

Personas importantes en el paso en esta escuela como Migdalia, la mejor instructora que puede existir, Maribel, que decir de ella, sin palabras.

A todas las amistades que he ganado en esta escuela, muchas de las cuales tal vez no volvamos a vernos por la distancia que nos separa, pero que siempre las recordaré, que de una forma u otra me han ayudado a formar parte de sus vidas y que de cierta forma he compartido muy buenos momentos y he tratado de ser igual para ellos, por tener el valor de rectificarme, por aconsejarme y apoyarme siempre que lo he necesitado. Especialmente a Yanara, por estar siempre cuando la necesité y preocuparse por mí. Roberto y Yaisel, las primeras personas en brindarme su amistad. Manano por estar en las buenas y las malas, Lázaro por ser mi consejero personal, el Flako y Taire

por ser incondicionales a mi persona. A todos, gracias por estar junto a mí en los buenos y malos momentos que he pasado, gracias por darme ánimos para seguir adelante y por no perder la calma cuando yo si lo hacía. Espero haber sido para ustedes un gran amigo también.

De Antonio:

Traer una nueva vida al mundo es el gesto más noble que una persona pueda realizar, ello implica un esfuerzo y una responsabilidad tan grande que no todo humano es capaz de soportar, por tal motivo agradezco de corazón a mis padres Antonio y Amalia, por darme la posibilidad de ver la luz de una existencia llena de esfuerzos y que, sin su ayuda, su paciencia, amor y dedicación no hubiese logrado.

A mi madre, ¡Gracias! Usted no se imagina, cuánto significa para mi cada una de las acciones, que a veces sin poder, realizó para que yo pudiera estar aquí, cinco años después de llegar a un mundo totalmente desconocido, hoy, con orgullo puedo mirarla y decirle que ya su hijo es ingeniero.

A mi tía Idalmis por ser la persona más fuerte que conozco, a Manuel y mi hermana Yerenys que, a pesar de la lejanía, en estos años, me hizo comprender lo que es tener una hermana.

La UCI ha sido una experiencia de esas que te marcan tan profundamente, que, aunque tengas muchos años jamás la olvidarás, porque es donde tienes la oportunidad de conocer a personas muy especiales. Personas que en el futuro te hacen agradecerle a Dios una y otra vez por haberlas puesto en tu camino. ¡Gracias a mis amigos por estar ahí! No puedo dejar de mencionar aquí a personas que han sido muy especiales para mí: Yadira, Yanet, Arian y Lázaro, tratando siempre de mantenernos unidos.

A mi novia Sandra, por aguantarme desde que nos conocimos y estar a mi lado en los momentos más difíciles, por ser mi motor impulsor y mi bastón cuando pensaba que todo estaba perdido.

Agradezco a mis tutores Yamila y Roberto por guiarme durante todo el proceso de investigación, por su apoyo y paciencia.

Finalmente, y no menos importante, doy gracias a todos los profesores que aportaron a mi formación como profesional, por ellos hoy Antonio se gradúa de ingeniero.

Dedicatoria

A nuestras madres por su entrega incondicional.

A toda la familia por su apoyo en todo momento.

A todos los profesores que han contribuido con nuestra formación.

Resumen

El uso de almacenes de datos ha alcanzado su mayor auge en las empresas e instituciones, al contener información que apoya la toma de decisiones y la inteligencia de negocios. En el centro DATEC los especialistas de inteligencia de negocio hacen uso de la herramienta Pentaho Schema Workbench para la construcción de los esquemas dimensionales necesarios en la elaboración de un almacén de datos. Esta investigación propone integrar un plugin al Visual Paradigm con el objetivo de permitir la generación automática de un diagrama Entidad-Relación en un esquema dimensional. Para llevar a cabo la misma se utilizó como metodología de desarrollo de software OpenUp; como IDE NetBeans y como herramienta de modelado Visual Paradigm for UML. Una vez concluida la investigación se obtuvo como resultado un plugin que genera automáticamente esquemas dimensionales, presenta un sistema de auto salvado del esquema y presenta opciones para gestionar los elementos del esquema. El mismo fue validado mediante la aplicación de las pruebas de integración, funcionales y de aceptación.

Palabras Claves: *Automática, diagrama Entidad-Relación, esquemas dimensionales, modelado, plugin.*

Abstract

The use of data warehouses has reached its peak in enterprises and institutions, containing information that supports decision-making and business intelligence. In the center DATEC the business intelligence specialists make use of Pentaho Schema Workbench tool for building dimensional diagrams necessary in the development of a data warehouse. This research proposes a plugin to integrate to Visual Paradigm in order to allow the automatic generation of an Entity - Relationship diagram in a dimensional scheme. To carry out this research the OpenUP software development methodology was used; NetBeans IDE and the modeling tool Visual Paradigm for UML were also used. After the investigation, a plugin that automatically generates dimensional schemas, presents an auto-saved scheme system and presents options to manage schema elements was obtained as a result. The plugin was validated by applying integration, functional and acceptance testing.

Keywords: A*utomatic, entity relationship diagram, dimensional drawings, modeling, plugin.*

Índice

Introducción 1
Capítulo 1: Fundamentos teóricos de los modelos dimensionales para soluciones de almacenes de datos. 5
1.1. Modelo Entidad-Relación 5
1.2. Almacenes de datos 5
1.3. Modelos dimensionales 6
1.4. Soluciones existentes para modelar almacenes de datos 10
1.5. Lenguajes de programación 12
1.6. Herramientas y metodología a utilizar 13
1.7. Conclusiones del capítulo 19
Capítulo 2: Análisis y diseño del plugin para generar esquemas dimensionales a partir del modelo Entidad-Relación desde el Visual Paradigm. 20
2.1. Propuesta del Sistema 20
2.2. Modelo de dominio 20
2.3. Requisitos Funcionales 22
2.4. Caso de uso 25
2.4.1. Descripción de los casos de uso del sistema 27
2.5. Requisitos no funcionales 30
2.6. Arquitectura de software propuesta 30
2.6.1. Patrón arquitectónico 31
2.6.2. Diagrama de clases del diseño 32
2.7. Patrones de diseño 34
2.7.1. Patrones GRASP 34
2.7.2. Patrones GoF 38
2.8. Conclusiones del capítulo 40
Capítulo 3: Implementación y pruebas del plugin para la generación de esquemas dimensionales a partir del modelo Entidad-Relación desde el Visual Paradigm. 41
3.1. Diagrama de componentes 41
3.2. Estándares de codificación 45
3.3. Pruebas del sistema 47

3.3.1. Pruebas funcionales 47
3.3.2. Prueba de Integración 52
3.3.3. Pruebas de aceptación 54
3.4. Conclusiones del capítulo 54
Conclusiones generales 55
Recomendaciones 56
Referencias Bibliográficas 57
Bibliografía 59

Índice de figuras

Figura 1: Esquema estrella 8
Figura 2: Esquema copo de nieve. 9
Figura 3: Esquema constelación de hechos. 10
Figura 4: Modelo de dominio. 21
Figura 5: Diagrama de caso de uso del sistema. 26
Figura 6: Diagrama de clase del diseño CU Generar esquema dimensional. 33
Figura 7: Patrón Experto. 34
Figura 8: Alta cohesión. 35
Figura 9: Controlador. 37
Figura 10: Patrón Creador. 38
Figura 11: Patrón Singleton. 39
Figura 12: Patrón Iterator. 39
Figura 13: Diagrama de Componentes CU Generar Esquema Dimensional. 42
Figura 14: Resultados de las pruebas funcionales. 52
Figura 15: Estructura de despliegue del plugin. 53
Figura 16: Estructura de implementación del plugin. 53
Figura 17: Herramienta de despliegue. 54

Índice de tablas

Tabla 1: Comparación entre entornos de desarrollo. 16

Tabla 2: Comparación entre metodologías ágiles. 17

Tabla 3: Requisitos funcionales. 22

Tabla 4: Descripción del CU Generar esquema dimensional. 27

Tabla 5: SC Adicionar cubo. 48

Tabla 6: SC Modificar cubo. 49

Tabla 7: SC Eliminar cubo. 49

Tabla 8: SC Visualizar cubo (Modo edición). 50

Tabla 9: SC Modificar cubo (Modo xml). 50

Tabla 10: Descripción de las variables. 50

Introducción

Las bases de datos desde sus inicios se convirtieron en una herramienta fundamental de control y manejo de las operaciones comerciales. Fue así, que en pocos años las grandes empresas y negocios contaban con un considerable número de información almacenada en diferentes fuentes de datos y estas, ya habían alcanzado un tamaño considerablemente grande. Por tanto, surge la necesidad de unificar las diferentes fuentes de información de las que disponían, sobre la base de una estructura organizada, integrada, lógica, dinámica y de fácil explotación. De esta forma surgen los almacenes de datos.

Según Ralph Kimball un almacén de datos es una copia de las transacciones de datos específicamente estructurada para la consulta y el análisis, además determinó que: es la unión de todos los mercados de datos de una entidad (Kimball 1996). En la actualidad las empresas e instituciones apuestan por esta tecnología para el almacenamiento de grandes volúmenes de información. Ejemplo de esto en Cuba son la Oficina Nacional de Estadística e Información y el Tribunal Supremo Popular.

En la Universidad de las Ciencias Informáticas, centro universitario surgido en el 2002 que tiene como misión la formación de profesionales integrales capaces de producir aplicaciones y brindar servicios informáticos, los estudiantes se vinculan a proyectos productivos reales dentro de los centros de desarrollo de software adjuntos a las facultades, desde los primeros años de la carrera. Entre los centros pertenecientes a la facultad 6 de la universidad, se encuentra el Centro de Tecnologías de Gestión de Datos (DATEC), el cual tiene como objetivo crear bienes y servicios informáticos relacionados con la gestión de datos; área del conocimiento que agrupa tanto a los sistemas de información, como a los denominados sistemas de inteligencia empresarial o de negocios.

Para el desarrollo de los almacenes de datos se hace uso de modelos dimensionales, el cual soporta las operaciones que se realizan sobre los datos. Entre los sistemas gestores de bases de datos que poseen soporte dimensional se encuentran *Oracle*, *IBM DB2* y *Microsoft Analysis Services*. Estas son herramientas con altos costos, lo que dificulta el acceso de Cuba a las mismas, por lo que se necesita recurrir a herramientas externas capaces de convertir los datos de una organización en información útil y, eventualmente, conocimiento. En el caso particular de DATEC la herramienta utilizada es el *Pentaho Schema Workbench*.

La herramienta Schema Workbench permite crear esquemas multidimensionales. El especialista de inteligencia de negocios, para realizar este proceso necesita un diagrama Entidad-Relación modelado en

la herramienta Visual Paradigm, el cual le es entregado previamente. Esta tarea para los desarrolladores es considerada como repetitiva, tediosa y propensa a errores, debido a la propia inexperiencia de los mismos, teniendo en cuenta que se insertan de forma manual los cubos, medidas, dimensiones, jerarquías, niveles[1], tablas, así como los respectivos valores de cada uno de ellos, obteniendo como resultado un esquema dimensional que posee puntos en común con el modelo físico de la base de datos.

A partir de la situación descrita anteriormente se define como **problema de la investigación**: ¿Cómo contribuir al diseño de esquemas dimensionales desde Visual Paradigm que permita facilitar el trabajo a los especialistas de inteligencia de negocios en DATEC?

El problema descrito centra su **objeto de estudio** en los modelos dimensionales para soluciones de almacenes de datos, enmarcado en el **campo de acción:** Los modelos dimensionales para soluciones de almacenes de datos utilizando la herramienta Visual Paradigm.

Con la finalidad de darle cumplimiento al problema de la investigación se precisa el siguiente **objetivo general:** Desarrollar un plugin para el Visual Paradigm que genere automáticamente esquemas dimensionales a partir del diagrama Entidad-Relación que permita contribuir a facilitar el trabajo a los especialistas de inteligencia de negocios en DATEC.

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

- ¿Cuáles son las bases teóricas que fundamentan el diseño del modelo dimensional?
- ¿Cuáles son las características y capacidades que debe tener el plugin para generar esquemas dimensionales a partir del diagrama Entidad-Relación desde el Visual Paradigm?
- ¿Cómo lograr el desarrollo del Plugin para generar esquemas dimensionales a partir del diagrama Entidad-Relación desde el Visual Paradigm?
- ¿Cómo validar el correcto funcionamiento del Plugin para generar esquemas dimensionales a partir del diagrama Entidad-Relación desde el Visual Paradigm?

Para darle cumplimiento al objetivo general planteado y responder a las preguntas científicas se diseñan las siguientes **tareas de la investigación**:

[1] Nivel de detalle al cual se identifican los componentes en una estructura de una base de datos.

1. Elaboración del marco teórico de la investigación para la generación de esquemas dimensionales desde el diagrama Entidad-Relación para el mejor entendimiento de la problemática que se plantea.
2. Selección de las metodologías, herramientas y tecnologías a utilizar en el desarrollo del plugin para la generación de esquemas dimensionales desde el diagrama Entidad-Relación en la herramienta de modelado Visual Paradigm, para lograr el desarrollo de la misma.
3. Análisis de las funcionalidades a incluir en el plugin para la generación de esquemas dimensionales desde el diagrama Entidad-Relación en la herramienta de modelado Visual Paradigm, para lograr una correcta implementación.
4. Descripción de la estructura base del plugin para la generación de esquemas dimensionales desde el diagrama Entidad-Relación en la herramienta Visual Paradigm, para guiar la construcción del mismo.
5. Implementación del plugin para generar esquemas dimensionales a partir del diagrama Entidad-Relación desde el Visual Paradigm.
6. Realización de pruebas al plugin para generar esquemas dimensionales a partir del diagrama Entidad-Relación desde el Visual Paradigm, para comprobar el correcto funcionamiento del mismo.

Los métodos teóricos que fueron empleados son:

- **Análisis - Síntesis:** Se empleó en la realización de un estudio del estado del arte del modelado de almacenes de datos en la herramienta Visual Paradigm. Además, permitió 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.
- **Inductivo-deductivo:** Se usó al realizar un análisis y observación del tema de la investigación y a partir del mismo deducir o inducir criterios sobre el tema en cuestión.

Los métodos empíricos que fueron empleados son:

- **Análisis documental:** Se usó en la revisión de la documentación oficial del expediente de proyecto del plugin para generar esquemas dimensionales desde el diagrama Entidad-Relación para la herramienta de modelado Visual Paradigm. Permitió obtener el conocimiento necesario para el desarrollo de la herramienta.

- **Entrevista:** Se utilizó la Entrevista para definir los requisitos funcionales y no funcionales con los que debe cumplir la propuesta solución. Realizadas de forma individual a profesionales que están vinculados directamente con el desarrollo y evaluación de proyectos de este tipo.

La investigación está estructurada de la siguiente manera: Resumen, Introducción, Capítulo 1, 2 y 3, Conclusiones, Recomendaciones, Referencias bibliográficas, Bibliografía y Anexos. La temática de cada capítulo será la siguiente: **Capítulo 1**: En el presente capítulo se abordan múltiples elementos teóricos pertenecientes a los almacenes de datos, así como sus principales procesos logrando definir las principales características que debe poseerla solución a implementar, así como la metodología y herramientas para su desarrollo. **Capítulo 2**: En el presente capitulo se realiza el análisis y diseño del plugin para la generación de esquemas dimensionales desde el visual paradigm a partir del correcto levantamiento de sus requisitos. Se establece además el patrón arquitectónico a utilizar, así como la identificación de los patrones de diseño utilizados los cuales se evidencian en los diagramas de clases del diseño elaborados. **Capítulo 3:** Este capítulo comprende la implementación del plugin, mediante la construcción del diagrama de componentes y la definición de los estándares de codificación. Además, se plasman los resultados arrojados por las pruebas realizadas a la propuesta de solución, las cuales fueron realizadas con el objetivo de verificar el cumplimiento de las peticiones hechas por el cliente y garantizar de esta forma la calidad de la propuesta de solución.

Capítulo 1: Fundamentos teóricos de los modelos dimensionales para soluciones de almacenes de datos.

En el presente capítulo se abordan múltiples elementos teóricos pertenecientes a los almacenes de datos, así como sus principales procesos logrando definir las principales características que debe poseerla solución a implementar, así como la metodología y herramientas para su desarrollo.

1.1. Modelo Entidad-Relación

El modelo de Entidad-Relación es un modelo de datos basado en una percepción del mundo real que consiste en un conjunto de objetos básicos llamados entidades y relaciones entre estos objetos, implementándose en forma gráfica a través del diagrama Entidad-Relación. A continuación, sus principales elementos (Ochando 2014):

- ✓ *Entidad:* La entidad es cualquier clase de objeto o conjunto de elementos presentes o no, en un contexto determinado dado por el sistema de información o las funciones y procesos que se definen en un plan de automatización. Dicho de otra forma, las entidades las constituyen las tablas de la base de datos que permiten el almacenamiento de los ejemplares o registros del sistema, quedando recogidos bajo la denominación o título de la tabla o entidad.
- ✓ *Atributos – Intención*: Son las características, rasgos y propiedades de una entidad, que toman como valor una instancia particular. Es decir, los atributos de una tabla son en realidad sus campos descriptivos, el predicado que permite definir lo que decimos de un determinado sujeto.
- ✓ *Relación:* Vínculo que permite definir una dependencia entre los conjuntos de dos o más entidades. Esto es la relación entre la información contenida en los registros de varias tablas.
- ✓ *Clave principal primaria:* Permiten identificar unívocamente cada registro de una tabla.
- ✓ *Clave externa:* Campo clave conformado por el valor de una clave principal primaria de otra tabla.

1.2. Almacenes de datos

El correcto modelado de un diagrama Entidad-Relación implica que sea sencillo modelar los almacenes de datos. Estos son un conjunto de datos orientados a temas, integrados, no volátil, variables en el tiempo, como soporte para la toma de decisiones (Kimball 1996). Los almacenes de datos contienen a menudo grandes cantidades de información que se subdividen a veces en unidades lógicas más pequeñas (mercados de datos) dependiendo del subsistema de la entidad del que procedan o para el que sea

necesario. Una vez reunidos los datos de los sistemas fuentes se guardan durante mucho tiempo, lo que permite el acceso a datos históricos; así los almacenes de datos proporcionan al usuario una interfaz consolidada única para los datos, lo que hace más fácil escribir las consultas para la toma de decisiones (Velasco 2014).

Ventajas

A continuación, se refieren algunas ventajas que presenta la implementación de un almacén de datos (Bernabeu 2007):

- ✓ Proporciona una herramienta para el proceso de toma de decisiones estratégicas y tácticas en cualquier área funcional, teniendo en cuenta información integrada y global del negocio.
- ✓ Simplifica la aplicación de técnicas estadísticas de análisis y modelación para encontrar relaciones ocultas entre los datos del almacén; obteniendo metadatos como valor añadido para el negocio sobre dicha información.
- ✓ Aporta la capacidad de aprender de los datos del pasado y de predecir situaciones futuras en diversos escenarios.

Desventajas

Algunas de las desventajas de los almacenes de datos son (Dario 2009):

- ✓ Requiere una gran inversión, debido a que su correcta construcción no es tarea sencilla y consume muchos recursos, además, su misma implementación implica desde la adquisición de herramientas de consulta y análisis, hasta la capacitación de los usuarios.
- ✓ Si se incluyen datos propios y confidenciales de los clientes, el depósito de datos atentará contra la privacidad de los mismos, ya que cualquier usuario podrá tener acceso a ellos.
- ✓ Infravaloración de los recursos necesarios para la captura, carga y almacenamiento de los datos.

1.3. Modelos dimensionales

Un modelo dimensional es un diseño lógico para construir almacenes de datos. Difiere del diagrama Entidad-Relación ya que estos se orientan a transacciones. Kimball define este modelo como la única

forma viable para llevar datos a usuarios en un almacén de datos. Cada dato de negocio puede representarse en un cubo con dos o más dimensiones (Kimball 1996).

El uso del modelo dimensional es una de las aproximaciones más acertadas y seguidas por los especialistas en estos días. Este se basa en el estudio de los eventos del negocio analizados desde sus distintas dimensiones. En la actualidad, las tecnologías de la información han automatizado los procesos de carácter típicamente repetitivo o administrativo, haciendo uso de lo que se denomina sistemas de información operacionales. Dichos sistemas resuelven las necesidades de funcionamiento de las empresas, donde sus principales características son la actualización y el tiempo de respuesta (Rondón, Domínguez y Berenguer 2011).

Las necesidades informacionales (necesidades de funcionamiento de la empresa), son aquellas que tienen por objeto obtener la información necesaria, que sirva de base para la toma de decisiones tanto a escala estratégica como táctica. Estas necesidades se basan en gran medida en el análisis de un alto volumen de datos, en el que es tan importante el obtener un valor muy detallado de negocio como el valor totalizado para el mismo. También, es fundamental la visión histórica de todas las variables analizadas y el análisis de los datos del entorno.

A nivel mundial, disímiles son las aplicaciones desarrolladas teniendo como base los modelos dimensionales. Ejemplo de esto son las aplicaciones para el desarrollo de nuevas tecnologías para el uso de la Información Geográfica (Rondón, Domínguez y Berenguer 2011), las usadas como mecanismo de mejora para la gestión de proyectos de construcción (Izaguirre y Alarcón 2008), entre otras. En Cuba su principal función es la de mejorar la rapidez de acceso a grandes volúmenes de datos en las aplicaciones para la gestión y almacenamiento de la información, contribuyendo así, al aumento considerable de la disponibilidad de dicha información, así como la creación de los modelos analíticos para luego proceder con los cálculos y análisis estadísticos.

Tablas de dimensiones

Son las tablas que almacenan información detallada acerca de los hechos. Cada tabla de dimensión contiene varias columnas y atributos que se utilizan para describir los procesos del negocio.

Cada tabla de dimensión podrá contener los siguientes campos (Dario 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.

Estas tablas según las técnicas de modelado utilizadas pueden adoptar forma de estrella, copo de nieve o constelación de hechos. La presente investigación propone el reconocimiento de esquemas estrella y constelación de hechos.

Esquema en estrella

Este esquema consta de una tabla de hechos central que se relaciona con varias tablas de dimensiones- Esto favorece a una mayor rapidez de las consultas, aunque tiene la desventaja de generar cierto grado de redundancia.

Entre sus ventajas más significativas se encuentran su simpleza de interpretación y la optimización de los tiempos de respuestas ante las consultas de los usuarios. Un esquema de estrellas puede tener un gran número de tablas de dimensiones (IBM 2013). En la figura 1 se puede apreciar un ejemplo de esquema estrella.

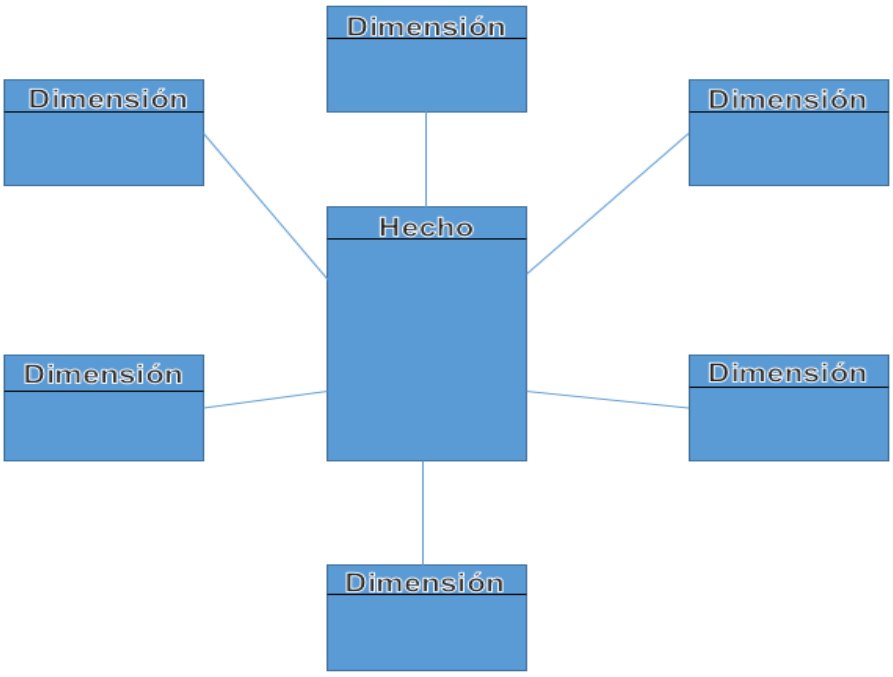

Figura 1: Esquema estrella

Esquema copo de nieve

Constituye una ampliación del esquema en estrella cuando las tablas de dimensiones se organizan en jerarquías de dimensiones[2]. Existe una tabla de hechos central que está relacionada con una o más tablas de dimensiones, quienes 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 difíciles de manipular. Además, su desempeño puede verse reducido si existen muchas uniones y relaciones entre tablas (IBM 2013). En la figura 2 se puede apreciar un ejemplo de esquema copo de nieve.

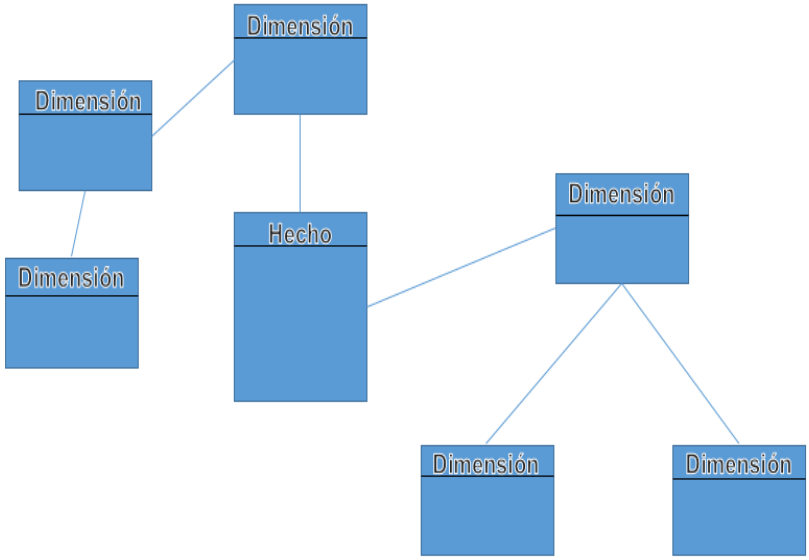

Figura 2: Esquema copo de nieve.

Esquema constelación de hechos

Este esquema puede estar formado por varios modelos en estrella definiéndose más de una tabla de hechos en la parte central del esquema relacionadas por sus respectivas tablas de dimensiones, vinculándose las tablas de hechos auxiliares con algunas dimensiones asignadas a la tabla de hecho

[2] Es cuando una tabla dimensión tiene relación con una o más tablas de tipo dimensión.

principal y también con nuevas tablas de dimensiones (IBM 2013). En la figura 3 se puede apreciar un ejemplo de esquema constelación de hechos.

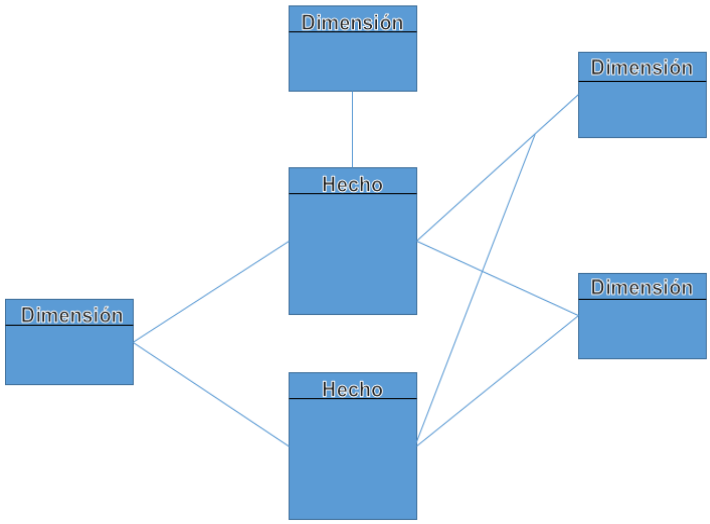

Figura 3: Esquema constelación de hechos.

Tablas de hechos

Son las tablas centrales del modelo dimensional, contienen los valores de las medidas del negocio tomadas mediante la intersección de las dimensiones que la definen y dichas dimensiones estarán reflejadas en sus correspondientes tablas de dimensiones.

Existen dos tipos de hechos, los básicos y los derivados (Bernabeu 2007):

- 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 sencillas y devolver resultados rápidamente, pero requieren más espacio físico en el almacén de datos.

1.4. Soluciones existentes para modelar almacenes de datos

En el mundo existen diversas empresas que se dieron a la tarea de implementar soluciones para modelar almacenes de datos. Las más destacadas actualmente son las ofrecidas por *ORACLE* y *ERWIN*. Estas soluciones ofrecen robustez y calidad en los productos obtenidos; un inconveniente para su empleo en

Cuba es que son distribuidas bajo licencia privativa. *ORACLE* ofrece un paquete completo y totalmente integrado de aplicaciones en la nube y servicios de plataforma (ORACLE 2016). Por otra parte, *ERWIN* se trata de una solución de modelado de datos, líder en la industria que proporciona una interfaz visual y sencilla para administrar su entorno de datos complejos en una empresa (ERwin 2016).

Pentaho BI Suite es un conjunto de programas libres para generar inteligencia empresarial. Incluye herramientas integradas para generar informes, minería de datos, ETL; entre estos programas se encuentra Schema Workbench. Es la herramienta gráfica que permite la construcción de los esquemas OLAP[3] y además permite publicarlos al servidor BI server para que puedan ser utilizados en los análisis por los usuarios de la plataforma. Los esquemas OLAP diseñados, se guardan en formato XML y describen las relaciones entre las dimensiones y medidas del cubo (modelo multidimensional) con las tablas y campos de la base de datos, a nivel relacional. Este mapeo se utiliza para ayudar a la traducción de las consultas MDX (que es el lenguaje con el que trabaja Mondrian), y para transformar los resultados recibidos de las consultas SQL a un formato dimensional (Díaz y Paz 2008). Esta herramienta es usada por los especialistas de inteligencia de negocios de DATEC para la creación de esquemas multidimensionales y servirá como apoyo para la presente investigación.

Con el fin de lograr una independencia tecnológica en la universidad, se han desarrollado algunas soluciones informáticas con el objetivo de modelar dichos almacenes. Entre las investigaciones desarrolladas por el centro DATEC podemos destacar los trabajos de diploma: “Extensión para modelar soluciones de almacenes de datos en la herramienta de modelado *Visual Paradigm”* de Sergio Martin Torres el cual permite diseñar el modelo lógico del almacén, generar el modelo físico y crear la estructura básica de los cubos OLAP; “Extensión para modelar soluciones de almacenes de datos en la herramienta de modelado *Visual Paradigm for UML”* que corrige deficiencias en la tesis descrita anteriormente tales como:

- No permite diagramar correctamente el diseño lógico del almacén de datos.
- Permite que un hecho se relacione con otro hecho.
- No permite la relación de un hecho con una dimensión de uno a uno y de uno a muchos, en cambio genera relacionado con el mismo una entidad lógica.

[3] Los esquemas OLAP son estructuras multidimensionales (cubos) que permiten analizar bases de datos relacionales de gran volumen y variedad.

- La entidad dimensional de tipo puente está conceptualmente mal utilizada pues se relaciona entre un hecho y una dimensión y recibe la llave primaria del hecho.

Y por último “Extensión para modelar soluciones de Almacenes de Datos en la herramienta Visual Paradigm” de Glennis Emilia Díaz del Toro y Jessie Cruz Pérez la cual incorpora nuevas funcionalidades a la extensión ya creada, la virtualización de los cubos, el trabajo con datos derivados y la jerarquía de usuario. Las soluciones vistas anteriormente no abordan la automatización del proceso, sino que definen un nuevo esquema.

Luego de terminado el análisis sobre soluciones existentes se decide tomar características y funcionalidades presentes en la herramienta Schema Workbench para aplicarlas en el desarrollo del plugin para generar esquemas dimensionales a partir del modelo Entidad-Relación. Algunas de estas características son:

- Representación del esquema dimensional generado automáticamente a través de un árbol jerárquico.
- Uso de los íconos que presenta la herramienta Schema Workbench y la forma de la barra de herramientas.
- Forma de visualizar los elementos, árbol jerárquico y en forma de código .xml.

1.5. Lenguajes de programación

Un lenguaje de programación es un lenguaje formal diseñado para realizar procesos que pueden ser llevados a cabo por máquinas como las computadoras. Pueden usarse para crear programas que controlen el comportamiento físico y lógico de una máquina, para expresar algoritmos con precisión, o como modo de comunicación humana (Lutz 2009). Para el desarrollo del plugin se utilizan los lenguajes de programación Java y XML: Java ofrecerá la implementación del plugin en el IDE *Netbeans* y XML hará posible la visualización del esquema dimensional.

Java

Java es un lenguaje de programación y una plataforma informática comercializada por primera vez en 1995 por *Sun Microsystems*. Es un lenguaje de programación con el cual se puede realizar cualquier tipo de aplicación. En la actualidad es un lenguaje muy extendido y cada vez cobra más importancia tanto en el ámbito de *Internet* como en la informática en general (MADEJA 2013).

Una de las principales características por las que *Java* se ha hecho muy famoso es que es un lenguaje independiente de la plataforma. Eso quiere decir que un programa en *Java* podrá funcionar en cualquier ordenador del mercado. Es una ventaja significativa para los desarrolladores de software, pues antes tenían que hacer un programa para cada sistema operativo, por ejemplo, *Windows*, *Linux*, *Apple*, etc. Esto lo consigue porque se ha creado una Máquina de Java para cada sistema que hace de puente entre el sistema operativo y el programa de *Java* y posibilita que este último se entienda perfectamente. La independencia de plataforma es una de las razones por las que Java es interesante para Internet, ya que muchas personas deben tener acceso con ordenadores distintos (Rodríguez 2013).

XML

Lenguaje ampliable de marcas del inglés *Extensible Markup Language* (XML) es un lenguaje de etiquetas, es decir, cada paquete de información está delimitado por dos etiquetas como se hace también en el lenguaje HTML, pero XML separa el contenido de la presentación. XML se plantea como un lenguaje estándar para el intercambio de información entre diferentes programas de una manera segura, fiable y libre, ya que no pertenece a ninguna compañía. Es un meta-lenguaje que nos permite definir lenguajes de marcado adecuados a usos determinados. Representa información estructurada en la web, de modo que esta información pueda ser almacenada, transmitida, procesada, visualizada e impresa, por muy diversos tipos de aplicaciones y dispositivos (Exes 2014).

XML está basado en texto. Es orientado a los contenidos no a la presentación. Las etiquetas se definen para crear los documentos, no tienen un significado preestablecido. No es sustituto de HTML además de no existir un visor genérico de XML. Presenta varias aplicaciones tales como publicar e intercambiar contenidos de bases de datos. Formatos de mensaje para comunicación entre aplicaciones (B2B) y descripción de metacontenidos. (Exes 2014)

1.6. Herramientas y metodología a utilizar

Para desarrollar un sistema fiable, robusto, eficaz y escalable se hace necesario el uso de herramientas y tecnologías manejables, confiables y que faciliten operaciones como el modelado del negocio, diseño, implementación y puesta en marcha del sistema. Para ello se analizaron diferentes herramientas y de este amplio conjunto se escogieron las que, por sus características, son más eficaces de acuerdo a las cualidades de la empresa y del equipo encargado de desarrollar la propuesta de solución. Las mismas son detalladas a continuación.

Visual Paradigm for UML 8.0 Enterprise Edition

Las herramientas CASE [4], las cuales cuentan con una credibilidad y exactitud que tienen un reconocimiento universal, siendo usadas por cualquier desarrollador y/o programador que busca un resultado óptimo y eficiente, actualmente brindan una gran gama de componentes que incluyen todos o la mayoría de los requisitos necesarios para el desarrollo de los sistemas. Han sido creadas con una gran precisión en torno a las necesidades de los desarrolladores de software para la automatización de procesos, incluyendo el análisis, diseño e implantación. Ofrecen una gran plataforma de seguridad a sistemas que las usan. Entre las más usadas se encuentra el Visual Paradigm. La misma puede crear ingeniería inversa del código de diagramas y ofrecer ida y vuelta de ingeniería para diversos lenguajes de programación. Esta herramienta ayuda a los desarrolladores de software a crear un modelo de excelencia durante la creación y distribución del proceso de desarrollo de aplicaciones.

Es un modelador que permite diseñar el sistema con todo tipo de diagramas UML. También es compatible con la gestión intensiva de casos de uso, requerimientos y diseño de base de datos. Con Visual Paradigm, el equipo de desarrollo de software puede realizar análisis y diseño de sistemas con facilidad. Proporciona una suite de productos premiados que facilita a las organizaciones diseñar visualmente y de forma esquemática, integrar y desplegar sus aplicaciones de misión crítica de la empresa y sus bases de datos subyacentes (Rondón, Domínguez y Berenguer 2011)

Entre sus características fundamentales se tiene:

- Multiplataforma: Soportada en plataforma *Java* para Sistemas Operativos *Windows*, *Linux*, *Mac OS*.
- Interoperabilidad: Intercambia diagramas UML y modelos con otras herramientas. Soporta la importación y exportación a formatos XMI, XML y archivos *Excel*. Permite importar proyectos de *Rational Rose* y la integración con *Microsoft Office Visio*.
- Extensible mediante complementos o *plugin*.
- Modelado de requisitos: Captura de requisitos mediante diagramas de requisitos, modelado de caso de uso y análisis textual.

[4] Son diversas aplicaciones informáticas o programas informáticos destinadas a aumentar la productividad en el desarrollo de software reduciendo el costo de las mismas en términos de tiempo y de dinero.

- Ingeniería de Código: Permite la generación de código e ingeniería inversa para los lenguajes: Java, C, C++, PHP, XML, *Python*, C#, VB .Net, *Flash*, *ActionScript*, *Delphi* y *Perl*.
- Integración con entornos de desarrollo: Apoyo al ciclo de vida completo de desarrollo de *software* en IDE como: *Eclipse*, *Microsoft Visual Studio*, *NetBeans*, *Sun ONE*, *Oracle JDeveloper*, *Jbuilder* y otros.
- Modelado de bases de datos: Generación de bases de datos y conversiones de diagramas Entidad-Relación a tablas de bases de datos, además de mapeos de objetos y relaciones.

Plugins

La herramienta Visual Paradigm permite la adición de *plugins*. Un *plugin* es una aplicación que, en un programa informático, añade una funcionalidad adicional o una nueva característica al software. En la actualidad, la mayoría de los programas trabajan con los mismos, debido a que entre las principales ventajas que ofrecen se encuentra que facilitan la colaboración de desarrolladores externos con el software. Estos desarrolladores pueden realizar sus aportes a las funcionalidades a través de distintos *plugins*. (Edukavital 2015).

UML 2.1

Es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema. UML ofrece un estándar para describir un "plano" del sistema (modelo), incluyendo aspectos conceptuales tales como procesos de negocio, funciones del sistema, y aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y compuestos reciclados. Es importante remarcar que UML es un "lenguaje de modelado" para especificar o describir métodos o procesos. Se utiliza para definir un sistema, para detallar los artefactos en el sistema y para documentar y construir. En otras palabras, es el lenguaje en el que está descrito el modelo (OMG 2016).

Se puede aplicar en el desarrollo de software gran variedad de formas para dar soporte a una metodología de desarrollo de software tal como OpenUP, pero no especifica en sí mismo qué metodologías o procesos usar. UML no puede compararse con la programación estructurada, pues UML significa Lenguaje Unificado de Modelado, no es programación, solo se diagrama la realidad de una utilización en un requerimiento. Mientras que, programación estructurada, es una forma de programar como lo es la orientación a objetos, la programación orientada a objetos viene siendo un complemento perfecto de UML,

pero no por eso se toma UML sólo para lenguajes orientados a objetos. UML cuenta con varios tipos de diagramas, los cuales muestran diferentes aspectos de las entidades representadas (OMG 2016).

Entorno de desarrollo integrado para Java

Un entorno de desarrollo integrado (IDE) es un programa que nos permite desarrollar código en un lenguaje y que incorpora habitualmente (Rodríguez 2013):

a) Un espacio para la escritura de código con cierta ayuda interactiva para generar código y para indicar los errores de sintaxis que se cometan por parte del programador.

b) La posibilidad de compilar y ejecutar el código escrito.

c) La posibilidad de organizar los proyectos de programación.

d) Herramientas auxiliares para programadores para detección de errores o análisis de programas (*debuggers*).

Tabla 1: Comparación entre entornos de desarrollo.

	Licencia	Sistema operativo	Diseño gráfico	Dominio
Eclipse	Licencia publica de eclipse	Multiplataforma	Si	No
NetBeans	CDDL	Multiplataforma	Si	Si
JCreator	Privativa	Windows	No	No
JBuilder	Privativa	Multiplataforma	Si	No

Teniendo en cuenta las características analizadas anteriormente se selecciona como entorno de desarrollo el Netbeans 8.0 debido a que la herramienta permite tener facilidades hacia el diseño gráfico, y como elemento más significativo se encuentra que es la de mayor dominio por parte el equipo de desarrollo.

NetBeans

Los proyectos desarrollados con esta herramienta poseen lanzadores (*launchers*) para cada plataforma. La misma hace fuerte hincapié sobre la construcción del *software* de forma modular, al ofrecer implementados los mecanismos de descubrimiento de nuevos módulos (y de actualizaciones de los

existentes) desde repositorios remotos, resolución de dependencias y comunicación entre los mismos (Oracle Corporation 2000).

NetBeans permite el uso de un amplio rango de tecnologías de desarrollo tanto para escritorio, como aplicaciones *web*, o para dispositivos móviles. Da soporte a las tecnologías: *Java*, PHP, *Groovy*, C/C++, *HTML5* entre otras. Además, puede instalarse en varios sistemas operativos: *Windows*, *Linux* y *Mac OS*. Teniendo en cuenta las características antes descritas y por su amplia documentación y tutoriales se selecciona este IDE como plataforma de programación.

Metodología de desarrollo

Una metodología de desarrollo de software se refiere a un framework que es usado para estructurar, planear y controlar el proceso de desarrollo en sistemas de información. A lo largo del tiempo, una gran cantidad de métodos han sido desarrollados diferenciándose por su fortaleza y debilidad. Se clasifican en dos tipos: metodologías pesadas y ágiles (Valdéz 2013).

Las metodologías pesadas son las más tradicionales, se centran en la definición detallada de los procesos y tareas a realizar, herramientas a utilizar, y requiere una extensa documentación, ya que pretende prever todo de antemano. Este tipo de metodología es más eficaz y necesaria, cuanto mayor es el proyecto que se pretende realizar respecto a tiempo y recursos que son necesarios emplear, donde una gran organización es requerida. Las ágiles por otra parte se encargan de valorar al individuo y las iteraciones del equipo más que a las herramientas o los procesos utilizados. Se hace mucho más importante crear un producto de software que funcione que escribir mucha documentación. Además, el cliente está en todo momento colaborando en el proyecto.

Tabla 2: Comparación entre metodologías ágiles.

	Tamaño de los proyectos	*Tamaño de equipo*	*Estilo de desarrollo*	*Mecanismo de abstracción*	*Cultura de negocio*
Open UP	Pequeños y medianos.	Equipos pequeños de tres a seis integrantes	Iterativo e incremental	Orientado a objetos	Colaborativo y cooperativo
Scrum	Pequeños, medianos y	Múltiples equipos menores que 10.	Iterativo y rápido	Orientado a objetos	No especificado

	grandes.				
XP	Pequeños y medianos.	Menos que 10.	Iterativo y rápido	Orientado a objetos	Colaborativo y cooperativo

A partir de la comparación realizada, dado el ambiente de trabajo, teniendo en cuenta la participación del cliente dentro del equipo de desarrollo, así como los conocimientos que presenta el equipo se decide emplear la metodología ágil Open UP.

Open Up

OpenUP fue desarrollada por un conjunto de empresas del sector del software que cedieron su creación a la fundación Eclipse para que lo difundiera. Es una metodología dirigida a la gestión y desarrollo de proyectos de software basados en un desarrollo iterativo, ágil e incremental apropiado para proyectos pequeños y de bajos recursos; y es aplicable a un conjunto amplio de plataformas y aplicaciones de desarrollo.

Es un proceso mínimo y suficiente, lo que significa que solo el contenido fundamental y necesario es incluido. Por lo tanto, no provee lineamientos para todos los elementos que se manejan en un proyecto, pero tiene los componentes básicos que pueden servir de base a procesos específicos. La mayoría de los elementos de OpenUP están declarados para fomentar el intercambio de información entre los equipos de desarrollo y mantener un entendimiento compartido del proyecto, sus objetivos, alcance y avances. (Juarez 2013).

Principios de OpenUP (Eclipse 2014):

- 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 cumple 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.

Ciclo de vida de Open UP (Juarez 2013):

- **Iteración inicial:** Primera de las 4 fases en el proyecto del ciclo de vida. El objetivo de ésta fase es capturar las necesidades de los *stakeholder* en los objetivos del ciclo de vida para el proyecto.
- **Iteración de elaboración:** Es el segundo de las 4 fases del ciclo de vida de OpenUP donde se trata los riesgos significativos para la arquitectura. El propósito de esta fase es establecer la base de la elaboración de la arquitectura del sistema.
- **Iteración de construcción:** Esta fase está enfocada al diseño, implementación y prueba de las funcionalidades para desarrollar un sistema completo. El propósito de esta fase es completar el desarrollo del sistema basado en la arquitectura definida.
- **Iteración de transición:** Es la última fase, cuyo propósito es asegurar que el sistema es entregado a los usuarios, y evalúa la funcionalidad y *performance* del último entregable de la fase de construcción.

1.7. Conclusiones del capítulo

Una vez finalizado el presente capítulo se puede concluir que:

A partir del análisis de los fundamentos teóricos estudiados se precisaron los principales conceptos relacionados con los almacenes de datos, permitiendo definir la necesidad de crear un plugin para la creación de esquemas multidimensionales a partir de los diagramas Entidad-Relación, así como las principales características que debe poseer el mismo. Se decidió utilizar en el desarrollo de la propuesta de solución como metodología de desarrollo de software OpenUP; como entorno de desarrollo NetBeans 8.0; como lenguaje de programación Java 8.0 y como lenguaje de modelado UML en su versión 2.1.

Capítulo 2: Análisis y diseño del plugin para generar esquemas dimensionales a partir del modelo Entidad-Relación desde el Visual Paradigm.

En el presente capítulo se realiza el análisis y diseño del plugin para la generación de esquemas dimensionales desde el visual paradigm a partir del correcto levantamiento de sus requisitos. Se establece además el patrón arquitectónico a utilizar, así como la identificación de los patrones de diseño utilizados los cuales se evidencian en los diagramas de clases del diseño elaborados.

2.1. Propuesta del Sistema

El especialista de inteligencia de negocios hace uso de la herramienta *Pentaho Schema Workbench* para crear un esquema dimensional a partir de un diagrama Entidad-Relación modelado en la herramienta Visual Paradigm, el cual le es entregado previamente. Esta tarea para los desarrolladores se considera un proceso repetitivo, tedioso y propenso a errores, teniendo en cuenta que se insertan de forma manual todos los datos. Con el objetivo de contribuir a facilitar este proceso a los especialistas, se desarrolla un plugin que ofrece una mayor rapidez en el proceso de creación de esquemas dimensionales. El mismo es capaz de generar automáticamente esquemas dimensionales a partir de un diagrama Entidad-Relación y exportar el mismo en formato XML. Posee un sistema de autoguardado y opciones para gestionar los elementos del esquema dimensional.

2.2. Modelo de dominio

Un modelo de dominio es una representación visual de las clases conceptuales u objetos del mundo real en un dominio de interés. También se les denomina modelos conceptuales, modelo de objetos del dominio y modelos de objetos de análisis (Larman 2004). A continuación, en la figura 4, se muestra el diagrama del modelo del dominio para la solución propuesta:

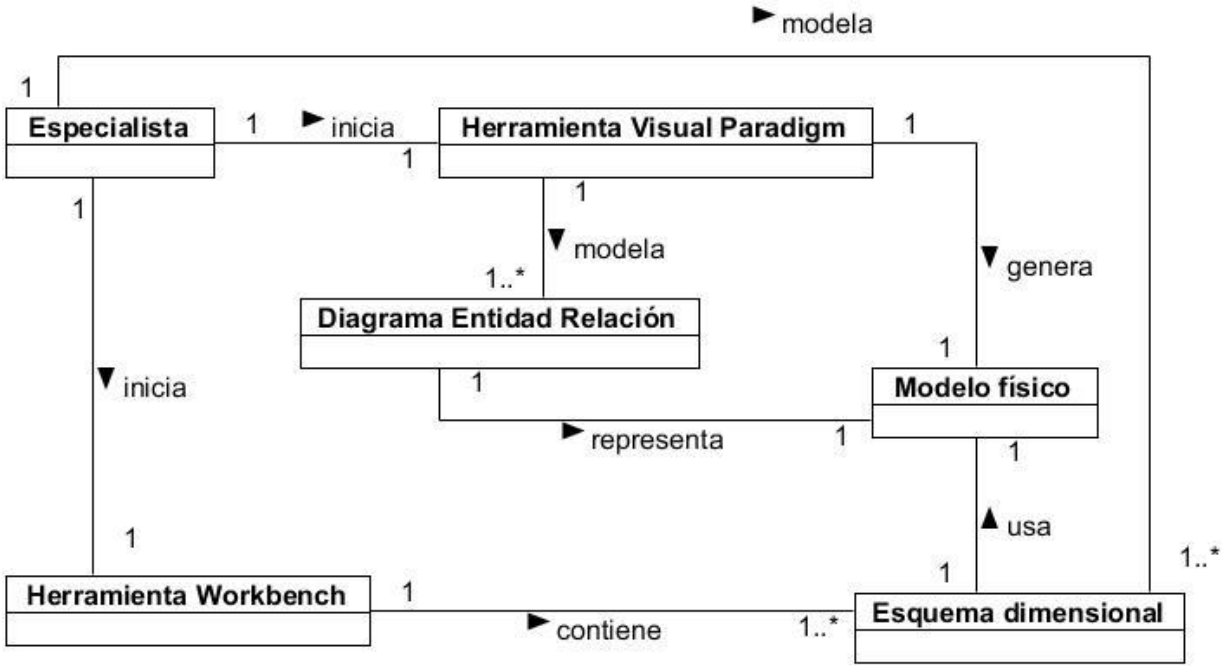

Figura 4: Modelo de dominio.

A continuación, se describen los conceptos del modelo de dominio presentado en la Figura 4.

Especialista: Especialista en inteligencia de negocios que se encarga de modelar un diagrama Entidad-Relación haciendo uso de la herramienta Visual Paradigm con el fin de crear un esquema dimensional haciendo uso de la herramienta Schema Workbench.

Herramienta Visual Paradigm: Es la herramienta que permite al especialista la construcción de los diagramas Entidad-Relación del cual se genera el modelo físico de la base de datos.

Diagrama Entidad-Relación: Es el diagrama generado por el especialista de inteligencia de negocios para representar las entidades relevantes de un sistema de información, así como sus interrelaciones y propiedades.

Esquema Dimensional: Es el esquema que permite organizar los datos en los sistemas de inteligencia de negocio, es contenido por la herramienta Schema Workbench y modelado por el especialista.

Herramienta Workbench: Es la herramienta gráfica que permite la construcción de los esquemas dimensionales. En el presente dominio está representado por la herramienta Pentaho Schema Workbench.

Modelo físico: Es un tipo de modelo de datos que determina la estructura lógica de una base de datos, el mismo está representado por un diagrama Entidad-Relación, generado por la herramienta Visual Paradigm y usado por el esquema dimensional para la obtención de datos.

2.3. Requisitos Funcionales

La ingeniería de requisitos, según Pressman, es un conjunto de procesos, tareas y técnicas que permiten la definición y gestión de los requisitos de un producto de un modo sistemático. Facilita los mecanismos adecuados para comprender las necesidades del cliente, analiza sus necesidades, confirma su viabilidad, negocia una solución razonable, especifica la solución sin ambigüedad, valida la especificación y gestiona los requisitos para que se transformen en un sistema operacional (Pressman 2005).

La especificación de requisitos funcionales (RF) documenta las operaciones y actividades que un sistema debe ser capaz de realizar. Estos deben incluir las descripciones de los datos que se introducen en el sistema, descripciones de las operaciones realizadas por cada pantalla, descripciones de flujos de trabajo realizadas por el sistema y descripciones de los informes del sistema u otras salidas (Pressman 2005).

A continuación, se muestran los RF identificados:

Tabla 3: Requisitos funcionales.

No.	Nombre
RF1	Seleccionar cubos.
RF2	Seleccionar dimensiones.
RF3	Generar esquema dimensional.
RF4	Exportar esquema dimensional.
RF5	Adicionar cubo.
RF6	Modificar cubo.
RF7	Eliminar cubo.
RF8	Visualizar cubo.
RF9	Adicionar dimensión.
RF10	Modificar dimensión.
RF11	Eliminar dimensión.

RF12	Visualizar dimensión.
RF13	Adicionar medida.
RF14	Modificar medida.
RF15	Eliminar medida.
RF16	Visualizar medida.
RF17	Adicionar miembro calculable.
RF18	Modificar miembro calculable.
RF19	Eliminar miembro calculable.
RF20	Visualizar miembro calculable.
RF21	Adicionar conjunto con nombre.
RF22	Modificar conjunto con nombre.
RF23	Eliminar conjunto con nombre.
RF24	Visualizar conjunto con nombre.
RF25	Adicionar función definida por el usuario.
RF26	Modificar función definida por el usuario.
RF27	Eliminar función definida por el usuario.
RF28	Visualizar función definida por el usuario.
RF29	Visualizar Jerarquía.
RF30	Adicionar Jerarquía.
RF31	Modificar Jerarquía.
RF32	Eliminar Jerarquía.
RF33	Visualizar Nivel.
RF34	Adicionar Nivel.
RF35	Modificar Nivel.
RF36	Eliminar Nivel.

RF37	Adicionar dimensión usada.
RF38	Modificar dimensión usada.
RF39	Eliminar dimensión usada.
RF40	Visualizar dimensión usada.

Patrón de Caso de Uso

Un patrón de casos de uso (CU) no describe un uso particular de un sistema, más bien, captura técnicas para que el modelo sea sostenible y entendible. Entonces, se puede decir que capturan mejores prácticas para modelar CU (Rodríguez 2015). En la agrupación de los requisitos en CU se hace uso de los siguientes patrones.

CRUD Completo

Este patrón consiste en un caso de uso que modela todas las operaciones que pueden ser realizadas sobre una parte de la información, tales como creación, lectura, actualización y eliminación. Suele ser utilizado cuando todos los flujos contribuyen al mismo valor del negocio y estos a su vez son cortos y simples.

El uso de este patrón en la investigación permitió agrupar los 40 requisitos funcionales en 11 CU, evidenciándose específicamente en los CU Gestionar dimensión usada, Gestionar medida, Gestionar función definida por el usuario, Gestionar nivel, Gestionar dimensión, Gestionar jerarquía, Gestionar conjunto con nombre, Gestionar cubo y Gestionar miembro calculable.

Entre las ventajas de la utilización del patrón CRUD pueden mencionarse (Rubenfa 2014):

- Se reúnen en un solo elemento de configuración del software todas las acciones básicas que se realizan sobre una entidad de dominio.
- Se facilita la comprensión por parte del cliente de la funcionalidad del sistema.
- Se facilita la especificación de los casos de uso, logrando un alto nivel de detalle sin tener que invertir esfuerzo en describir aspectos generales de funcionalidad más de una vez.
- Se facilita la reusabilidad del código, a partir de identificar relaciones entre los CU, con un mínimo de esfuerzo.

Extensión concreta

Una de las características básicas del patrón es que hay situaciones en que el CU de extensión no es indispensable que ocurra, y cuando lo hace ofrece un valor extra (extiende) al objetivo original del CU base.

El uso de este patrón se evidencia en el CU Exportar esquema dimensional, donde este extiende del caso de uso Generar esquema dimensional. Esto se debe a que una vez que el especialista indica generar el esquema dimensional el sistema lo guarda automáticamente en un directorio de carpetas establecido según el sistema operativo que se esté utilizando. Por lo que no es necesario exportar el esquema ya que se puede buscar en la ruta donde el sistema lo guarda. Pero si el especialista desea exportar el esquema en una ruta específica es donde entra en ejecución el caso de uso extendido Exportar esquema dimensional como parte del caso de uso base Generar esquema dimensional. En otras palabras, no es indispensable que ocurra el caso de uso extendido para obtener el esquema dimensional en un archivo con la extensión .xml.

2.4. Caso de uso

En ingeniería del software, un CU es una técnica para la captura de requisitos potenciales de un nuevo sistema o una actualización de software. Los mismos proporcionan uno o más escenarios que indican cómo debería interactuar el sistema con el usuario o con otro sistema para conseguir un objetivo específico, además son 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. Hace referencia al sistema a construir, detallando su comportamiento, el cual se traduce en resultados que pueden ser observados por el actor. Describen las cosas que los actores quieren que el sistema haga, por lo que un CU debería ser una tarea completa desde la perspectiva del actor (Rubenfa 2014).

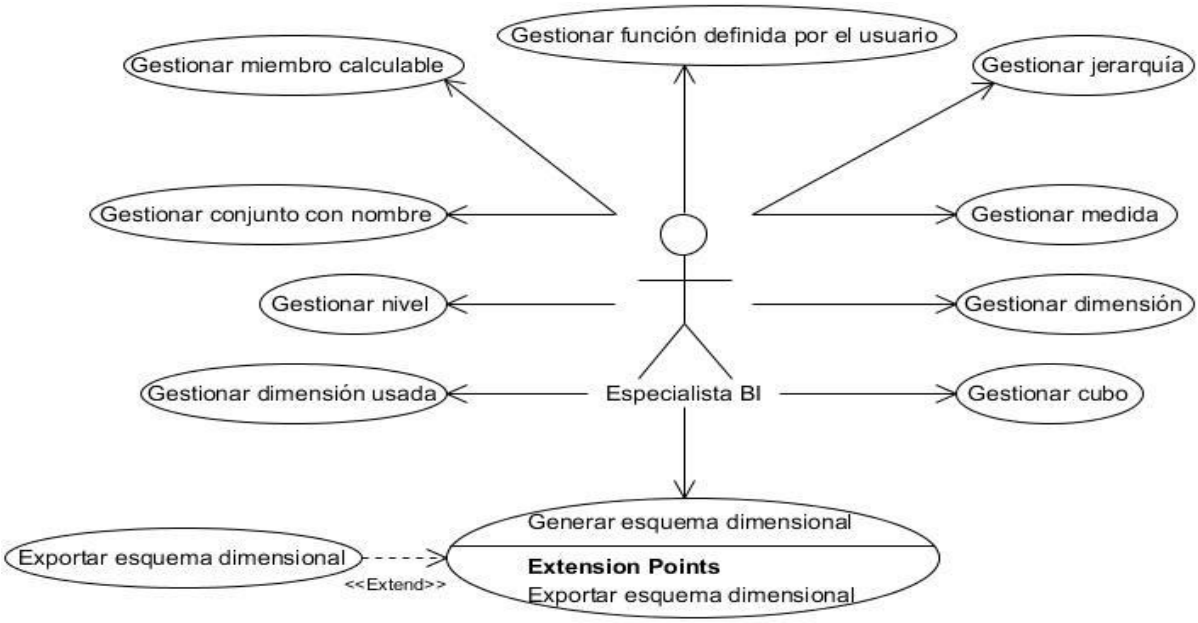

Figura 5: Diagrama de caso de uso del sistema.

Descripción de los conceptos del diagrama de CU del sistema:

Especialista de BI: Profesional encargado de generar los cubos con sus respectivas características, también genera los miembros calculables y todo el proceso de construcción del esquema dimensional con el plugin.

Generar esquema dimensional: Este CU permite generar un esquema dimensional, selecciona cubos y dimensiones y se encarga del autoguardado del mismo.

Gestionar dimensión compartida: Este CU permite adicionar, modificar, eliminar y visualizar una dimensión compartida.

Gestionar medida: Este CU permite adicionar, modificar, eliminar y visualizar una medida.

Gestionar función definida por el usuario: Este CU permite adicionar, modificar, eliminar y visualizar una función definida por el usuario.

Gestionar nivel: Este CU permite adicionar, modificar, eliminar y visualizar un nivel.

Gestionar dimensión: Este CU permite adicionar, modificar, eliminar y visualizar una dimensión.

Gestionar jerarquía: Este CU permite adicionar, modificar, eliminar y visualizar una jerarquía.

Gestionar conjunto con nombre: Este CU permite adicionar, modificar, eliminar y visualizar un conjunto con nombre.

Gestionar cubo: Este CU permite adicionar, modificar, eliminar y visualizar un cubo.

Gestionar miembro calculable: Este CU permite adicionar, modificar, eliminar y visualizar un miembro calculable.

Exportar esquema dimensional: Este CU permite, si el especialista desea, exportar el esquema dimensional a una dirección específica.

2.4.1. Descripción de los casos de uso del sistema

La especificación de los CU del sistema se refiere a la descripción de cada una de las partes definidas para lograr su descripción completa. 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 (Rodríguez 2015).

A continuación, se muestra la descripción del CU*: Generar esquema dimensional.* Ver el resto de las descripciones en el expediente del proyecto 0114_ESPECIFICACIÓN DE CASOS DE USO.

Tabla 4: Descripción del CU Generar esquema dimensional.

Objetivo	Permite generar un esquema dimensional a partir del diagrama Entidad-Relación que se encuentre activo en la herramienta Visual Paradigm(VP).
Actor	Especialista BI.
Resumen	El caso de uso inicia cuando el especialista selecciona la opción “Seleccionar hechos y dimensiones”. El sistema lista los hechos y las dimensiones contenidos en el diagrama Entidad-Relación activo. El especialista selecciona los hechos y dimensiones que desee incluir en el esquema, elige la opción “*Generar Esquema*”; el sistema genera el esquema dimensional automáticamente y muestra una vista al especialista con el esquema incorporado en un árbol jerárquico. Terminando así el caso de uso.
Complejidad	Alta
Prioridad	Alta

Precondiciones	Debe existir un diagrama Entidad-Relación activo y no debe estar vacío.
Postcondiciones	Se obtiene el esquema dimensional resultante a partir del diagrama Entidad-Relación seleccionado en un archivo con extensión .xml.

Flujo de eventos		
Flujo básico Generar esquema dimensional		
	Actor	**Sistema**
1.	Selecciona la opción "*Seleccionar hechos y dimensiones*".	2. Muestra una interfaz con la lista de hechos y de dimensiones contenidos en el diagrama Entidad Relación que se encuentre activo.
3.	Selecciona los hechos que desee incluir en el esquema dimensional.	4. Muestra las dimensiones relacionadas con los hechos seleccionados.
5.	Selecciona las dimensiones que desee incluir en el esquema dimensional.	
6.	Elige la opción "*Generar Esquema*".	7. Genera el esquema dimensional automáticamente, lo guarda en la ruta **`C:\Users\usuario\Documents\CGED_XML_STORE`** si el sistema operativo es Windows, si es Linux en **`/home/usuario/CGED_XML_STORE`** y muestra una vista con el esquema generado, en una estructura de árbol jerárquico. Finaliza el caso de uso.
	8. Selecciona la opción exportar el esquema dimensional en una dirección específica ver descripción de caso de uso extendido **Exportar esquema dimensional**.	

Flujos alternos		
No existe diagrama activo		
	Actor	**Sistema**
		2.a Emite un mensaje de error **"No existe diagrama activo"**. Se regresa al paso 1**.**
Diagrama no válido		
	Actor	**Sistema**
		2.b Emite un mensaje de error **"El diagrama que se encuentra activo no es de tipo Entidad Relación"**. Se regresa al paso 1**.**
Diagrama Entidad Relación vacío		
	Actor	**Sistema**
		2.c Emite un mensaje de error **"Diagrama Entidad Relación vacío"**. Se regresa al paso 1**.**
Cancelar		
	Actor	**Sistema**
	6.a Elige la opción "*Cancelar*".	6.b El sistema cierra la ventana que muestra los hechos y las dimensiones y finaliza el caso de uso.
Ningún elemento seleccionado		
	Actor	**Sistema**
		7.a Emite un mensaje de error **"Debe seleccionar algún cubo o dimensión"**. Se regresa al paso 2.
Relaciones	CU Incluidos	No procede
	CU Extendidos	Exportar esquema dimensional.

2.5. Requisitos no funcionales

Los requisitos no funcionales son los que especifican criterios para evaluar la operación de un servicio de tecnología de información, en contraste con los requerimientos funcionales que especifican los comportamientos específicos (pmoinformatica 2013). A continuación, los requisitos no funcionales identificados.

Apariencia o interfaz externa

- Se tendrá en cuenta que los íconos utilizados en la interfaz sean los utilizados en la herramienta *Schema Workbench* de la suite Pentaho.
- Al desplegar el plugin este deberá tener una interfaz semejante a la del *Schema Workbench* de la suite Pentaho en su versión 3.10.

Software

- GNU/Linux o Windows preferentemente Ubuntu 14.4 y Windows 7, además Máquina Virtual de Java (JDK por sus siglas en inglés) o Entorno de Ejecución de Java (JRE por sus siglas en inglés) versión 7.0 o superior.

Hardware

- Se deberá disponer de 512 de RAM y 20 GB de espacio en el disco duro como mínimo teniendo en cuenta el sistema operativo y las herramientas necesarias para el despliegue del plugin.

Soporte

- Proveer un manual de usuario para la interacción con el plugin.

Extensibilidad

- Se debe lograr un diseño adaptable con la capacidad de soportar funcionalidades adicionales o modificar las funcionalidades existentes sin impactar el resto de los requerimientos contemplados en el sistema.

2.6. Arquitectura de software propuesta

La arquitectura de software se refiere a las estructuras de un sistema, compuestas de elementos con propiedades visibles de forma externa y las relaciones que existen entre ellos. Es una representación de alto nivel de la estructura del sistema describiendo las partes que lo integran, puede incluir los patrones

que supervisan la composición de sus componentes y las restricciones al aplicar los patrones. 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 2005).

2.6.1. 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 2004). La herramienta Visual Paradigm proporciona una interfaz de programación para aplicaciones que permiten a los desarrolladores implementar y reutilizar clases e interfaces para desarrollar funciones agregadas de software. Para definir la arquitectura del plugin propuesto, resulta esencial regirse por los elementos arquitectónicos definidos en la interfaz de programación de aplicaciones (API) de la herramienta Visual Paradigm que está basada en el patrón arquitectónico Modelo-Vista-Controlador (MVC). Este patrón de arquitectura de software 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 arquitectura de software se basa en las ideas de reutilización de código y la separación de conceptos, características que buscan facilitar las tareas 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 (Larman 2004).
- **La Vista**: Las vistas, como su nombre nos hace entender, contienen el código de nuestra aplicación que va a producir la visualización de las interfaces de usuario, o sea, el código que nos permitirá renderizar los estados de nuestra aplicación (Larman 2004).
- **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 el 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 (Larman 2004).

2.6.2. Diagrama de clases del diseño

El diagrama de clase es el diagrama principal de diseño y análisis para un sistema. En este, 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 la implementación. Un diagrama de clases de diseño muestra la especificación para las clases de software de una aplicación. Incluye la siguiente información: clases, asociaciones y atributos, interfaces con sus operaciones, constantes, métodos, navegabilidad y dependencias (Larman 2004).

A continuación, se muestra el diagrama de clases del diseño para el CU Generar esquema dimensional (ver figura 6), el mismo está diseñado según el patrón arquitectónico Modelo-Vista-Controlador, definido para establecer la estructura, organización y dependencias funcionales entre las clases que lo integran. El paquete *Vistas* contiene las interfaces que interactúan directamente con el usuario. Cada una de estas clases contiene un objeto de la clase controladora que le corresponde en el paquete Controlador. Es en este paquete donde se encuentran las clases que se encargan de responder a las acciones del usuario que se inician desde las vistas. Las clases controladoras son las encargadas de recibir los datos introducidos por el usuario en las vistas y delegar responsabilidades entre las clases del paquete Modelo que se encargan de guardar la información referente a los elementos del esquema dimensional. Ver el resto de los diagramas de clases del diseño en el expediente del proyecto Modelo de diseño.

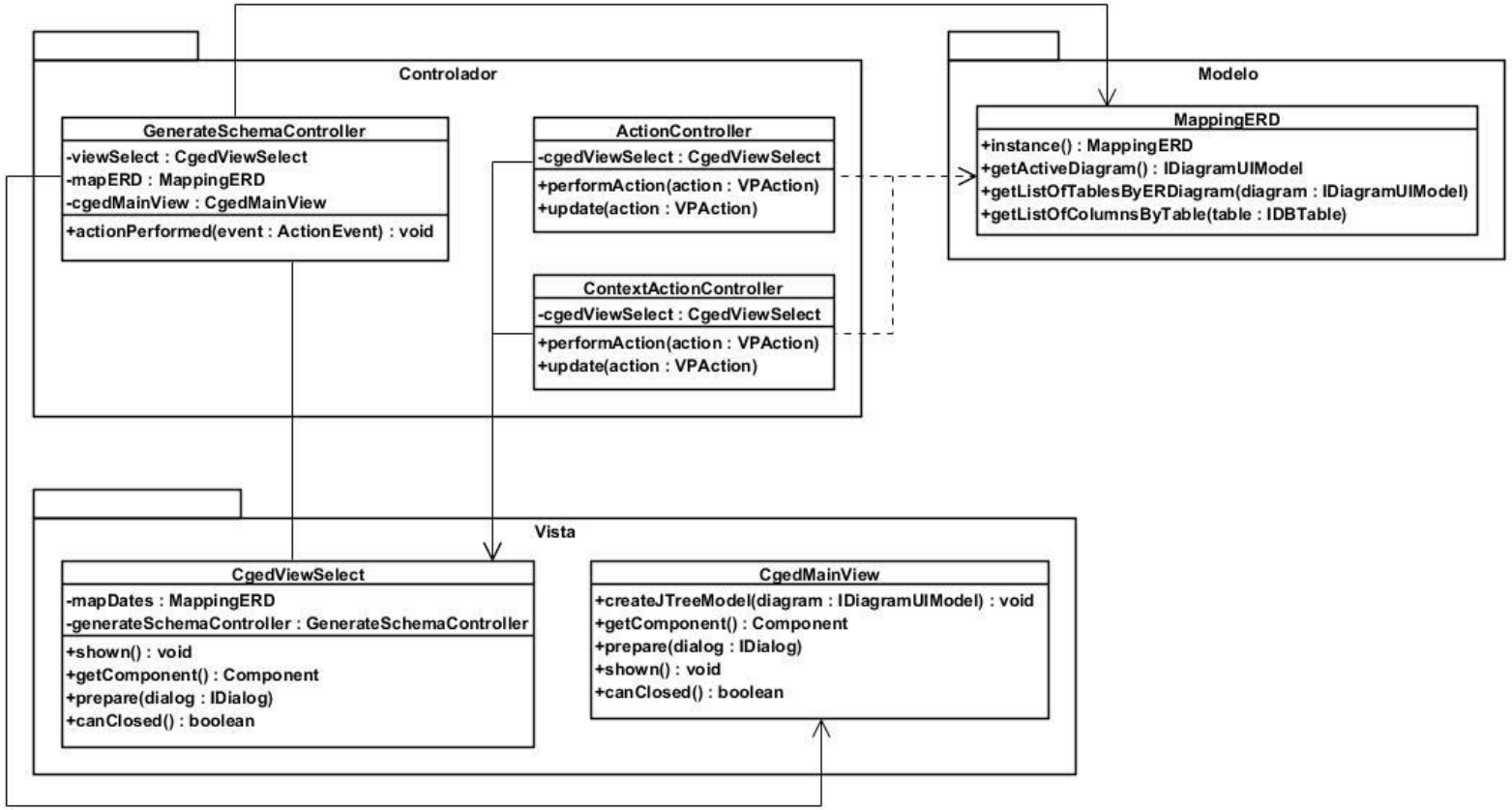

Figura 6: Diagrama de clases del diseño CU Generar esquema dimensional.

2.7. Patrones de diseño

Los patrones de diseño son soluciones para problemas típicos y recurrentes que se puede encontrar a la hora de desarrollar una aplicación. Estos son agrupados fundamentalmente por dos grandes grupos conocidos como: Patrones de software para la asignación general de responsabilidad (GRASP) y La Banda de los cuatro (GoF) (Rubenfa 2014).

2.7.1. Patrones GRASP

Los patrones GRASP son utilizados para describir los principios fundamentales de diseño de objetos para la asignación de responsabilidades (Mora 2011). Los empleados en el desarrollo del plugin se relacionan a continuación:

- Experto en información: Consiste en asignar una responsabilidad al experto en información: la clase que cuenta con la información necesaria para cumplir la responsabilidad. Experto es un patrón que se utiliza más que cualquier otro al asignar responsabilidades; es un principio básico que suele utilizarse en el diseño orientado a objetos. Con él no se pretende designar una idea oscura y extraña; expresa simplemente la intuición de que los objetos hacen cosas relacionadas con la información que poseen (Larman 2004).

Este patrón se pone en evidencia en la clase NodeCube, esta almacena toda la información referente a un cubo perteneciente al esquema dimensional que se desea generar, lo que la clasifica como elemento de tipo modelo según el patrón arquitectónico MVC (Ver figura 7).

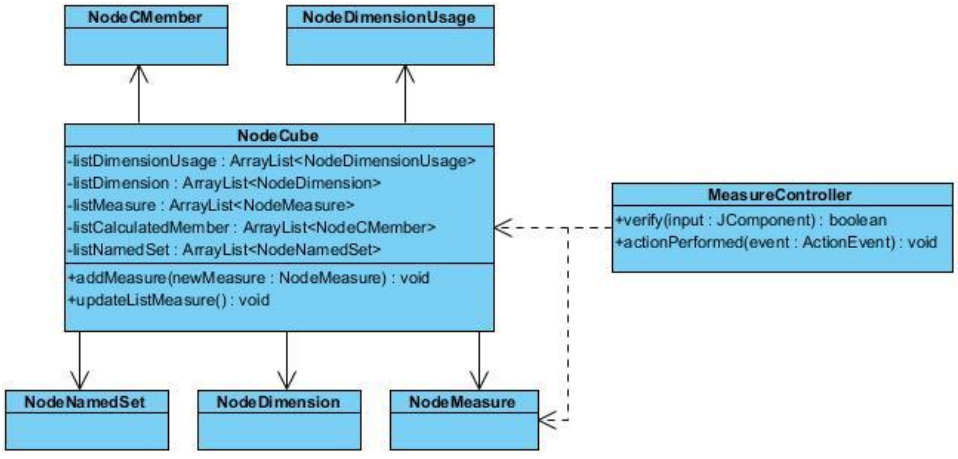

Figura 7: Patrón Experto.

- Alta cohesión: Consiste en asignar una responsabilidad de modo que la cohesión siga siendo alta. En la perspectiva del diseño orientado a objetos, la cohesión es una medida de cuan relacionadas y enfocadas están las responsabilidades de una clase. Una alta cohesión caracteriza a las clases con responsabilidades estrechamente relacionadas que no realicen un trabajo enorme. Se dice que una clase tiene alta cohesión si la misma tiene responsabilidades moderadas en un área funcional y colabora con las otras para llevar a cabo las tareas (Larman 2004).

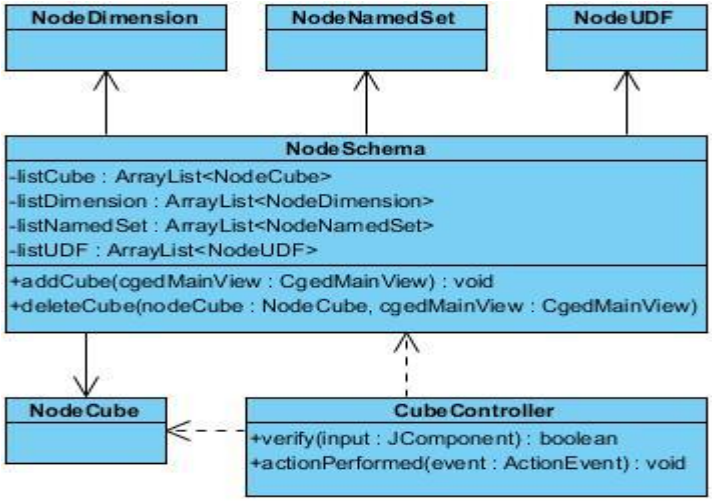

Figura 8: Alta cohesión.

En la figura 8 se ejemplifica como el controlador CubeController maneja la gestión de la información referente a un elemento de tipo cubo en conjunto con la clase NodeCube que es la contenedora de la información del cubo que se desea gestionar. Para eliminar el cubo seleccionado, CubeController debe acceder a una instancia de la clase NodeSchema que es el padre directo de los elementos de tipo cubo y la única autorizada a eliminar o insertar uno nuevo. De igual forma, alta cohesión se pone de manifiesto en las clases contenedoras de información que representan la parte del modelo dentro de la arquitectura del plugin.

La clase NodeSchema es la contenedora del esquema una vez generado este, esta clase se divide y está estrechamente relacionada con NodeCube, NodeDimension, NodeNamedSet y NodeUDF que son los hijos directos de la raíz del esquema. En otras palabras, NodeSchema divide la información ente las clases antes mencionadas, mejorando la claridad y la facilidad con que se entiende el diseño, se simplifica el mantenimiento y las mejoras en funcionalidad (Ver figura 8).

- Bajo Acoplamiento: Consiste en asignar una responsabilidad de manera que el acoplamiento permanezca bajo. El acoplamiento es una medida de la fuerza con que un elemento está conectado a, tiene conocimiento de, confía en otros elementos. Un elemento con bajo (o débil) acoplamiento no depende de demasiados otros elementos. Estos elementos pueden ser clases, subsistemas, sistemas, etcétera (Larman 2004).

Este patrón se pone en evidencia en las clases que representan a los controladores del plugin ya que para cada caso de uso se diseñó una clase controladora. Esta clase se encarga de manejar toda la información relacionada con la gestión de un elemento en el esquema dimensional. Por lo que solo necesita tener visibilidad sobre la clase que representa a la vista y al modelo para el caso de uso, permaneciendo el acoplamiento entre clases lo más bajo posible. Los controladores también se encargan de delegar la mayoría de las responsabilidades entre las clases del modelo, obteniéndose una clase con pocas responsabilidades, más sencilla y fácil de mantener, dando soporte a una alta cohesión.

- Controlador: Consiste en asignar la responsabilidad de controlar el flujo de eventos del sistema, a clases específicas. Un controlador es un objeto de interfaz no destinada al usuario que se encarga de manejar un evento del sistema, define además el método de su operación (Larman 2004).

Este patrón se ve reflejado en las clases que representan los controladores de la aplicación que son las encargadas de manejar los eventos iniciados por el usuario desde las vistas. También trae consigo un mayor potencial de los componentes reutilizables, esto garantiza que los procesos de dominio sean manejados por la capa de los objetos del dominio y no por la de la interfaz. El hecho de delegar a un controlador la responsabilidad de las operaciones del sistema entre las clases del dominio soporta la reutilización de la lógica para manejar los procesos a fines del negocio en aplicaciones futuras. Por ejemplo, la clase DimensionController se encarga de manejar los eventos relacionados con la gestión de la información de un elemento de tipo dimensiones en esquema dimensional. (Ver figura 9).

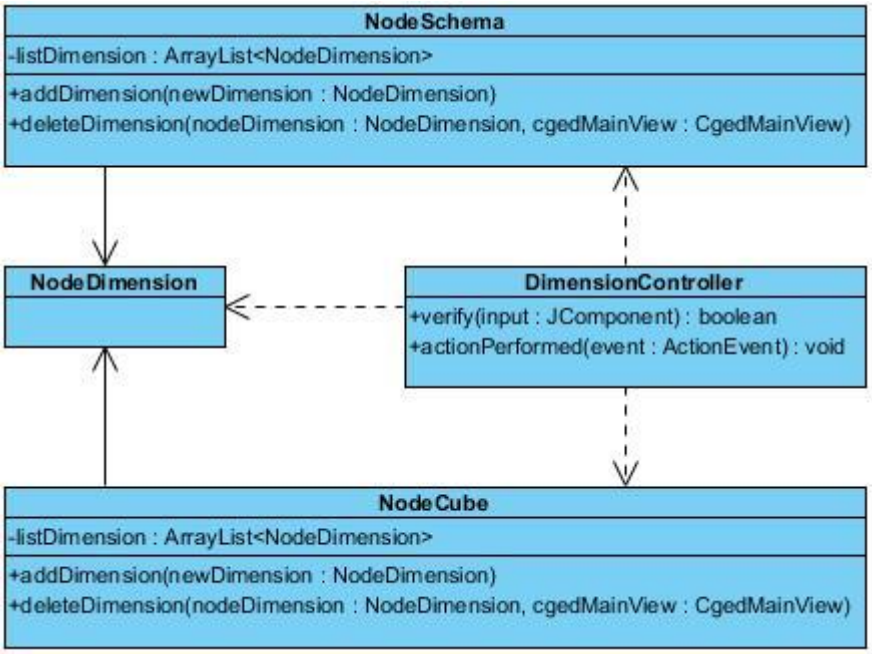

Figura 9: Controlador.

- 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 debemos conectar con el objeto producido en cualquier evento (Larman 2004).

Este patrón se pone de manifiesto en la clase ContextActionController encargada de responder al evento de seleccionar la opción “Generar Esquema Dimensional” por vía sensitivo al contexto (clic derecho encima del diagrama que se encuentre activo en la herramienta Visual Paradigm), por lo que esta clase clasifica como controladora según el patrón arquitectónico MVC. ContextActionController tiene la responsabilidad de crear un objeto de tipo CgedViewSelect que es la vista encargada de mostrar el listado de cubos y dimensiones que serán seleccionados para posteriormente generar el esquema dimensional. Todo lo antes expuesto hace que esta clase sea idónea para asumir la responsabilidad de crear la instancia de CgedViewSelect para que sea mostrada al usuario. (Ver figura 10).

```
public class ContextActionController implements VPContextActionController {

    private final ViewManager viewManager;
    private static CgedViewSelect cgedViewSelected;

    public ContextActionController() {
        viewManager = ApplicationManager.instance().getViewManager();
    }

    @Override
    public void performAction(VPAction action, VPContext vpc, ActionEvent ae) {

        MapeoDER mapeo = MapeoDER.instance();
        mapeo.setIsVPcontext(true);
        mapeo.setVpc(vpc);

        if (vpc.getDiagram().getType().equals(DiagramManager.DIAGRAM_TYPE_ENTITY_RELATIONSHIP_DIAGRAM)) {

            if (!vpc.getDiagram().isDiagramElementEmpty()) {
                cgedViewSelected = new CgedViewSelect();
                viewManager.showDialog(cgedViewSelected);
```

Figura 10: Patrón Creador.

2.7.2. Patrones GoF

Describen soluciones simples y elegantes a problemas específicos en el diseño de software orientado a objetos. Eric Braude define los patrones de diseño como combinaciones de componentes, casi siempre clases y objetos que por experiencia se sabe que resuelven ciertos problemas de diseño comunes. En términos generales es posible decir que un patrón de diseño es una solución a un problema recurrente en el diseño de software. Los empleados en el desarrollo del complemento se relacionan a continuación:

- *Singleton*: Garantiza la existencia de una única instancia para una clase y la creación de un mecanismo de acceso global a dicha instancia. En el diseño e implementación del plugin se hizo uso de este patrón en la clase MappingERD ya que esta constituye una clase de mapeo de datos donde es común la utilización de este patrón, por esto no se necesitan varias instancias de MappingERD dentro de una determinada clase que utilice alguna funcionalidad de sus funcionalidades. Ver figura 11.

```
public class MappingERD {

    private static MappingERD instance;

    public static MappingERD instance() {
        if (instance == null) {
            instance = new MappingERD();
        }
        return instance;
    }

    private MappingERD() {
    }
```

Figura 11: Patrón Singleton.

- *Iterator*: 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). La figura 12 muestra el uso de este patrón en el diseño e implementación del plugin.

```
public ArrayList<IDBTable> ListaDeTablasPorDiagrama(IDiagramUIModel diagram) {

    ArrayList<IDBTable> lista_taBTables = new ArrayList<IDBTable>();

    Iterator diagramElementIter = diagram.diagramElementIterator();
    while (diagramElementIter.hasNext()) {
        IDiagramElement diagramElement = (IDiagramElement) diagramElementIter.next();

        if (diagramElement.getShapeType().equals("DBTable")) {
            IDBTable table = (IDBTable) diagramElement.getModelElement();
            if (table.getName() != null) {
                lista_taBTables.add(table);
```

Figura 12: Patrón Iterator.

2.8. Conclusiones del capítulo

Después de realizar el proceso de diseño del plugin para generar esquemas dimensionales se identificaron 40 requisitos funcionales que fueron agrupados en 11 casos de usos haciendo uso del patrón CRUD completo y Extensión Concreta, lo cual permitió la elaboración del diagrama de caso de uso. Se definieron además los requisitos no funcionales que debe cumplir la aplicación. A través de las clases del diseño se logra la correcta estructura de clases del sistema y se solucionan problemas típicos y recurrentes que se puede encontrar a la hora de desarrollar una aplicación mediante patrones de diseño.

Capítulo 3: Implementación y pruebas del plugin para la generación de esquemas dimensionales a partir del modelo Entidad-Relación desde el Visual Paradigm.

Este capítulo comprende la implementación del plugin, mediante la construcción del diagrama de componentes y la definición de los estándares de codificación. Además, se plasman los resultados arrojados por las pruebas realizadas a la propuesta de solución, las cuales fueron realizadas con el objetivo de verificar el cumplimiento de las peticiones hechas por el cliente y garantizar de esta forma la calidad de la propuesta de solución.

3.1. Diagrama de componentes

Los diagramas de componentes se utilizan para describir la vista de implementación estática de un sistema, representa como un sistema de software, es dividido en componentes y muestra las dependencias entre estos. Los diagramas de componentes prevalecen en el campo de la arquitectura de software, pero pueden ser usados para modelar y documentar cualquier arquitectura de sistema. Son utilizados para formar sistemas de software de cualquier tamaño y complejidad. Un componente es una parte física de un sistema (Roldan 2009).

Existen diferentes tipos de componentes, algunos son descritos a continuación:

- **Executable:** Especifica un componente que se puede ejecutar en un nodo.
- **Library:** Especifica una biblioteca de objetos estática o dinámica.
- **Folder:** Especifica un paquete que contiene en su interior uno o varios componentes.
- **File:** Especifica un componente que representa un documento que contiene código fuente o datos.

A continuación, se representa el diagrama de componentes correspondiente al caso de uso ***Generar Esquema Dimensional*** ya que es el más importante de la aplicación, sencillo y fácil de entender. El mismo describe cada uno de los componentes asociados al diseño de clases propuesto para el caso de uso, así como la relación de dependencia entre los componentes que integran el plugin. Para visualizar el diagrama de componentes general ver anexo 1.

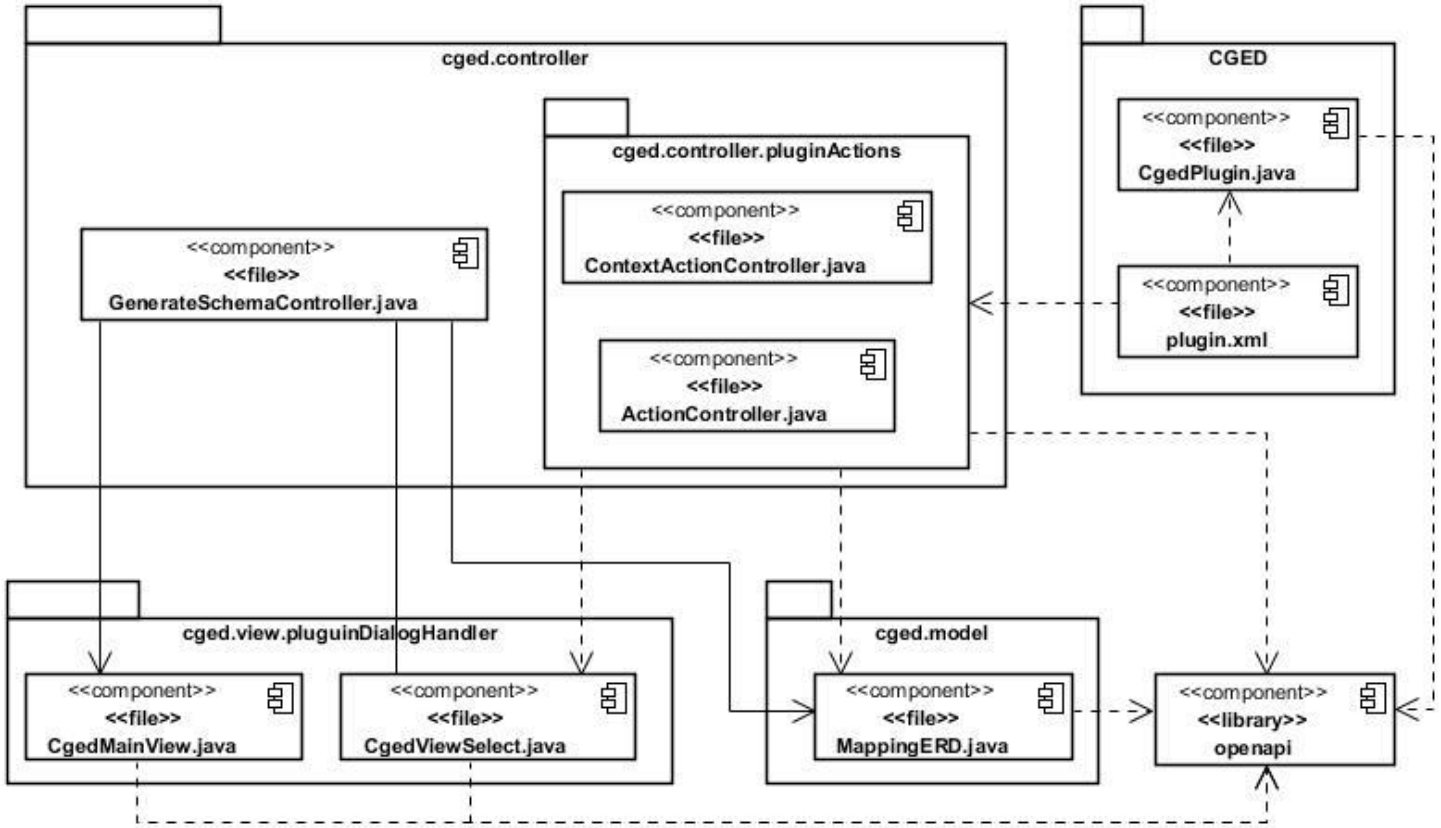

Figura 13: Diagrama de Componentes CU Generar Esquema Dimensional.

Descripción de los componentes más relevantes:

- ***Paquete CGED:*** Agrupa los archivos asociados a la configuración del plugin como son *plugin.xml* y *plugin.properties* definidos en la estructura que propone la herramienta Visual Paradigm para el plugin. También contiene la clase principal de la aplicación *CgedPlugin.java.*
- ***plugin.xml:*** Su función consiste en la configuración del plugin, define su nombre, descripción, proveedor, librerías a utilizar y la clase principal del mismo (en este caso la clase *CgedPlugin.java*) así como las acciones y los controladores de acciones a ejecutar.
- ***CgedPlugin.java***: Es la clase principal del plugin; en el archivo plugin.xml se hace referencia a esta clase debido a que implementa la interfaz *VPPlugin* definida en la API de desarrollo de la herramienta Visual Paradigm (Open API) y necesaria para el correcto funcionamiento del mismo, esta interfaz define los métodos *loaded()*, encargado de cargar el plugin capturando la información definida en el archivo *plugin.xml* este método provee un objeto de tipo *VPluginInfo* que proporciona el identificador del plugin y el directorio donde se encuentra desplegado, la interfaz también define el método *unloaded()* ejecutado cuando se sale de la herramienta.
- ***Paquete cged.controller.pluginActions:*** Contiene las clases controladoras encargadas de responder y procesar los eventos iniciados a nivel de la herramienta Visual Paradigm (eventos iniciados por el usuario desde la interfaz visual de Visual Paradigm).
- ***ActionController.java:*** Es el controlador encargado de responder a la acción *"Almacén de datos -> Seleccionar cubos y dimensiones"* que se encuentra en la barra de herramientas del Visual Paradigm. Esta acción es definida en el archivo *plugin.xml* y se le asocia un controlador de acción, en este caso *ActionController.java*; el mismo implementa la interfaz *VPActionController* definida por la Open API para el manejo de acciones en el Visual Paradigm desde la barra de herramientas. Al seleccionarse la opción, *ActionController.java* obtiene mediante una instancia de la clase del modelo *MappingERD.java* un listado de cubos y otro de dimensiones necesarios para crear una instancia de la vista *CgedViewSelect.java,* la cual muestra estos listados al especialista BI para que seleccione los elementos que desee incluir en el esquema dimensional.
- ***ContextActionController.java:*** Es el controlador encargado de responder a la acción "*Seleccionar cubos y dimensiones"* que se muestra haciendo clic derecho encima del diagrama Entidad-Relación que se encuentre activo en la herramienta Visual Paradigm. Esta acción es definida en el

archivo *plugin.xml* y se le asocia un controlador de acción, en este caso *ContextActionController.java*; el mismo implementa la interfaz *ContextActionController.java* definida por la Open API para el manejo de las acciones a nivel de contexto (clic derecho). Al seleccionarse la opción, *ContextActionController.java* obtiene mediante una instancia de la clase del modelo *MappingERD.java* un listado de cubos y otro de dimensiones necesarios para crear una instancia de la vista *CgedViewSelect.java.* Esta última muestra estos listados al especialista BI para que seleccione los elementos que desee incluir en el esquema dimensional.

- ***Paquete cged.controller:*** Agrupa las clases referentes a los controladores del caso de uso, encargados de responder a las acciones iniciadas por el usuario desde las vistas.
- ***GenerateSchemaController.java:*** Es el controlador encargado de procesar el evento que responde a la acción de generar automáticamente del esquema dimensional. Este controlador está asociado a la opción *"Generar esquema"* que se encuentra en la vista *CgedViewSelect.java*, *GenerateSchemaController.java* necesita una instancia de la clase del modelo *MappingERD.java* para utilizar varias de sus funcionalidades. Esta obtiene el listado de cubos y dimensiones seleccionados por el especialista y con esta información genera el esquema dimensional. Luego de obtenida dicha información, esta es transferida a la vista *CgedMainView.java* la cual lo muestra en forma de una estructura de árbol jerárquico.
- ***Paquete cged.view.pluginDialogHandler:*** Agrupa las vistas principales del plugin, estas se encuentran estrechamente implicadas en el proceso de generación del esquema dimensional.
- ***CgedViewSelect.java:*** Es la vista encargada de mostrar un listado de cubos y dimensiones al especialista de BI para que elija cuál de estos desea generar en el esquema dimensional. La clase implementa la interfaz *IDialogHandler* definida en la Open API y necesaria para mostrar la vista como un diálogo por encima de la interfaz visual de la herramienta Visual Paradigm. La clase *CgedViewSelect.java* es creada y mostrada por los controladores *ActionController.java* y *ContextActionController.java****.***
- ***CgedMainView.java:*** Representa la vista principal del plugin, muestra el esquema dimensional una vez generado en forma de estructura de árbol jerárquico. Muestra además una barra de herramientas con opciones para gestionar los elementos del esquema. La clase implementa la interfaz *IDialogHandler* definida en la Open API y necesaria para mostrar la vista como un diálogo

por encima de la interfaz visual de la herramienta Visual Paradigm. *CgedMainView.java* es creada y mostrada por el controlador *GenerateSchemaController.java.*

- ***Paquete cged.model:*** Agrupa los componentes que representan a los modelos del caso de uso, aquellas clases contenedoras de información que intervienen en la generación del esquema dimensional.
- ***MappingERD.java:*** Es la clase que proporciona todos los atributos y métodos necesarios para obtener la información del diagrama Entidad-Relación que se encuentre activo en la herramienta Visual Paradigm. Esta información es requerida para la generación automática del esquema dimensional.
- ***Openapi.jar:*** Librería que permite al plugin trabajar con los recursos de Visual Paradigm-UML para su posterior integración. Es la encargada de proporcionar todas las clases e interfaces para la construcción de proyectos en Visual Paradigm.

3.2. Estándares de codificación

Un estándar de codificación son reglas que se siguen para la escritura del código fuente, de tal manera que a otros programadores se les facilite entender el código (MADEJA 2013).

Las convenciones de código son importantes para los programadores por un gran número de razones (MADEJA 2013):

- El 80% del costo del código de un programa va a su mantenimiento.
- Las convenciones de código mejoran la lectura del software, permitiendo entender código nuevo mucho más rápidamente y más a fondo.
- Si distribuyes tu código fuente como un producto, necesitas asegurarte de que está bien hecho y presentado como cualquier otro producto.

En el desarrollo del Plugin para generar esquemas dimensionales a partir del diagrama Entidad-Relación desde el Visual Paradigm sobre la plataforma Java, se pusieron en práctica los estándares de codificación mostrados a continuación.

Organización de los ficheros

Un fichero consiste de secciones que deben estar separadas por líneas en blanco y comentarios opcionales que identifican cada sección (MADEJA 2013). Serán evitados los ficheros de más de 2000

líneas pues son incómodos y en ocasiones provoca que la clase no encapsule un comportamiento claramente definido.

Identación

Se evitarán las líneas de más de 80 caracteres, ya que no son manejadas bien por muchas terminales y herramientas. Cuando una expresión no entre en una línea, se romperá de acuerdo a estos principios:

- Romper después de una coma.
- Romper antes de un operador.

Declaraciones

Se realizará una declaración por línea, ya que facilita los comentarios. Se inicializará las variables locales donde se declaran. Se pondrán las declaraciones solo al principio de los bloques para evitar confusión en programadores no preavisados.

En las declaraciones de clases e interfaces se tendrá en cuenta no usar ningún espacio en blanco entre el nombre de un método y el paréntesis que abre su lista de parámetros. Los métodos se separarán con una línea en blanco.

Sentencias

Todas las sentencias de un bloque deberán encerrarse entre llaves, aunque el bloque conste de una única sentencia.

Espacios en blanco

Las líneas en blanco mejorarán la facilidad de lectura separando secciones de código que están lógicamente relacionadas. Se hará uso de dos líneas en blanco entre secciones de un fichero fuente y entre las definiciones de clases e interfaces y una línea en blanco entre métodos, variables locales de un método y su primera sentencia y entre las distintas secciones lógicas de un método para facilitar la lectura.

Declaraciones de clases e interfaces

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

- Ningún espacio en blanco entre el nombre de un método y el paréntesis "(" que abre su lista de parámetros.
- La llave de apertura "{" aparece al final de la misma línea de la sentencia declaración.

- Los métodos se separan con una línea en blanco.

También se tuvo en cuenta:

1. Importar clases específicamente, no utilizar * imports.

2. Eliminar las importaciones de clases que no se utilicen.

3. Variables, parámetros y métodos: Se deberá utilizar CamelCase para nombrar variables, clases y parámetros de métodos.

4. Paquetes: El prefijo de un nombre de paquete único, siempre se escribe en letras minúsculas.

5. No se debe anteponer ningún prefijo que represente el tipo de datos de variables o constantes.

6. El nombre del archivo debe coincidir con el nombre de la clase y debe respetar la notación Pascal Case.

El uso de estos estándares fue de gran importancia para alcanzar la calidad necesaria del plugin desarrollado y para aumentar la legibilidad del mismo.

3.3. Pruebas del sistema

La prueba del software es un elemento crítico para la garantía de 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 2005). A continuación, se detalla la estrategia de prueba empleada.

3.3.1. Pruebas funcionales

A nivel de sistema se realizan pruebas funcionales, estas se centran en las necesidades del cliente. Permite al ingeniero obtener conjuntos de condiciones de entrada que ejerciten completamente todos los requisitos funcionales de un programa, es decir consideran la función para la cual fue creado el producto (lo que hace). Se llevan a cabo sobre la interfaz del sistema reduciendo el número de casos de prueba mediante la elección de entradas y salidas válidas y no válidas que ejercitan toda la funcionalidad del sistema (Torres 2016).

Dentro de esta clasificación se hace uso de la técnica partición de equivalencia perteneciente al método de prueba de caja negra que divide el dominio de entrada de un programa en clases de datos de los que pueden derivarse casos de prueba. En (Torres 2016) se plantea que:

- Se dirige a la definición de casos de prueba que descubran clases de errores, reduciendo así el número total de casos de prueba que hay que desarrollar.
- El diseño de casos de prueba para la partición equivalente se basa en una evaluación de las clases de equivalencia para una condición de entrada.
- Una clase de equivalencia representa un conjunto de estados válidos y no válidos para condiciones de entrada.
- Una condición de entrada es un valor numérico específico, un rango de valores, un conjunto de valores relacionados o una condición lógica.

Casos de Prueba

Según la IEEE, el estándar 610 define un caso de prueba como *"Conjunto de entrada de prueba, condición de ejecución y resultado esperado desarrollados para un objetivo en particular, como el ejercicio de una ruta de programa en particular o para verificar el cumplimiento de un requisito específico y como documentación que especifique las entradas, los resultados previos y un conjunto de condiciones de ejecuciones de un elemento de prueba"* (Torres 2016). Para la presente investigación se definieron ocho casos de prueba en correspondencia con los CU. A continuación, se presenta un ejemplo del diseño de caso de prueba del CU Gestionar cubo. Ver demás casos de prueba en el expediente del proyecto 010318_Diseño de casos de prueba.

Tabla 5: SC Adicionar cubo.

Escenario	Descripción	V #1	V #2	V #3	V #4	V #5	V #6	Respuesta del sistema	Flujo central
EC 1.1 Adicionar cubo	*En este escenario se realiza la adición de un nuevo cubo al sistema correctamente.*	*V*	*V*	*V*	*V*	*V*	*V*	*El sistema visualiza el cubo creado.*	*1. Seleccionar la raíz del esquema. 2. Luego la opción "Adicionar cubo".*
		Nuevo cubo 5	*Cubo*	*Nuevo cubo 5*	*true*	*true*	*true*		

EC 1.2 Adicionar cubo (no está seleccionada la raíz del esquema)	*Este escenario no debe permitir crear un cubo si no se encuentra seleccionada la raíz del esquema.*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*El sistema muestra una notificación "Para adicionar un nuevo cubo debe seleccionar el esquema"*	*1. No tener seleccionada la raíz del esquema. 2. Luego la opción "Adicionar cubo".*

Tabla 6: SC Modificar cubo.

Escenario	Descripción	V #1	V #2	V #3	V #4	V #5	V #6	Respuesta del sistema	Flujo central
EC 1.1 Modificar cubo	*En este escenario se modifica un cubo correctamente.*	*V*	*V*	*V*	*V*	*V*	*V*	*El sistema visualiza el cubo modificado.*	*1. Seleccionar cubo a modificar. 2. Realizar modificaciones.*
		Nuevo cubo #5	*Cubo*	*Nuevo cubo 5*	*true*	*true*	*true*		
EC 1.2 Modificar cubo (campo nombre vacío)	*Este escenario no debe permitir modificar un cubo si el campo nombre está vacío.*	*I*	*V*	*V*	*V*	*V*	*V*	*El sistema muestra una notificación "El campo nombre del cubo es obligatorio".*	*1. Seleccionar el cubo a modificar. 2. Dejar campo nombre vacío.*
		(vacío)	*Cubo*	*Nuevo cubo 6*	*true*	*true*	*true*		
EC 1.2 Modificar cubo (cubo existente)	*Este escenario no debe permitir modificar un cubo si existe otro con el mismo nombre en el esquema.*	*I*	*V*	*V*	*V*	*V*	*V*	*El sistema muestra una notificación "Error al cambiar la propiedad nombre: Cubo Nuevo cubo 7 ya existe".*	*1. Seleccionar el cubo a modificar. 2. Nombrar el cubo con el mismo nombre de un cubo en existencia.*
		Nuevo cubo #5	*Cubo*	*Nuevo cubo 7*	*true*	*true*	*true*		

Tabla 7: SC Eliminar cubo.

Escenario	Descripción	V #1	V #2	V #3	V #4	V #5	V #6	Respuesta del sistema	Flujo central
EC 1.1 Eliminar cubo	*En este escenario se elimina el cubo correctamente.*	*V*	*V*	*V*	*V*	*V*	*V*	*El sistema elimina el cubo.*	*1. Seleccionar cubo a eliminar. 2. Se selecciona la opción Eliminar*
		Nuevo cubo 5	*Cubo*	*Nuevo cubo 5*	*true*	*true*	*true*		

									ubicada en los íconos de la interfaz principal. 3.Se da clic en la opción Sí.
EC 1.2 Eliminar cubo (no está seleccionado el cubo a eliminar)	*Este escenario no debe permitir eliminar el cubo si el mismo no está seleccionado.*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*El sistema no es capaz de eliminar el cubo de no estar seleccionado.*	

Tabla 8: SC Visualizar cubo (Modo edición).

Escenario	Descripción	V #1	V #2	V #3	V #4	V #5	V #6	Respuesta del sistema	Flujo central
EC 1.1 Visualizar cubo	*En este escenario se visualiza el cubo correctamente.*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*El sistema visualiza el cubo seleccionado.*	*1. Seleccionar cubo a visualizar.*

Tabla 9: SC Modificar cubo (Modo xml).

Escenario	Descripción	V #1	V #2	V #3	V #4	V #5	V #6	Respuesta del sistema	Flujo central
EC 1.1 Visualizar cubo	*En este escenario se visualiza el código .xml del cubo correctamente.*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*El sistema visualiza el código .xml del cubo seleccionado.*	*1. Seleccionar cubo a visualizar. 2. Se selecciona la opción Modo xml ubicada en los íconos de la interfaz principal.*

Tabla 10: Descripción de las variables.

No	Nombre de campo	Clasificación	Valor Nulo	Descripción
1	nombre	campo de texto	No	En este campo se le permite al usuario nombrar el cubo y permite caracteres tales como: • A-Z, a-z, 0-9 • Caracteres especiales: ¡$%&/()=?¿

2	descripción	campo de texto	Si	En este campo se le permite al usuario hacer una descripción del cubo a crear y permite caracteres tales como: • A-Z, a-z, 0-9 Caracteres especiales: ¡$%&/()=?¿
3	subtítulo	campo de texto	Si	En este campo se le permite al usuario poner un subtítulo al cubo creado y permite caracteres tales como: • A-Z, a-z, 0-9 Caracteres especiales: ¡$%&/()=?¿
4	cache	checkbox	Si	Especifica si Mondrian mantendrá o no en cache este cubo. Su valor por defecto es true.
5	enabled	checkbox	Si	Indica si este cubo debe ser tenido en cuenta o ignorado.
6	visible	checkbox	Si	Especifica si se verá o no en la interfaz de usuario esta medida. Por defecto, su valor es true.

Resultados obtenidos

Luego de ser aplicados los casos de pruebas diseñados se detectaron un total de 13 no conformidades. En la primera iteración se detectaron ocho no conformidades de las cuales tres fueron referidas a las funcionalidades y cinco a ortografía. Durante la segunda iteración se verificó que las no conformidades detectadas en la primera iteración fueron resueltas surgiendo además cinco nuevas no conformidades, cuatro de funcionalidad y una de ortografía, las cuales fueron resueltas en su totalidad verificándose esto en la tercera iteración. Para conocer el listado de no conformidades ver anexo 2.

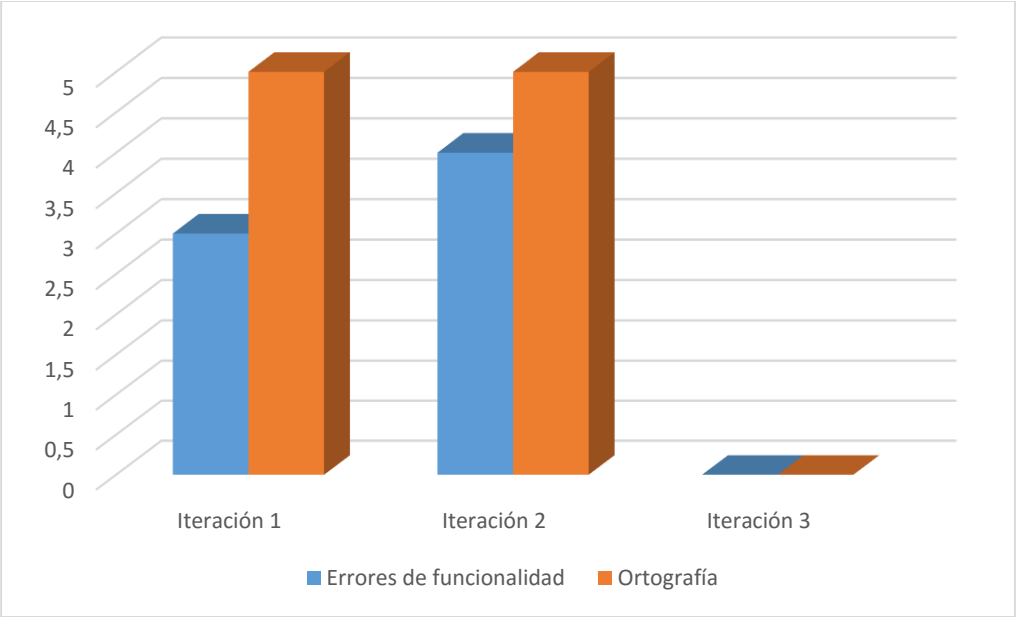

Figura 14: Resultados de las pruebas funcionales.

3.3.2. Prueba de Integración

Las pruebas de integración están diseñadas para probar la interacción entre los distintos componentes de un sistema. Para la integración del plugin se llevaron a cabo pruebas incrementales haciendo uso de la integración ascendente en la cual el programa fue construido y probado en pequeños incrementos. Esto facilitó el aislamiento de errores y las pruebas sistemáticas.

La herramienta Visual Paradigm propone una estructura de despliegue compuesta por una carpeta con el nombre plugins, que se encuentra dentro del directorio de instalación de la misma. Donde todas las clases están organizadas según el patrón arquitectónico utilizado. En la carpeta controller se encuentran las clases controladoras del plugin; la carpeta model contiene las clases que representan a los modelos y la carpeta view contiene las vistas del plugin. (ver figura 15).

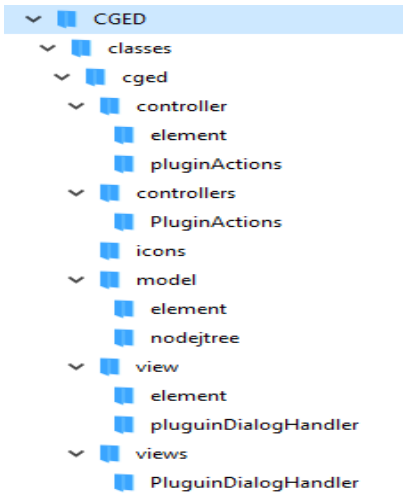

Figura 15: Estructura de despliegue del plugin.

Para las pruebas de integración teniendo en cuenta la diferencia entre la estructura de implementación del plugin (ver figura 16) y la estructura de integración con la herramienta se hizo uso de una herramienta de despliegue (ver figura 17) la cual facilitaría la integración con el Visual Paradigm. La herramienta para su funcionamiento necesita de la dirección origen del plugin y de la dirección origen de la carpeta de instalación del Visual Paradigm, luego de contar con estas dos direcciones se realiza el despliegue de la aplicación.

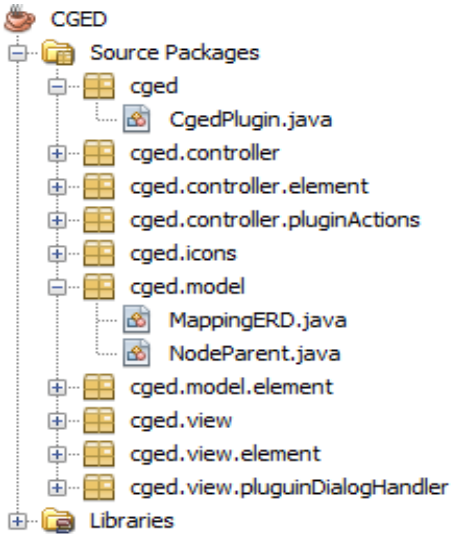

Figura 16: Estructura de implementación del plugin.

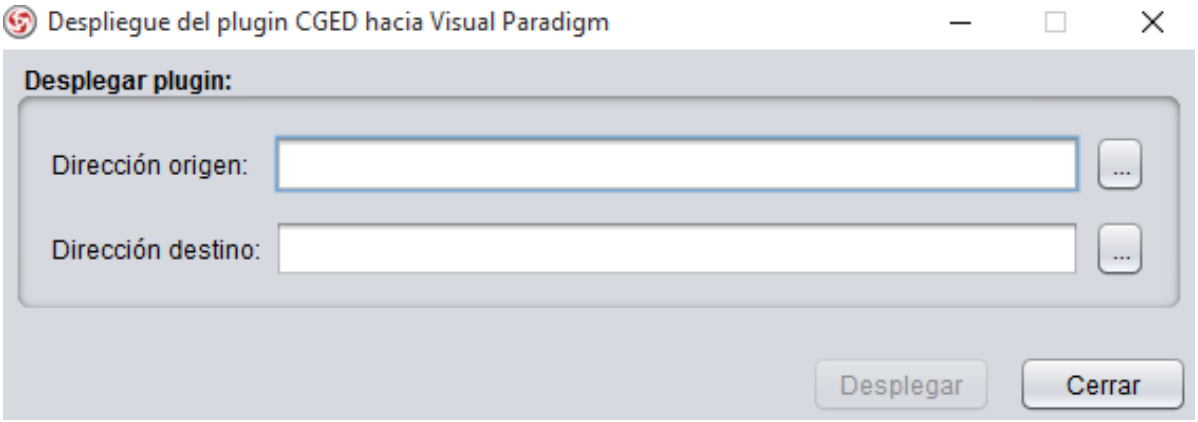

Figura 17: Herramienta de despliegue.

Las pruebas de integración fueron exitosamente realizadas logrando la correcta integración con la herramienta Visual Paradigm.

3.3.3. Pruebas de aceptación

Con el objetivo de validar que el sistema cumple con el funcionamiento esperado y permitir que el usuario determine su aceptación desde el punto de vista de su funcionalidad y rendimiento, se realizan pruebas de aceptación. Para ello se tuvieron en cuenta los casos de prueba de mayor criticidad. Las pruebas fueron realizadas exitosamente haciendo entrega por parte del cliente de la carta de aceptación (ver Anexo 3).

3.4. Conclusiones del capítulo

En el presente capítulo se detallaron los elementos correspondientes a la implementación del plugin para generar esquemas dimensionales y como conclusiones se plantean que con el modelado del diagrama de componentes se establecieron las principales relaciones entre clases y una mejor organización. Como resultado de la implementación se llevaron a cabo pruebas funcionales sobre la interfaz del plugin, pruebas de integración para examinar su calidad y las pruebas de aceptación del cliente afirmaron que el sistema es correcto y cumple con los requisitos trazados. Se obtuvieron un total de 13 no conformidades en tres iteraciones y se logró una correcta integración del plugin a la herramienta Visual Paradigm.

Conclusiones generales

Una vez finalizada la investigación desarrollada se arribó a las siguientes conclusiones:

- El estudio de los fundamentos teóricos de la investigación, permitió identificar las principales características del plugin para generar esquemas dimensionales a partir del modelo Entidad-Relación desde el Visual Paradigm, así como la selección de la metodología y las herramientas necesarias para su desarrollo.
- A partir de la identificación de los requisitos funcionales y no funcionales se abordaron las principales características y capacidades que debe poseer el plugin para generar esquemas dimensionales a partir del modelo Entidad-Relación desde el Visual Paradigm.
- El diseño de la propuesta de solución permitió la implementación de un plugin multiplataforma, que automatiza el proceso de generar esquemas dimensionales desde el Visual Paradigm.
- La aplicación de los casos de pruebas, la prueba de integración y la prueba de aceptación permitió obtener un plugin funcional que cumple con los requisitos especificados por el cliente.

Recomendaciones

Partiendo de que con el desarrollo del presente trabajo se dio cumplimiento a su objetivo general se recomienda lo siguiente:

- Agregar al plugin desarrollado la capacidad de manejar esquemas avanzados que contengan elementos como esquemas copo de nieve o hechos transaccionales.
- Agregar al plugin desarrollado la capacidad de definir jerarquías por cada dimensión antes de generar el esquema.

Referencias Bibliográficas

BERNABEU, R.D., 2007. *HEFESTO: Metodología propia para la Construcción de un Data*. Córdoba, Argentina: s.n.

DARIO, B., 2009. DataPrix. [en línea]. Disponible en: http://www.dataprix.com/data-warehousing-y-metodologia-hefesto/i-data-warehousing-investigacion-y-sistematizacion-conceptos-6.

DÍAZ, L.L. y PAZ, C.R.L., 2008. *El uso de la Plataforma de Inteligencia de Negocio PENTAHO en el Modulo Generador de Reportes del Sistema GREHUCORP*. S.l.: s.n.

ECLIPSE, 2014. Eclipse Process Framework Project (EPF). [en línea]. [Consulta: 25 junio 2016]. Disponible en: http://www.eclipse.org/epf/.

EDUKAVITAL, 2015. Definición de Plugin – Significado de Plugin : Educación para la Vida. [en línea]. Disponible en: http://edukavital.blogspot.com/2013/05/plugin-definicion-de-plugin-concepto-de.html.

ERWIN, 2016. ERwin. [en línea]. Disponible en: http://erwin.com/.

EXES, 2014. XML ¿Qué es? | Manual de XML. *Área de Programación y Desarrollo* [en línea]. [Consulta: 25 junio 2016]. Disponible en: http://www.mundolinux.info/que-es-xml.htm.

IBM, 2013. IBM Knowledge Center - IBM InfoSphere Data Architect V7.5.3 welcome page. *Creación de modelos dimensionales lógicos y físicos.* [en línea]. [Consulta: 25 junio 2016]. Disponible en: http://www.ibm.com/support/knowledgecenter/SS9UM9_7.5.3/com.ibm.datatools.nav.doc_7.1.5v20101007_2230/topics/helpindex_rda.html.

IZAGUIRRE, L.E. y ALARCÓN, L.F., 2008. Modelación multidimensional: un mecanismo de mejora para la gestión de proyectos de construcción. [en línea], vol. 8, no. 3. Disponible en: http://www.seer.ufrgs.br/ambienteconstruido/article/view/5368.

JUAREZ, E.A.O., 2013. Open UP: Metodología Open UP. *Open UP* [en línea]. [Consulta: 25 junio 2016]. Disponible en: http://openupeaojmp.blogspot.com/2013/09/metodologia-open-up.html.

KIMBALL, R., 1996. *El Juego de Herramientas del Almacén de Datos*. S.l.: s.n.

LARMAN, C., 2004. *UML y Patrones. Introducción al análisis y diseño orientado a objetos.* La Habana: Félix Varela.

LUTZ, M., 2009. *Learning Python*. S.l.: s.n.

MADEJA, 2013. Convenio de codificación específico para Java | Marco de Desarrollo de la Junta de Andalucía. [en línea]. [Consulta: 25 junio 2016]. Disponible en: http://www.juntadeandalucia.es/servicios/madeja/contenido/libro-pautas/15.

MORA, R.C., 2011. Patrones de GRASP | adictosaltrabajo. [en línea]. [Consulta: 25 junio 2016]. Disponible en: https://www.adictosaltrabajo.com/tutoriales/grasp/.

OCHANDO, M.B., 2014. Fundamentos y Diseño de Bases de Datos: Modelo entidad-relación ER. *Fundamentos y Diseño de Bases de Datos* [en línea]. [Consulta: 25 junio 2016]. Disponible en: http://ccdoc-basesdedatos.blogspot.com/2013/02/modelo-entidad-relacion-er.html.

OMG, 2016. OMG. [en línea]. Disponible en: http://www.omg.org/spec/UML/2.1.2/.

ORACLE, 2016. Oracle Cloud. [en línea]. Disponible en: https://www.oracle.com/es/index.html.

ORACLE CORPORATION, 2000. NetBeans. [en línea]. Disponible en: http://netbeans.org.

PMOINFORMATICA, 2013. Requerimientos No Funcionales: Porque son importantes - La Oficina de Proyectos de Informática. [en línea]. [Consulta: 25 junio 2016]. Disponible en: http://www.pmoinformatica.com/2013/01/requerimientos-no-funcionales-porque.html.

PRESSMAN, R.S., 2005. *Ingeniería del software. Un enfoque práctico*. 6. S.l.: s.n.

RODRÍGUEZ, A., 2013. Netbeans, Eclipse, JCreator, JBuilder… ¿Cuál es el mejor entorno de desarrollo (IDE) para Java? (CU00613B). [en línea]. [Consulta: 25 junio 2016]. Disponible en: http://aprenderaprogramar.com/index.php?option=com_content&view=article&id=398:netbeans-eclipse-jcreator-jbuilder-icual-es-el-mejor-entorno-de-desarrollo-ide-para-java-cu00613b&catid=68:curso-aprender-programacion-java-desde-cero&Itemid=188.

RODRÍGUEZ, S.C., 2015. Patrones de Casos de Uso. | SG. [en línea]. [Consulta: 25 junio 2016]. Disponible en: http://sg.com.mx/revista/6/patrones-casos-uso.

ROLDAN, D., 2009. Diagrama de Componentes: VENTAJAS Y DESVENTAJAS DE LOS CASOS DE USO. *Diagrama de Componentes* [en línea]. [Consulta: 25 junio 2016]. Disponible en: http://exposicionuml40099.blogspot.com/2009/02/ventajas-y-desventajas-de-los-casos-de.html.

RONDÓN, Y.Q., DOMÍNGUEZ, L.C. y BERENGUER, A.D., 2011. Diseño de la base de datos para sistemas de digitalización y gestión de medias., vol. 8. ISSN 1667.

RUBENFA, 2014. Patrones de diseño: qué son y por qué debes usarlos. *Genbeta Dev* [en línea]. [Consulta: 25 junio 2016]. Disponible en: http://www.genbetadev.com/metodologias-de-programacion/patrones-de-diseno-que-son-y-por-que-debes-usarlos.

TORRES, G.M., 2016. Web de la asignatura Laboratorio de proyectos. [en línea]. [Consulta: 25 junio 2016]. Disponible en: http://indalog.ual.es/mtorres/LP/index.php?opcion=inicio.

VALDÉZ, J.L.C., 2013. *IMPLEMENTACIÓN DEL MODELO INTEGRAL COLABORATIVO (MDSIC) COMO FUENTE DE INNOVACIÓN PARA EL DESARROLLO ÁGIL DE SOFTWARE EN LAS EMPRESAS DE LA ZONA CENTRO - OCCIDENTE EN MÉXICO*. S.l.: s.n.

VELASCO, R.H., 2014. Almacenes de datos. [en línea]. Disponible en: http://www2.rhernando.net/modules/tutorials/doc/bd/dw.html.

Bibliografía

ALVAREZ, M.A., [sin fecha]. Qué es Java. *DesarrolloWeb.com* [en línea]. [Consulta: 27 junio 2016]. Disponible en: http://www.desarrolloweb.com/articulos/497.php.

BAZAN, D., 2013. BASE DE DATOS ESTRATEGICAS. *BASE DE DATOS ESTRATEGICAS* [en línea]. [Consulta: 27 junio 2016]. Disponible en: https://diegobazan7.wordpress.com/.

BÉDARD, J.W., 2011. *EL PARADIGMA MULTIDIMENSIONAL: DESARROLLO DE NUEVAS TECNOLOGIAS PARA LA GESTION DEL TERRITORIO*. S.l.: s.n.

BERNABEU, R.D., 2007. *HEFESTO: Metodología propia para la Construcción de un Data*. Córdoba, Argentina: s.n.

BÖCK, H., 2011. *The Definitive Guide to NetBeans™ Platform 7*. S.l.: s.n.

BOUDEAU, T., 2002. *NetBeans: the definitive guide*. S.l.: s.n.

CCM, 2014. Lenguajes de programación. *CCM* [en línea]. [Consulta: 27 junio 2016]. Disponible en: http://es.ccm.net/contents/304-lenguajes-de-programacion.

CONTRERAS, J.A., 2013. *Contenidos de las materias de Base de Datos en los Planes de Estudio Universitarios de Grado en Informática en el EEES. En 12 Conferencia Iberoamericana en Sistemas, Cibernetica e Informática*. Florida: s.n.

CÓRDOBA, I.R.D., 2009. *Data Warehousing: Investigación y Sistematización de Conceptos*. S.l.: s.n.

DARIO, B., 2009. DataPrix. [en línea]. Disponible en: http://www.dataprix.com/data-warehousing-y-metodologia-hefesto/i-data-warehousing-investigacion-y-sistematizacion-conceptos-6.

DÍAZ, L.L. y PAZ, C.R.L., 2008. *El uso de la Plataforma de Inteligencia de Negocio PENTAHO en el Modulo Generador de Reportes del Sistema GREHUCORP*. S.l.: s.n.

ECLIPSE, 2014. Eclipse Process Framework Project (EPF). [en línea]. [Consulta: 25 junio 2016]. Disponible en: http://www.eclipse.org/epf/.

EDUKAVITAL, 2015. Definición de Plugin – Significado de Plugin : Educación para la Vida. [en línea]. Disponible en: http://edukavital.blogspot.com/2013/05/plugin-definicion-de-plugin-concepto-de.html.

ERWIN, 2016. ERwin. [en línea]. Disponible en: http://erwin.com/.

EXES, 2014. XML ¿Qué es? | Manual de XML. *Área de Programación y Desarrollo* [en línea]. [Consulta: 25 junio 2016]. Disponible en: http://www.mundolinux.info/que-es-xml.htm.

HERNÁNDEZ, L., 2013. MODELO DE IMPLEMENTACIÓN. [en línea]. [Consulta: 27 junio 2016]. Disponible en: http://ithleovi.blogspot.com/2013/06/unidad-5-modelo-deimplementacion-el.html.

HERRERA, C.K., 2007. Tutoriales en adictos al trabajo/Datawarehouse. [en línea]. [Consulta: 27 junio 2016]. Disponible en: https://www.adictosaltrabajo.com/.

IBM, 2013. IBM Knowledge Center - IBM InfoSphere Data Architect V7.5.3 welcome page. *Creación de modelos dimensionales lógicos y físicos.* [en línea]. [Consulta: 25 junio 2016]. Disponible en:

http://www.ibm.com/support/knowledgecenter/SS9UM9_7.5.3/com.ibm.datatools.nav.doc_7.1.5v20101007_2230/topics/helpindex_rda.html.

IZAGUIRRE, L.E. y ALARCÓN, L.F., 2008. Modelación multidimensional: un mecanismo de mejora para la gestión de proyectos de construcción. [en línea], vol. 8, no. 3. Disponible en: http://www.seer.ufrgs.br/ambienteconstruido/article/view/5368.

JUAREZ, E.A.O., 2013. Open UP: Metodología Open UP. *Open UP* [en línea]. [Consulta: 25 junio 2016]. Disponible en: http://openupeaojmp.blogspot.com/2013/09/metodologia-open-up.html.

KIMBALL, R., 1996. *El Juego de Herramientas del Almacén de Datos*. S.l.: s.n.

LARMAN, C., 2004. *UML y Patrones. Introducción al análisis y diseño orientado a objetos.* La Habana: Félix Varela.

LARMAN, C. y HALL, P., 2003. *Modelo del dominio.* S.l.: s.n.

LEOPOLDO ZENAIDO ZEPEDA SÁNCHEZ, 2008. *Tesis doctoral Metodología para el Diseño Conceptual*. 2008. S.l.: s.n.

LUTZ, M., 2009. *Learning Python*. S.l.: s.n.

MADEJA, 2013. Convenio de codificación específico para Java | Marco de Desarrollo de la Junta de Andalucía. [en línea]. [Consulta: 25 junio 2016]. Disponible en: http://www.juntadeandalucia.es/servicios/madeja/contenido/libro-pautas/15.

MORA, R.C., 2011. Patrones de GRASP | adictosaltrabajo. [en línea]. [Consulta: 25 junio 2016]. Disponible en: https://www.adictosaltrabajo.com/tutoriales/grasp/.

OCHANDO, M.B., 2014. Fundamentos y Diseño de Bases de Datos: Modelo entidad-relación ER. *Fundamentos y Diseño de Bases de Datos* [en línea]. [Consulta: 25 junio 2016]. Disponible en: http://ccdoc-basesdedatos.blogspot.com/2013/02/modelo-entidad-relacion-er.html.

OFNI SYSTEMS, 2014. Functional Requirements (Functional Requirement Specifications, Functional Specs, FRS, FS). *Ofni Systems* [en línea]. [Consulta: 27 junio 2016]. Disponible en: http://www.ofnisystems.com/services/validation/functional-requirements/.

OLGUIN, E.A., 2013. Metodología Open UP. [en línea]. Disponible en: http://openupeaojmp.blogspot.com/2013/09/metodologia-open-up.html.

OMG, 2016. OMG. [en línea]. Disponible en: http://www.omg.org/spec/UML/2.1.2/.

ORACLE, 2016. Oracle Cloud. [en línea]. Disponible en: https://www.oracle.com/es/index.html.

ORACLE CORPORATION, 2000. NetBeans. [en línea]. Disponible en: http://netbeans.org.

PAU URQUIZU, 2012. Business Intelligence fácil. [en línea]. Disponible en: http://www.businessintelligence.info/definiciones/que-es-modelo-dimensional.html.

PEREZ, R.M. y ESPONDA, E.R., 2013. Manual de Metodologia de Investigaciones. [en línea]. Disponible en: http://www.sld.cu/galerias/pdf/sitios/cielam/manual_de_metodologia_deinvestigaciones._1.pdf.

PMOINFORMATICA, 2013. Requerimientos No Funcionales: Porque son importantes - La Oficina de Proyectos de Informática. [en línea]. [Consulta: 25 junio 2016]. Disponible en: http://www.pmoinformatica.com/2013/01/requerimientos-no-funcionales-porque.html.

PRESSMAN, R.S., 2005. *Ingeniería del software. Un enfoque práctico*. 6. S.l.: s.n.

RODRÍGUEZ, A., 2013. Netbeans, Eclipse, JCreator, JBuilder… ¿Cuál es el mejor entorno de desarrollo (IDE) para Java? (CU00613B). [en línea]. [Consulta: 25 junio 2016]. Disponible en: http://aprenderaprogramar.com/index.php?option=com_content&view=article&id=398:netbeans-eclipse-jcreator-jbuilder-icual-es-el-mejor-entorno-de-desarrollo-ide-para-java-cu00613b&catid=68:curso-aprender-programacion-java-desde-cero&Itemid=188.

RODRÍGUEZ, S.C., 2015. Patrones de Casos de Uso. | SG. [en línea]. [Consulta: 25 junio 2016]. Disponible en: http://sg.com.mx/revista/6/patrones-casos-uso.

ROLDAN, D., 2009. Diagrama de Componentes: VENTAJAS Y DESVENTAJAS DE LOS CASOS DE USO. *Diagrama de Componentes* [en línea]. [Consulta: 25 junio 2016]. Disponible en: http://exposicionuml40099.blogspot.com/2009/02/ventajas-y-desventajas-de-los-casos-de.html.

RONDÓN, Y.Q., DOMÍNGUEZ, L.C. y BERENGUER, A.D., 2011. Diseño de la base de datos para sistemas de digitalización y gestión de medias., vol. 8. ISSN 1667.

RUBENFA, 2014. Patrones de diseño: qué son y por qué debes usarlos. *Genbeta Dev* [en línea]. [Consulta: 25 junio 2016]. Disponible en: http://www.genbetadev.com/metodologias-de-programacion/patrones-de-diseno-que-son-y-por-que-debes-usarlos.

RUIZ DURÁN, S., PILOTO LASTRA, Y. y ROSELLÓ NÚÑEZ, R., 2008. El peligro de un Caso de Uso muy largo. Mitos y realidades., vol. 1, no. 4.

SUN MICROSYSTEMS INC, 1999. Convenciones de código para el lenguaje de programación Java - Convenciones_Codigo_Java.pdf. [en línea]. [Consulta: 25 junio 2016]. Disponible en: http://mmc.geofisica.unam.mx/cursos/edp/Herramientas/Java/JavaStyle/Convenciones_Codigo_Java.pdf.

TORRES, G.M., 2016. Web de la asignatura Laboratorio de proyectos. [en línea]. [Consulta: 25 junio 2016]. Disponible en: http://indalog.ual.es/mtorres/LP/index.php?opcion=inicio.

VALDÉZ, J.L.C., 2013. *IMPLEMENTACIÓN DEL MODELO INTEGRAL COLABORATIVO (MDSIC) COMO FUENTE DE INNOVACIÓN PARA EL DESARROLLO ÁGIL DE SOFTWARE EN LAS EMPRESAS DE LA ZONA CENTRO - OCCIDENTE EN MÉXICO*. S.l.: s.n.

VELASCO, R.H., 2014. Almacenes de datos. [en línea]. Disponible en: http://www2.rhernando.net/modules/tutorials/doc/bd/dw.html.

ZORRILLA, M., 2008. *Data warehouse y OLAP*. S.l.: s.n.