Universidad de las Ciencias Informáticas
FACULTAD 6

Extensión de la herramienta Visual Paradigm para la generación de las clases de acceso a datos con Doctrine 2.0.

Trabajo de Diploma para optar por el título de

Ingeniero en Ciencias Informáticas

Autor: Adrian Trujillo Oliva

Tutores: Ing. Yanet Rosales Morales
Ing. Michael Eduardo Marrero Clark

La Habana, junio de 2013
"Año 55 de la Revolución"

DECLARACIÓN DE AUTORÍA

Declaro ser autor de la presente tesis y reconocer a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

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

Adrian Trujillo Oliva

_____
Firma del Autor

Yanet Rosales Morales

_____
Firma del Tutor

Michael Eduardo Marrero Clark

_____
Firma del Tutor

DATOS DE CONTACTO

Autor:
Adrian Trujillo Oliva
Universidad de las Ciencias Informáticas
e-mail: atoliva@estudiantes.uci.cu

Tutora:
Ing. Yanet Rosales Morales
Universidad de las Ciencias Informáticas
e-mail: yrmorales@uci.cu.

Tutor:
Ing. Michael E. Marrero Clark
Universidad de las Ciencias Informáticas
e-mail: memarrero@uci.cu.

Agradecimientos

Con esta tesis termina una etapa en mi vida de esfuerzo y estudio, por eso quiero expresar mis agradecimientos:

A mis padres por ser mi razón de ser, por haber creído en mí en momentos en que yo mismo no lo hacía, por todo su amor, y sacrificios que han hecho para que hoy este aquí.

A mi familia por apoyarme en todo momento de forma incondicional e impulsarme a seguir.

A mi novia por haber entrado en mi vida y darme todos los momentos hermosos que hemos vivido juntos.

A mis amigos por haber hecho más amena la estancia en la Universidad.

A mis profesores de todos los niveles de enseñanza que han contribuido a mi formación, transmitiéndome sus conocimientos y sobre todo a aquellos que aún se preocupan por mí.

A mis tutores por la dedicación y ayuda brindada que ha permitido la realización de este trabajo.

A todos los que de una forma u otra aportaron su granito de arena, ¡Gracias!

Dedicatoria

Un éxito adquiere más valor cuanto mayor fue el sacrificio para alcanzarlo. Siempre se persigue teniendo en mente a aquellos que confían en ti y que esperan que lo logres. Este sueño, transformado en éxito, quiero dedicarlo a las personas más importantes de mi vida:

A mi papá: que desde la distancia supo siempre estar presente y darme el apoyo incondicional que me dio fuerza para recorrer todo el camino. Por mantenerse al tanto de cuanto acontecía en mi vida y formar parte imprescindible de ella.

Y de manera especial

A mi mamá: por ser mi guía y mi ejemplo, por estar siempre a mi lado en los momentos buenos y malos. Por su amor infinito, sus consejos y apoyo constante. Por haber sabido educarme, formarme y darme lo mejor de sí. Por ser un ejemplo de dedicación, entrega, y sobre todo por ser mi mayor orgullo.

RESUMEN

En el proceso de desarrollo de software generalmente se utiliza la técnica de programación orientada a objeto y se emplea el sistema de base de datos de tipo relacional. La diferencia entre estos paradigmas induce a los desarrolladores a convertir los objetos del lenguaje de programación a registros de la base de datos y de igual forma la operación inversa. La inexistencia de una herramienta que facilitara las operaciones y que generara de forma automática las clases de acceso a datos para Doctrine 2.0, promovieron la construcción de la extensión Cader. La construcción fue realizada a través de las tres etapas de implementación definidas, para superar la incompatibilidad que existe entre las clases de acceso a datos generadas desde el Visual Paradigm y la versión actual de Doctrine. Este es el principal resultado que se alcanza con la realización de la investigación al lograr integrar satisfactoriamente las clases generadas con Doctrine 2.0, además de la disminución del tiempo de desarrollo de la capa de acceso a datos comprobado mediante el método Delphi. Estos resultados fueron obtenidos mediante el uso de un grupo de tecnologías y herramientas libres como el Entorno de Desarrollo Integrado NetBeans, la herramienta Visual Paradigm for UML y su Interfaz de Programación de Aplicaciones y la guía ofrecida por la metodología de desarrollo de software OpenUP.

PALABRAS CLAVE

anotaciones; base de datos; mapeo; objetos; orientado a objetos; relacional.

ÍNDICE GENERAL

AGRADECIMIENTOS I
DEDICATORIA II
RESUMEN III
INTRODUCCIÓN 1
CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA 5
1.1 Mapeo Objeto Relacional 5
1.2 Doctrine 7
1.3 Mapeo Objeto Relacional con Doctrine 9
1.4 Metodología de desarrollo de software 10
1.5 Modelos 12
1.6 Lenguajes de programación 14
1.7 Tecnologías y herramientas 16
1.7.1 Herramientas CASE 16
1.7.2 Entorno de Desarrollo Integrado 17
1.7.3 Herramienta a extender 18
1.7.4 Interfaz de Programación de Aplicaciones 22
1.8 Métodos de Expertos 22
1.9 Conclusiones 24
CAPÍTULO 2: ANÁLISIS Y DISEÑO DEL SISTEMA 25
2.1 Modelo de Dominio 25
2.2 Propuesta de solución 27
2.3 Especificación de los Requisitos del sistema 28
2.4 Modelo de Casos de Uso del Sistema 33
2.5 Modelo de diseño 39
2.6 Patrones utilizados 41
2.7 Diagrama de Secuencia 47
2.8 Conclusiones 48
CAPÍTULO 3: IMPLEMENTACIÓN Y PRUEBA 49
2.1 Modelo de implementación 49
2.2 Implementación de la extensión Cader 51
2.3 Estándares de codificación 56
2.4 Ejemplo de código 58
2.5 Interfaz de la extensión Cader 59
2.6 Prueba de software 59
2.7 Integración con Doctrine 2.0 62
2.8 Evaluación de la solución 64
3.9 Conclusiones 67
CONCLUSIONES 68
RECOMENDACIONES 69
REFERENCIAS BIBLIOGFRÁFICAS 70
BIBLIOGRAFÍA 71

ÍNDICE DE FIGURAS

Figura # 1: Estructura general de paquetes.....20
Figura # 2: Estructura de despliegue. 21
Figura # 3: Herramienta para el despliegue de la extensión. 22
Figura # 4: Diagrama del Modelo de Dominio.....26
Figura # 5: Proceso de generación de las CAD. 27
Figura # 6: Diagrama de Casos Uso del Sistema. 35
Figura # 7: Prototipo de interfaz de usuario “Generar CAD para Doctrine”. 39
Figura # 8: Diagrama de Clase del Diseño. 40
Figura # 9: Arquitectura por capa de la extensión. 43
Figura # 10: Ejemplo de método Singleton en la extensión. 45
Figura # 11: Ejemplo de método Iterator en la extensión.....46
Figura # 12: Diagrama de Secuencia. CU uso “Administrar clases de acceso a datos”. 47
Figura # 13: Diagrama de componentes de la extensión. 50
Figura # 14: Ejemplo de código de la extensión implementada. 58
Figura # 15: Interfaz de la extensión Cader.....59
Figura # 16: Resultados de pruebas. 61
Figura # 17: Caso de estudio “Cupón”.....62
Figura # 18: Creación del esquema de la base de datos. 63
Figura # 19: Entidades modeladas.....64
Figura # 20: Columnas de la entidad tienda. 64
Figura # 21: Relaciones de la entidad venta. 64
Figura # 22: Cuestionario aplicado a los expertos. 66

ÍNDICE DE TABLAS

Tabla # 1: Definición de los principales conceptos del Modelo de Dominio.....27
Tabla # 2: Descripción del actor del sistema identificado. 34
Tabla # 3: Descripción del CU “Administrar clases de acceso a datos”. 39
Tabla # 4: Descripción de las clases del diagrama de diseño. 40
Tabla # 5: Descripción de los componentes. 51
Tabla # 6: Descripción de los componentes utilizados en Visual Paradigm. 52
Tabla # 7: Descripción de las anotaciones utilizadas en Doctrine 2.0.....55
Tabla # 8: Resultados de los cuestionarios.....67

INTRODUCCIÓN

El uso de tecnologías de la información se ha convertido en un componente central de toda empresa o negocio que busque un crecimiento sostenido. Como resultado del uso de estas tecnologías se puede obtener y procesar información oportuna en la toma de decisión, a través de la utilización de sistemas de base de datos encargados de persistir los datos para un posterior uso.

Entre los diferentes tipos de base de datos que existe para dicha gestión se encuentran las de fichero, relacional, orientada a objetos e híbridas, donde la más común y extendida son las de tipo relacional. Esta última, posee características que favorece su utilización ya que evita la duplicidad de registros con el uso de campos claves o llaves, beneficia la normalización por ser más comprensible y garantiza la integridad referencial porque al borrar un registro elimina todos los registros relacionados dependientes.

Son varias las ventajas que presenta el uso de las bases de datos de tipo relacional, que han permitido una enmarcada utilización en el proceso de desarrollo de software. Este proceso se ha caracterizado por la variedad de formas y tipo de técnicas utilizadas para establecer el modo de diseño, desarrollo y mantenimiento del software. Sin embargo, para lograr una mejor representación de lo que se quiere modelar y dada la existencia de un mundo lleno de objetos, se sigue generalmente la técnica de Programación Orientada a Objeto (POO).

A pesar de los beneficios que aporta la técnica de POO en simplificar los datos complejos y agilizar el proceso de desarrollo de software, es obstaculizada cuando se usa el sistema de base de datos relacional. Esto induce a los desarrolladores a convertir los objetos con sus atributos del lenguaje de programación hacia registros de la base de datos, así como la herencia con las relaciones entre las tablas. De forma similar, deben realizar la implementación de la operación inversa por la diferencia lógica entre estos paradigmas que imposibilita la persistencia de forma directa de los datos.

El modo más utilizado para persistir un objeto en una base de datos relacional, consiste en que el desarrollador utilice llamadas manuales, a comandos de consultas y lo persista o lo construya a partir de los resultados, trayendo como consecuencia la dificultad de abstracción de los programadores para realizar la conversión.

Para facilitar este proceso se emplea la técnica de programación denominada Mapeo Objeto Relacional (ORM) que consiste en la generación de forma automática de las consultas a la base de datos para convertir los registros en objetos y viceversa. Permite a los programadores avanzar con mayor rapidez debido a que se concentrarán en codificar la lógica del negocio y no en realizar las consultas. Dicha técnica es utilizada por distintos lenguajes de programación como es en el caso de Java con Hibernate, en .NET se usa ADO.NET Entity Framework, y en PHP el más utilizado es Doctrine. Este último ORM es el más empleado por poseer una documentación que ha estado en constante crecimiento así como una comunidad activa y un bajo nivel de configuración para iniciar un proyecto. Estas características hacen que Doctrine sea incluido para la construcción de software por herramientas de Ingeniería de Software Asistida por Computadora (CASE).

Las herramientas CASE facilitan a las organizaciones automatizar y aumentar la productividad de los aspectos claves de todo el proceso de desarrollo de software. Entre estas herramienta se encuentra Visual Paradigm for UML (VP), que permite visualizar, diseñar e integrar diferente aplicaciones. Además de soporte de modelado, ofrece generación de informes, código de diagrama y su ingeniería inversa. Comprende el ciclo de vida completo del proceso de desarrollo del software y dentro de sus principales funcionalidades para la versión 7.2 incorpora la generación automática de la capa de acceso a datos para Doctrine 1.0.

La Universidad de las Ciencias Informáticas (UCI) tiene como objetivos informatizar el país y desarrollar la industria cubana del software con la realización de aplicaciones y servicios informáticos, a partir de la vinculación estudio-trabajo como modelo de formación. Está concebida en centros de desarrollo vinculados a las facultades con temáticas afines, entre los que se encuentra el Centro de Tecnologías de Gestión de Datos (DATEC). Este centro está dedicado a crear bienes y servicios informáticos relacionados con la gestión de datos, cuyo propósito fundamental es apoyar el proceso de toma de decisiones. Para alcanzar tales fines, se usan tecnologías para el desarrollo de software destinadas a aumentar la eficiencia y eficacia a través de las ventajas que incorporan, convirtiéndose en un eslabón importante para la calidad del software.

Específicamente en el Departamento de Soluciones Integrales perteneciente a DATEC, se utilizan para el desarrollo de aplicaciones web tecnologías tales como Symfony 2.0. Al incluir nuevas versiones trae consigo modernas características del framework, entre ellas se puede mencionar la utilización del ORM

Doctrine 2.0, impidiendo construir la capa de acceso a datos de forma automática a través de la herramienta VP, por ser incompatible con la versión (Doctrine 1.0) que soporta actualmente. La incompatibilidad entre las versiones implica que el proceso de obtención de la capa de acceso a datos se realice de forma manual por los desarrolladores. Esto trae como consecuencia que se cometan errores por las tareas repetitivas de conversión entre los distintos paradigmas y que aumente la complejidad de abstracción de los desarrolladores a la hora de trasformar el diseño del sistema incidiendo directamente en varios aspectos que afectan el desarrollo del proyecto. Entre ellos se encuentran el aumento del tiempo establecido por el equipo de desarrollo para cumplir con los objetivos del proyecto, el retraso de las entregas planificadas inicialmente con el cliente y el incremento de los costos concebidos por el departamento para sufragar los gastos.

Teniendo en cuenta lo anteriormente expuesto se propone como **problema de la investigación** ¿Cómo reducir el tiempo de desarrollo en la construcción de la capa de acceso a datos para Doctrine 2.0? por ello se hará uso del siguiente **objeto de estudio:** Mapeo de objetos relacionales y como **Campo de acción:** Mapeo de objetos relacionales con Doctrine 2.0.

En búsqueda de la solución al problema planteado se establece como **objetivo general:** Desarrollar la extensión de la herramienta Visual Paradigm para generar las clases de acceso a datos con Doctrine 2.0 a partir de un Diagrama Entidad Relación.

Objetivos Específicos:

- Analizar las características que presenta Doctrine 2.0 para el mapeo objeto-relacional.
- Realizar el análisis y diseño de la extensión de la herramienta Visual Paradigm para generar las clases de acceso a datos.
- Implementar la extensión de la herramienta Visual Paradigm para generar las clases de acceso a datos.
- Realizar las pruebas funcionales con el fin de garantizar los indicadores de calidad de la extensión desarrollada.

Tareas de la investigación:

1. Análisis de las características que soporta Doctrine 2.0 para el mapeo objeto-relacional.
2. Análisis de las especificaciones que presentan las clases generadas con Doctrine 2.0 para el mapeo objeto-relacional con las entidades de la base de datos.
3. Análisis de la metodología, herramientas y tecnologías para el desarrollo de la extensión.
4. Identificación y descripción de los requisitos con el fin de evidenciar el análisis de la solución.
5. Descripción de la arquitectura base de la extensión.
6. Definición de los componentes de diseño que se ajusten a la arquitectura de la herramienta Visual Paradigm for UML.
7. Implementación de los componentes de diseño.
8. Realización de las pruebas funcionales a la extensión para dar cumplimiento a los requisitos definidos.
9. Evaluación de la solución obtenida.

Posibles resultados:

Al finalizar esta investigación se espera obtener la extensión de la herramienta Visual Paradigm para la generación de clases de acceso a datos con Doctrine 2.0 a partir de un Diagrama Entidad Relación, que permita reducir el tiempo de producción de software en términos de implementación.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

En el presente capítulo se abordan los conceptos básicos que contribuyen a un mejor entendimiento del entorno de la extensión a implementar. Se estudia la definición del Mapeo Objeto Relacional y las características de Doctrine para analizar cómo se realiza el mapeo, en aras de entender el funcionamiento de la extensión y la elaboración de las clases de acceso a datos a través del Diagrama Entidad Relación. Se analiza la metodología, modelo de desarrollo, lenguaje de programación y de modelado así como las tecnologías y herramientas a utilizar, además del método propuesto por criterio de experto.

1.1 Mapeo Objeto Relacional

El Mapeo Objeto relacional (ORM - *Object Relational Mapping*) es la técnica de programación que permite transformar de forma automática los datos entre el lenguaje de programación orientado a objetos y el sistema de base de datos relacional, utilizando un motor de persistencia. Necesidad que surge debido a que prácticamente todas las aplicaciones están diseñadas para usar la Programación Orientación a Objetos (POO), mientras que las bases de datos más extendidas son del tipo relacional. Estas bases de datos solo permiten guardar tipos de datos primitivos (enteros, cadenas de texto) impidiendo guardar de forma directa los objetos de la aplicación en las tablas, sino que estos se deben de convertir antes en registros. En el momento de volver a recuperar los datos, hay que hacer el proceso inverso de convertir los registros en objetos. En la transformación se obtiene como resultado una base de datos orientada a objetos virtual sobre la base de datos relacional. Esto permite el uso de las características propias de la programación orientada a objetos (básicamente herencia y polimorfismo). Para desarrollar el mapeo relacional existen paquetes comerciales y de uso libre disponibles, aunque algunos programadores prefieren crear sus propias herramientas ORM.

Características

- Rapidez en el desarrollo: La mayoría de las herramientas permiten la creación del modelo por medio de la lectura del esquema de tabla y relación de la base de datos de forma automática. Esto evita el tiempo de codificación repetitiva por parte del programador y los errores que puede traer en la implementación.

- Abstracción del motor de base de datos: Permite separar totalmente del sistema de base de datos que se utilice, y si en el futuro se debe cambiar de motor de bases de datos, no afectará al sistema.
- Lenguaje propio para realizar las consultas: Estos sistemas de mapeo traen su propio lenguaje para hacer las consultas, lo que hace que los usuarios dejen de utilizar las sentencias de Lenguaje de Consulta Estructurado (SQL - *Structured Query Language*) por el lenguaje propio de cada herramienta.
- Interfaz simple: proporciona una interfaz más simple para el manejo de objetos a través de su propio lenguaje de consulta.

Ventajas

- Reutilización: Utiliza los métodos de un objeto desde distintas zonas de la aplicación como puede ser dentro o fuera de esta.
- Seguridad: Suelen implementar sistemas para evitar tipos de ataques como pueden ser las inyecciones de SQL.
- Mantenimiento del código: Facilita el mantenimiento del código debido al correcto orden de la capa de datos, haciendo que el mantenimiento del código sea mucho más sencillo.
- Encapsulación: Encapsula la lógica de los datos permitiendo hacer cambios que afectan a toda la aplicación únicamente modificando una función.

Existen distintos tipos de ORM en dependencia del lenguaje de programación que se desee utilizar para el desarrollo de determinado software. Específicamente en la construcción de aplicaciones web para la interpretación del lado del servidor se emplea el lenguaje de programación PHP. En este lenguaje los principales ORM que se suele recurrir para realizar el mapeo de los datos son Propel y Doctrine. Entre ellos dos ha resaltado más Doctrine por estar en un constante crecimiento en cuanto a documentación y miembros de su comunidad así como las características que presenta.

1.2 Doctrine

Doctrine es un ORM para PHP 5.3.0 o superior que proporciona persistencia transparente de los objetos desde el lenguaje PHP. Utiliza el patrón Data Mapper[1] en el núcleo del proyecto, logrando separar la lógica de dominio / negocio de la persistencia en un sistema de bases de datos relacional. Una de las principales características de Doctrine es su lenguaje SQL llamado Lenguaje de Consulta de Doctrine (DQL - *Doctrine Query Language*) inspirado en el Lenguaje de Consulta de Hibernate (HQL – *Hibernate Query Language*) y el bajo nivel de configuración que se necesita para comenzar un proyecto. Soporta las operaciones de Crear, Obtener, Actualizar y Borrar (CRUD - *Create, Retrieve, Update and Delete*) habituales, desde la creación de nuevos registros a la actualización de los antiguos. Crea manual y automáticamente el modelo de base de datos a implementar. Soportan varios motores de bases de datos (como MySQL, PostgreSQL, Microsoft SQL).

Doctrine 2 marca el comienzo de un nuevo enfoque ORM. Entre las ventajas que traen consigo cambios en el desacoplamiento de su lógica de negocio de la capa de persistencia y capacidad de prueba fácil de su modelo de dominio. Además de una reescritura de más de 90% de la base del código existente y las nuevas características que se desglosan de la manera siguiente. (1)

- DQL es ahora un lenguaje real dentro de Doctrine. Los beneficios de esta refactorización son mensajes de error de lectura, la generación de un AST (Árbol de sintaxis abstracta) que nos permite apoyar a diferentes proveedores y ganchos de gran alcance para los desarrolladores de modificar y ampliar el lenguaje DQL a sus necesidades. DQL puede ser escrito como una cadena o ser generado utilizando un objeto constructor de consulta potente.
- Sus objetos persistentes (llamadas entidades en Doctrine 2) no están obligados a extender más de una clase base abstracta. Doctrine 2 permite el uso de simples objetos de PHP.
- El UnitOfWork[2] es el patrón más céntrico de Doctrine 2. En lugar de llamar a guardar() o borrar() en Doctrine_Record[3] ahora se pasan los objetos al encargado de asignar los datos denominado

[1] Data Mapper: Es una capa del software que separa los objetos en memoria de la base de datos. Su función es transferir los datos entre estos dos y también para aislarlos unos de otros.

[2] UnitOfWork. Permite mantener todo lo que se desea actualizar, insertar y eliminar en un paquete antes de enviarlo a la base de datos.

EntityManager[4] y realiza un seguimiento de todos los cambios hasta que realice una sincronización entre bases de datos y los objetos actuales en la memoria. Esta es una mejora significativa sobre Doctrine 1 en términos de rendimiento y facilidad de uso.

- No hay medidas de generación de código de YAML a PHP que participan más en la biblioteca. YAML[5], XML[6], PHP y Anotaciones Docblock[7] son cuatro importantes entes para definir la asignación de metadatos entre los objetos y las bases de datos. Una capa de almacenamiento en caché de gran alcance permite a Doctrine 2 utilizar los metadatos en tiempo de ejecución sin depender de la generación de código.
- Presenta una mejor arquitectura y potentes algoritmos que logran realizar un funcionamiento más rápido en comparación con la versión anterior.
- Soporta una API que permite transformar una sentencia SQL arbitraria en un objeto-estructura.
- La herencia no es un problema en la actualidad porque hay tres tipos diferentes para elegir. (2)

Disímiles son las características y funcionalidades nuevas incorporadas a Doctrine en su versión 2 que motivan a los programadores a tener en cuenta estas técnicas para el desarrollo de aplicaciones web en conjunto con otras herramientas que facilitan y contribuyen a dicho desarrollo. A estos aspectos no está exento el Departamento de Soluciones Integrales del centro de DATEC, donde se utilizan tecnologías como el framework Symfony 2.0 y la nueva versión del ORM Doctrine 2.0. Esto constituye una perfecta unión que permite a los desarrolladores del departamento, cumplir con sus objetivos al contar con los elementos necesarios para construir aplicaciones web mantenibles, logrando mejorar en gran medida la productividad por la rapidez y flexibilidad que se puede llegar a alcanzar en un corto período de tiempo.

[3] Doctrine_Record. Representa una entidad con sus propiedades (columnas) y nos facilita métodos como para insertar, actualizar o eliminar registros.

[4] EntityManager. Es el punto de acceso a la funcionalidad, para manejar la persistencia de todos los objetos utilizando.

[5] YAML es un lenguaje muy sencillo que permite describir los datos como en XML, pero con una sintaxis mucho más sencilla.

[6] XML no es un lenguaje en particular sino que sirve de marco para definir lenguajes para diferentes necesidades. Algunos ejemplos son XHTML, XSLT, SOAP.

[7] Docblock. Es el nombre de las anotaciones realizadas por Doctrine para proporcionar metadatos siguiendo una sintaxis que está fuertemente inspirado en las anotación introducida con Java 5.

1.3 Mapeo Objeto Relacional con Doctrine

Doctrine proporciona varias formas para realizar el mapeo objeto relacional a través de asignación básica de objetos y propiedades. La especificación de estas asignaciones de datos se realiza con el uso de archivos XML, YAML o Anotaciones Docblock. Todos ellos tienen exactamente el mismo rendimiento en cuanto a su empleo, quedando a elección del programador cuál seleccionar.

En el desarrollo de la extensión se propone el uso de las Anotaciones Docblock por incluirse como forma de comentarios en archivos de PHP. A diferencia de los comentarios normales, las anotaciones pueden influir en la ejecución del código y propiciar una mayor flexibilidad para modificar el comportamiento de la aplicación desde los archivos en PHP. El uso de estos archivos para la generación de los metadatos, es por ser Doctrine 2 el ORM de Symfony, un framework realizado con PHP.

Las Anotaciones Docblock son herramientas para incrustar metadatos dentro de la sección de documentación que luego, alguna herramienta puede procesar. Doctrine 2 generaliza el concepto de anotaciones para que se puedan utilizar en cualquier tipo de metadatos y así facilitar la definición de nuevas Anotaciones Docblock. A fin de implicar más los valores de anotación y para reducir las posibilidades de enfrentamiento entre estos, Doctrine 2 presenta una sintaxis alternativa que está fuertemente inspirada en la sintaxis de las anotaciones introducidas en Java 5. (3)

Las anotaciones son utilizadas con el objetivo de representar la estructura de las clases de acceso a datos (CAD) para Doctrine, permitiendo la realización del mapeo objeto relacional que consiste básicamente en el trabajo con entidades o clases persistentes que representan las tablas con sus campos o columnas. Para la representación de una entidad se utilizan anotaciones específicas que permiten la definición, identificadores, asignación de los tipos de datos, asociación o relación entre entidades con cardinalidad y direccionalidad que pueden ser de tipo unidireccional o bidireccional. Estos son algunos de los elementos necesarios para la conformación de las entidades o clases de acceso a datos, siendo una característica que presenta la mayoría de los ORM que consiste en crearlas con procedimientos distintos. Entre las formas que se pueden generar estas entidades se encuentran:

- Manual: Puede ser desde la implementación directamente de las entidades o con el uso de un generador de entidades invocado por comandos de consola. La generación invocada por comandos de consola consiste en un grupo de preguntas que se realizarán para conformar las

entidades en cuanto a nombre, formato del archivo a obtener, columna de la tabla y por cada columna el tipo de dato entre otras propiedades de los datos.

- Automático: Se realiza con el uso de esquema de la base de datos. Primeramente se crea el esquema de la base de datos y la herramienta automáticamente mapeará las tablas y relaciones a través de las instrucciones de comandos indicados por consola para obtener cada entidad.

Con la implementación de la extensión se pretende obtener las entidades de una nueva forma que no incluye ninguna de las vías mencionadas anteriormente. La extensión permitirá una funcionalidad a la herramienta de VP para realizar el mapeo directamente desde su interfaz. Esta funcionalidad consistirá en la obtención de la información necesaria sobre la base de datos modelada a través del Diagrama o Modelo de Entidad - Relación (DER o MER) para la implementación de las entidades.

A partir de este tipo de diagrama se podrán modelar los elementos fundamentales para las clases que representarán cada tabla de la base de datos, basado en la percepción del mundo real que consta de una colección de objetos y relaciones entre ellos. Estos diagramas son muy comunes por la calidad que se puede alcanzar, debido a estar tutelado por la metodología de diseño de base de datos que garantiza una instrumentación de la forma más eficiente posible.

Luego de haber asignado los metadatos a las entidades obtenidas del diagrama de entidad relación, se genera el esquema de la base de datos relacional que utilizará Doctrine 2.0. Inmediatamente se puede empezar a realizar operaciones de adición o consulta de la información, en dependencia de la necesidad presentada en un momento determinado por el programador. Estas operaciones son algunas de las permitidas por Doctrine como resultado del mapeo del modelo de la base de datos, aspecto clave de su funcionamiento y tema a profundizar durante esta investigación.

1.4 Metodología de desarrollo de software

El uso de las metodologías de desarrollo de software es un punto importante para cualquier proyecto que busca minimizar riesgos e incrementar las posibilidades de éxito en el desarrollo de productos informáticos. Muestra el camino a seguir para la obtención de un producto predecible y eficiente, que posea calidad a pesar de su tamaño y complejidad. Especifica quién debe hacer qué, cuándo y cómo, a través de la estructuración, planificación y control del proceso de desarrollo.

En la actualidad no existe una metodología universal que se le pueda aplicar a todos los proyectos y su selección depende de las características que posea cada proyecto. El proceso de elección se realiza entre dos grandes grupos representadas por las tradicionales y ágiles. Las metodologías tradicionales están pensadas para el uso exhaustivo de documentación durante todo el ciclo del proyecto. Estas han demostrado ser efectivas y necesarias en un gran número de proyectos, sobre todo aquellos de gran tamaño (respecto a tiempo y recursos). Algunos ejemplos de metodologías tradicionales son: RUP (*Rational Unified Process*) y MSF (*Microsoft Solution Framework*).

En cambio, las metodologías ágiles centran su atención en la capacidad de respuesta a los cambios, la confianza en las habilidades del equipo y a mantener una buena relación con el cliente. Son menos orientadas al documento, exigiendo una cantidad más pequeña de documentación para una tarea dada y dedicada mayormente a la implementación del producto. Ejemplos de metodologías ágiles son: XP (*eXtreme Programming*), AUP (*Agil Unified Process*), Scrum y OpenUP (*Open Unified Process*).

A partir del análisis entre los grupos de metodologías existentes y las siguientes características identificadas, se selecciona para el proceso de desarrollo de software la metodología de desarrollo ágil OpenUP:

- Es apropiado para proyectos pequeños
- Presenta bajos recursos permitiendo disminuir las probabilidades de fracaso e incrementar las probabilidades de éxito.
- Evita la elaboración de documentación, diagramas e iteraciones innecesarios requeridos en la metodología RUP.
- Posee un enfoque centrado al cliente y con iteraciones cortas.

Proceso Unificado Abierto

Proceso Unificado Abierto (OpenUp) es una metodología de desarrollo basada en la metodología de Proceso Unificado Relacional (RUP). Es un subconjunto de esta última que contiene el conjunto mínimo de prácticas que ayuden a un equipo de desarrollo de software a realizar un producto de alta calidad y de una forma más eficiente. OpenUp utiliza un punto de vista pragmático y una filosofía ágil que se centraliza en la naturaleza colaborativa del proceso de desarrollo del software. Una de sus principales características es su alto grado de adaptabilidad a las necesidades de un proyecto en particular. Intenta incluir dentro de

su proceso de desarrollo únicamente el contenido imprescindible para garantizar un proceso de desarrollo de calidad y eficiente. (4)

OpenUp tiene las características de un Proceso Unificado al cual se aplican enfoques iterativos e incrementales que permite detectar errores tempranos a través de un ciclo estructurado en cuatro fases: concepción, elaboración, construcción y transición. El proceso de desarrollo en OpenUp utiliza casos de uso, escenarios, gestión del riesgo y un enfoque centrado en la arquitectura. (4)

1.5 Modelos

El modelo, en el proceso de desarrollo del software, es el artefacto más utilizado por los trabajadores para mostrar las perspectivas diferentes y necesarias del sistema, por tanto, la construcción de un sistema es un proceso de construcción de modelos. La utilización de modelos para diseñar software es una práctica bien establecida, sin embargo, en la mayoría de los casos los modelos se utilizan sólo como borradores que comunican de manera informal algún aspecto del sistema, o como planos arquitectónicos que describen un diseño que posteriormente se implementará. Esta práctica de utilizar modelos como documentación y especificación es valiosa, pero requiere de una estructura o lenguaje que proporcione un estándar durante la realización del modelado.

Lenguaje Unificado de Modelado

Lenguaje Unificado de Modelado (UML- *Unified Modeling Language*), es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad siendo respaldado por el Grupo Manejador de Objetos (OMG - *Object Management Group*). Es una consolidación de muchas de las notaciones y conceptos más usados en la metodología orientada a objetos. Ofrece un estándar para describir un "plano" del sistema (modelo), incluyendo aspectos conceptuales tales como procesos de negocios, funciones del sistema, aspectos concretos como expresiones de lenguajes de programación, componentes de software reutilizables y esquemas de bases de datos.

Permite especificar y visualizar un sistema aunque no admite la descripción de métodos o procesos. Se utiliza para definir un sistema de software, detallar los artefactos, especificar su documentación y construcción. Se puede aplicar en una gran variedad de formas para dar soporte a una metodología de desarrollo de software, contando con varios tipos de diagramas para llegar a representar los diferentes

puntos de vistas de un sistema. Esto es posible por la definición de UML como un lenguaje de modelado de propósito general que puede ser utilizado para representar el punto de partida de otros modelos de desarrollo, dentro de los que se puede mencionar el Desarrollo Dirigido por Modelo.

Desarrollo Dirigido por Modelo

El Desarrollo Dirigido por Modelo (MDD - *Model-Driven Development*) es un nuevo paradigma que promete mejorar la construcción de software apoyado en procesos dirigidos por modelos. Constituye una aproximación para el desarrollo de sistemas basado en la separación entre la especificación de la estructura, funcionalidades esenciales del sistema y la implementación final. La idea fundamental de MDD es sustituir al código de lenguajes de programación específicos por modelos. De este modo y en el contexto de este paradigma, los modelos permiten nuevas posibilidades de crear, analizar y manipular sistemas a través de diversos tipos de herramientas y de lenguajes.

En el desarrollo de este paradigma los modelos no son utilizados solamente como borradores, diagramas o planos, sino como artefactos primarios, de los cuales se genera automáticamente la implementación. En MDD, los modelos orientados al dominio de la aplicación son el ente principal cuando se desarrollan nuevos componentes de software. El código y otros artefactos se generan utilizando transformaciones diseñadas con entradas provistas tanto por expertos en modelado, como por expertos del dominio.

La utilización de los modelos en MDD es para dirigir las tareas de comprensión, diseño, construcción, pruebas, despliegue, operación, administración, mantenimiento y modificación de los sistemas. Estos modelos facilitan la especificación completa y precisa de la información conceptual, permitiendo la posterior generación, que de otra manera necesitaría ser generado manualmente por los desarrolladores. El proceso de generación automatizado constituye un paso importante para la producción del software por permitir su realización de forma ágil y fácil.

MDD tiene el potencial para producir grandes beneficios como incremento en la productividad, repetitividad, sistemas más adaptables, mejor comunicación y captura del conocimiento, sin embargo, la estrategia debe aplicarse correctamente para asegurar los resultados positivos que se desean, los cuales en la realización de la extensión serán obtenidos por medio del diseño del Diagrama Entidad Relación modelado con la herramienta Visual Paradigm.

1.6 Lenguajes de programación

El lenguaje de programación es el idioma artificial utilizado para expresar procesos que pueden ser llevadas a cabo por máquinas como las computadoras. Permite especificar de manera precisa sobre qué datos debe operar una computadora, cómo estos datos deben ser almacenados o transmitidos y qué acciones debe tomar bajo otras circunstancias. Está compuesto por un conjunto de símbolos, reglas sintácticas y semánticas que definen su estructura y el significado de sus elementos y expresiones. Son utilizados para controlar el comportamiento físico y lógico de una máquina a través de la creación de programas, logrando expresar algoritmos con precisión, o como modo de comunicación humana. (5)

Los lenguajes de programación se pueden clasificar en diferentes tipos como los de máquina, ensamblador, compilado, interpretado, declarativo, imperativo y orientado a objeto. Estos se fueron desarrollando a medida que la complejidad de las tareas que realizaban las computadoras aumentaban, donde pasaron de instrucciones que ejercían el control directo sobre el hardware a algoritmos en los que se podía expresar de una manera adecuada a la capacidad cognitiva humana. Entre esta última clasificación se encuentra el lenguaje de programación orientado a objeto, siendo el más utilizado por los desarrolladores de software en su diseño, debido a la representación y trabajo con los objetos.

Programación orientada a objetos

La Programación Orientada a Objetos (POO) es un paradigma de programación que utiliza los objetos e interacciones para diseñar aplicaciones y programas informáticos. Está basado en varias técnicas, incluyendo herencia, abstracción, polimorfismo y encapsulamiento. (6)

Representa el mundo real tan fielmente como sea posible a través de las Clases, Objeto, Métodos, Eventos, Estados, Atributos, y componentes de objetos. Entre algunas de las ventajas que se pueden obtener son:

- Agiliza el desarrollo del software
- Facilita la creación de programas visuales.
- Fomenta la reutilización y extensión del código.
- Facilita el mantenimiento del software.

- Proporciona conceptos y herramientas para modelar. (7)

Contiene una amplia variedad de librerías con funciones estándar (esto hace que no sea necesario re-inventar “la rueda” para empezar a desarrollar en ella), infinidad de API’s para convivir con otras aplicaciones, una comunidad de desarrolladores muy grande además de ser una tecnología gratuita, segura y de gran demanda en el mercado.

Java

Entre los lenguajes de POO que existen en la actualidad se encuentran Ada, C, C++, C#, PHP y otros donde el más utilizado es Java. Este lenguaje de programación orientado a objetos fue desarrollado por Sun Microsystems en la década del noventa. La intención era crear un lenguaje con una estructura y una sintaxis similar a C y C++, aunque con un modelo de objetos más simple y eliminando las herramientas de bajo nivel. Uno de sus pilares es la posibilidad de ejecutar un mismo programa en diversos sistemas operativos (independiente de la plataforma).Su diseño presenta como características ser robusto, seguro, portable, independiente a la arquitectura, dinámico e interpretado. En la implementación de la extensión es utilizado Java por ser el lenguaje que permite tener acceso a las funcionalidades administradas por la Interfaz de Programación de Aplicaciones (Open API).

PHP

El acrónimo recursivo que significa PHP (*Hypertext Pre-processor*) es un lenguaje interpretado de propósito general ampliamente usado, diseñado especialmente para desarrollo web y que puede ser introducido dentro de código HTML. Generalmente se ejecuta en un servidor web, tomando el código en PHP como su entrada y creando páginas web como salida. Puede ser desplegado en la mayoría de los servidores web y en casi todos los sistemas operativos y plataformas sin costo alguno. (8)

Es una alternativa de fácil acceso debido a que es libre y permite aplicar técnicas de programación orientada a objetos. Está completamente orientado al desarrollo de aplicaciones web dinámicas con acceso a información almacenada en una base de datos. Su programación es segura y confiable porque al escribir el código fuente en PHP, se hace invisible al navegador y al cliente, debido a que el servidor es el encargado de ejecutar el código y enviar su resultado HTML al navegador.

Este lenguaje, es empleado durante el desarrollo de la extensión para la confección de las clases necesarias en el trabajo con la capa de acceso a datos del ORM. Las clases que se pretenden generar con el uso de las Anotaciones Docblock, se obtendrán como resultado de la generación de forma automática por la extensión a desarrollar.

1.7 Tecnologías y herramientas

Las tecnologías de información y las herramientas que soportan el desarrollo de una aplicación, juegan un papel clave en su evolución. Representan el apoyo a la administración del proyecto, adaptándose a las características y objetivos propios del equipo de desarrollo. Seguido de la necesidad de realizar un software con la calidad requerida, se realizó un análisis de las tecnologías y herramientas necesarias para la implementación del componente.

1.7.1 Herramientas CASE

Las herramientas de Ingeniería de Software Asistida por Ordenador (CASE - *Computer Aided Software Engineering*) son diversas aplicaciones informáticas destinadas a aumentar la productividad en el desarrollo de software, reduciendo el costo de los mismos, en términos de tiempo y presupuesto. Estas herramientas ayudan en todos los aspectos del ciclo de vida de desarrollo del software, en tareas como: el proceso de realizar un diseño del proyecto, cálculo de costes, implementación automática de parte del código con el diseño dado, compilación automática, documentación o detección de errores, entre otras. (9)

Entre sus principales objetivos se encuentran:

- Mejorar la productividad en el desarrollo y mantenimiento del software.
- Aumentar la calidad del software.
- Mejorar el tiempo y coste de desarrollo y mantenimiento de los sistemas informáticos.
- Mejorar la planificación de un proyecto.
- Ayudar a la reutilización del software, portabilidad y estandarización de la documentación.
- Gestionar las fases de desarrollo del software con una misma herramienta.

Para la generación de los artefactos necesarios en el ciclo de vida del proceso de desarrollo de la extensión, se seleccionará la herramienta CASE de Visual Paradigm for UML por las características que a continuación se mencionan.

Visual Paradigm for UML

Visual Paradigm for UML (VP) es una herramienta que soporta el ciclo de vida completo en el desarrollo de software: análisis y desarrollos orientados a objetos, construcción, prueba y despliegue. Permite diseñar todo tipo de diagrama de clases, código inverso, generación de código a partir de diagramas y generar documentación. (10) Entre sus características fundamentales se tiene:

- Multiplataforma: Soportada en plataforma Java para Sistemas Operativos Windows, Linux, Mac OS.
- Interoperabilidad: Intercambia diagramas UML y modelos con otras herramientas. Soporta la Importación y Exportación a formatos XMI y XML y archivos Excel. Permite importar proyectos de Rational Rose y la integración con Microsoft Office Visio.
- Modelado de Requisitos: Captura de requisitos mediante diagramas de requisitos, modelamiento de caso de uso y análisis textual.
- 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. (11)

1.7.2 Entorno de Desarrollo Integrado

El Entorno de Desarrollo Integrado (IDE - *Integrated Development Environment*) es un entorno de programación que ha sido empaquetado como un programa de aplicación que brinda facilidades y herramientas para los desarrolladores. Consiste en un editor de código, compilador, depurador y constructor de interfaz gráfica de usuario. Los IDEs pueden ser aplicaciones por sí solas o parte de

aplicaciones existentes. Entre sus componentes están el editor de texto, intérprete, herramientas de automatización y posibilidad de ofrecer un sistema de control de versiones. Como características se encuentran el soporte de diversos lenguajes de programación, reconocimiento de sintaxis, multiplataforma, integración con framework, múltiples idiomas, manual de usuarios y ayuda. Para el desarrollo de la extensión se propone el uso del IDE NetBeans, por ser uno de los más utilizados y por las diferentes características que a continuación se describen.

NetBeans

El IDE NetBeans 7.1 es una herramienta para programadores de código abierto escrito completamente en Java pero puede servir para cualquier otro lenguaje de programación. Tiene gran éxito con una gran base de usuarios y una comunidad en constante crecimiento. Está compuesta por una base modular y extensible usada como una estructura de integración para crear aplicaciones de escritorio grandes desde los sistemas operativos tales como Windows, Mac, Linux, y Solaris. Ofrece servicios comunes a las aplicaciones de escritorio, permitiéndole al desarrollador enfocarse en la lógica específica de su aplicación. Entre sus características están las administraciones de ventanas, almacenamiento, interfaces y configuraciones de usuario. Soporta el desarrollo de aplicación Java (J2SE, web, EJB y aplicaciones móviles), empresariales con Java EE 5, JavaFX 2.0, WebLogic 12c y CSS3.Incluye herramientas de desarrollo visuales de SOA, herramientas de esquemas XML, orientación a servicios web (para BPEL) y modelado UML. El NetBeans C/C++ Pack soporta proyectos de C/C++, mientras el PHP Pack, soporta PHP 5.Empresas independientes asociadas, especializadas en desarrollo de software, proporcionan extensiones adicionales que se integran fácilmente en la plataforma y que pueden también utilizarse para desarrollar sus propias herramientas y soluciones.

1.7.3 Herramienta a extender

La implementación de extensiones para aplicaciones constituye un mecanismo para la incorporación de funcionalidades que la aplicación no provee a los usuarios. Por lo general se asocian las extensiones con plugin, que son fragmentos de software que interactúan con el núcleo de la aplicación para proporcionar algunas funcionalidades que en la mayoría de los casos son muy específicos. Visual Paradigm es una de las herramientas CASE que posee gran prestigio para el modelado de software. (11)

Esta herramienta se caracteriza por la constante incorporación de numerosas funcionalidades en cada una de sus versiones que se desarrollan, pero desde la versión 7.2 de enero del 2010, no se ha realizado cambio referente a la generación de las clases de acceso a datos para el ORM Doctrine. Esto ha traído como consecuencia que se deje de utilizar la funcionalidad por ser incompatible con la nueva versión 2.0 de Doctrine promocionada al mercado en diciembre del 2010. Desde entonces VP ha realizado numerosas versiones de su herramienta de modelado hasta llegar a la 10.0 en que se encuentra, sin ningún cambio hacia la generación de las clases de acceso a datos con Doctrine.

Sin embargo VP cuenta con los medios necesarios para extender funcionalidades, dando soporte a las extensiones de aplicación. La aplicación provee de forma libre una interfaz de programación permitiendo a los desarrolladores implementar y reutilizar clases e interfaces, desarrollando funciones agregadas que son útiles para el desarrollo de software. (11)

Para poder desarrollar la extensión en VP es necesario conocer sobre cómo interactuar con la librería openapi.jar proporcionada por la herramienta CASE y cómo lograr la unión de la extensión creada. Este aspecto es el propósito fundamental que recogen los dos temas siguientes: Estructura de desarrollo y la Integración con la herramienta.

Estructura de desarrollo

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

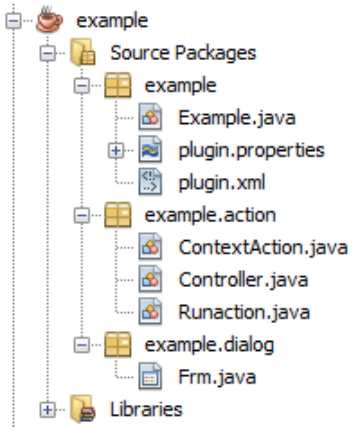


Figura # 1: Estructura general de paquetes.

Paquete example:

La clase plugin.xml permite por medio de un script XML la configuración de la extensión. Entre los elementos que se establecen en esta clase se encuentra el tipo de controlador de acción que se desea utilizar, que puede ser: actionSet o contextSensitiveActionSet. Estos permiten definir si se realizarán desde el contexto del diagrama en que se trabaje y/o dada una opción de la barra de menú determinada por una ruta de acceso previamente establecida.

La clase plugin.java es la encargada de la carga y descarga de las propiedades de la extensión configuradas en la clase plugin.xml. Esto es concebido por la implementación en la clase plugin.java de la interfaz VPPlugin, que trae consigo los métodos load() y unload() permitiendo poder realizar dichas operaciones.

Paquete example.actions:

Contiene las acciones de la extensión definidas a nivel de herramientas y de contexto. En dependencia de la acción realizada en la clase plugin.xml, se implementan las interfaces asociadas a las acciones VPActionController para la clase RunAction y VPContextAction para la clase ContexAction. Ambas clases definen el performAction el cual permite ejecutar acciones referentes al evento onClick de la acción. Pueden ser incluidas otras clases que contribuyan al aumento de las acciones que se realicen en la extensión, como es el caso de la clase Controller.

Paquete example.dialog:

Se encuentran los diálogos o interfaz que se desea mostrar, para que el usuario pueda interactuar y percibir los beneficios agregados a la herramienta con la extensión. La creación de la interfaz se realiza mediante el método performAction() asociado a la clase de acción correspondiente al dialogo.

Integración con la herramienta

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

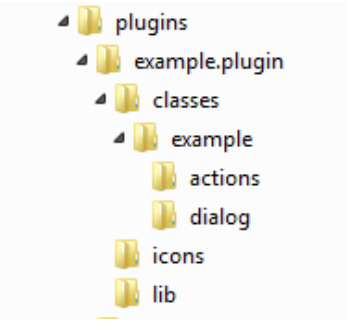


Figura # 2: Estructura de despliegue.

Si se observa la estructura de implementación de la extensión, es diferente a la de integración con la herramienta, lo que dificulta el proceso de pruebas al integrar la extensión implementada para “Visual Paradigm for UML”. La solución a este inconveniente es utilizar una herramienta de despliegue de extensiones que facilite la integración del mismo con “Visual Paradigm for UML”, pasando los valores a la herramienta como es el nombre de la extensión, el directorio de origen de la implementación en el IDE y la dirección donde será desplegada la extensión. (11)

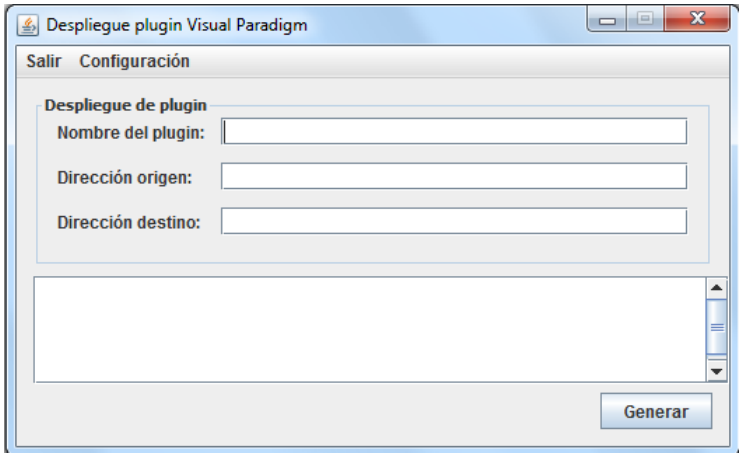


Figura # 3: Herramienta para el despliegue de la extensión.

1.7.4 Interfaz de Programación de Aplicaciones

Interfaz de Programación de Aplicaciones (API – *Application Programming Interface*) es una interfaz de comunicación entre componentes de software. Consiste en proporcionar un conjunto de funciones de uso general, que son llamadas a ciertas bibliotecas que ofrecen acceso a ciertos servicios desde los procesos y representa un método para conseguir abstracción en la programación.

Open API

Open API es un mecanismo de extensión proporcionada por Visual Paradigm para extender las funcionalidades del software de cliente. El API es basado en Java y permite tener acceso completo a los datos del modelo en el archivo de proyecto. Mediante el uso de este mecanismo se pueden implementar algunas funciones personalizadas llamadas plugins o extensiones para lograr determinados fines sobre la herramienta VP.

1.8 Métodos de Expertos

Cuando se realiza una investigación, uno de los principales inconvenientes que tiene asociado es la dificultad de verificar y demostrar la confiabilidad de la propuesta resultante. Ante esta situación se emplean los métodos de expertos para evaluar los resultados arrojados por dicha investigación. Estos métodos utilizan como fuente de información un grupo de personas que poseen cierto grado de

conocimiento de la materia en cuestión, permitiendo la realización de un análisis multicriterio. Como ventajas de los métodos de expertos se tiene que:

- El conjunto de opiniones de las personas que se consulten es siempre más valioso que la opinión individual de la persona mejor preparada, aunque esta última contraste con las anteriores.
- Un grupo de personas siempre tendrá en cuenta mayor número de factores para realizar la evaluación, que los considerados por una sola persona. (12)

Sin embargo, también presentan las siguientes desventajas:

Estos grupos son vulnerables a la posición y personalidad de algunos de los individuos. Una persona con habilidades de comunicador puede convencer al resto, aunque su opinión no sea la más acertada. Esta situación se puede dar también cuando uno de los expertos ocupe un alto cargo en la organización, ya que sus subordinados no le rebatirán sus argumentos con fuerza. (12)

Teniendo en cuenta las ventajas y desventajas que presentan los métodos expertos como Kendall, Scoring, Promethee y Delphi, lo ideal sería utilizar un método que extraiga los beneficios de su aplicación y elimine sus inconvenientes. Estas son las principales cualidades que sigue la filosofía del método Delphi, motivo por lo cual será utilizado para la evaluación del resultado de la investigación propuesta.

Método de evaluación de la solución

El método Delphi aprovecha la sinergia del debate en el grupo y se eliminan las interacciones sociales indeseables que existen dentro de todo grupo. De esta forma se espera obtener un consenso lo más fiable posible del grupo de expertos. (12)

Este consenso es posible, porque durante la realización del método ningún experto conoce la identidad de los otros que componen el grupo, lo cual aporta una serie de aspectos positivos, como son:

- Impide la posibilidad de que un miembro del grupo sea influenciado por la reputación de otro miembro, o por el peso que supone oponerse a la mayoría.
- La única influencia posible es la de la congruencia de los argumentos.
- Permite que un miembro pueda cambiar sus opiniones sin que eso suponga una pérdida de imagen.

- El experto puede defender sus argumentos con la tranquilidad que da saber que en caso de que sean erróneos, su equivocación no va a ser conocida por los otros expertos. (12)

Para la aplicación del método se suele seguir con las cuatro fases fundamentales siguientes:

- Fase 1: Formulación del problema: En esta fase se plantea el problema que enfrentaran los expertos.
- Fase 2: Selección de los expertos: El experto será elegido por poseer conocimientos sobre el tema consultado.
- Fase 3: Elaboración y envío de los cuestionarios: Los cuestionarios se elaborarán de manera que faciliten, en la medida en que una investigación de estas características lo permite, la respuesta por parte de los consultados.
- Fase 4: Análisis de los resultados: Luego de tener los resultados de los cuestionarios enviados a los expertos, se procede a realizar un análisis cuantitativo de los resultados obtenidos.

1.9 Conclusiones

Al concluir el capítulo se pudo realizar un estudio sobre los elementos fundamentales de composición y realización del mapeo, además de una descripción de las principales propiedades que posee Doctrine fundamentalmente en su versión 2.0. Se obtuvo como metodología de desarrollo ágil OpenUP así como la utilización del lenguaje modelado UML para la visualización, especificación y documentación de la extensión. Se analizaron las características de la herramienta a extender y la estructura de desarrollo a utilizar para la implementación extensión, además de la forma de integración propuesta por Visual Paradigm. Finalmente para demostrar la confiabilidad de la propuesta resultante se seleccionó el método Delphi para verificar el cumplimiento del problema planteado mediante las cuatro fases que implementa.

CAPÍTULO 2: ANÁLISIS Y DISEÑO DEL SISTEMA

En este capítulo se realiza la descripción de las principales definiciones asociadas al dominio del problema. También se especifican los requisitos funcionales y no funcionales que tendrán lugar en la implementación. Se identifican el actor y los casos de uso del sistema con los cuales interactúa, así como las relaciones entre ellos mediante el diagrama de casos de uso del sistema, además de las descripciones textuales de cada uno. Finalmente se proponen los elementos del diseño a emplear, además del diagrama de secuencia de la extensión.

2.1 Modelo de Dominio

Para proporcionar ayuda a los usuarios, desarrolladores e interesados a entender el contexto en que se ubica la extensión y la posterior representación de la solución, se recurre al empleo del modelo de dominio. Este ocupa un rol protagónico en el desarrollo de la extensión, por ser el ente de partida para el diseño de la extensión, al crear una representación visual del entorno real de la extensión.

El modelo de dominio es un artefacto de la disciplina de análisis, construido con las reglas de UML durante la fase de concepción. El modelo de dominio presenta uno o más diagramas de clases que no contienen conceptos propios de un sistema de software sino de la propia realidad física. Es un subconjunto del modelo de negocio y se realiza cuando no están claros los procesos o cuando no se identifican claramente los actores y trabajadores del negocio. Además, el modelo de dominio captura los tipos más importantes de objetos que existen, o los eventos que suceden en el entorno donde estará el sistema, se identifican conceptos, se definen estos conceptos y se unen o relacionan en un diagrama de clases UML. (13)

Diagrama conceptual del dominio

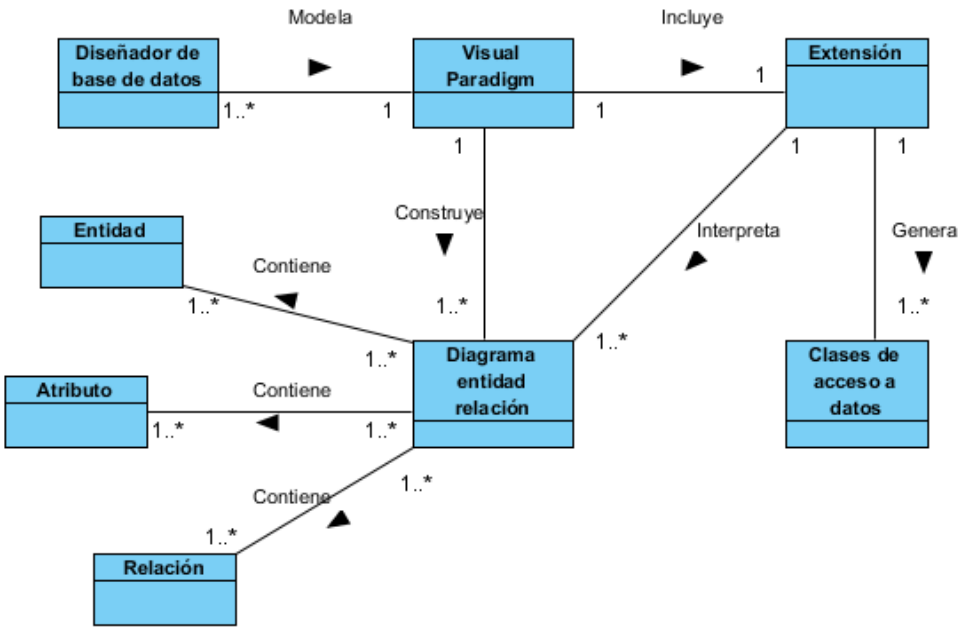


Figura # 4: Diagrama del Modelo de Dominio.

Definición de conceptos del Modelo de Dominio

Elementos	Definiciones
Diseñador de Base de Datos	Ente encargado de modelar los datos que se almacenarán en la base de datos. Para ello elige las estructuras o representación de la base de datos, sus relaciones de cardinalidad y herencia.
Visual Paradigm	Es una herramienta CASE de modelado para el proceso de desarrollo de software. Entre sus característica incluye la construcción del DER e incorporación de funcionalidades a través de extensiones.
Extensión	Extensión incluida a la herramienta Visual Paradigm. Contiene las funcionalidades que permiten interpretar la información de un DER, para la transformación del modelo y la generación de las clases de acceso a datos para el ORM Doctrine.
Diagrama	Permite modelar una percepción del mundo real, representado por

Entidad Relación	atributos, entidades y sus relaciones entre ellas. Es el punto de partida para la transformación del modelo.
Clases de acceso datos	Son generadas por la extensión para el ORM Doctrine, con el uso de las anotaciones Docblock en el lenguaje de programación de PHP. Se obtienen a través de la transformación del modelo realizado en el DER.

Tabla # 1: Definición de los principales conceptos del Modelo de Dominio.

2.2 Propuesta de solución

Se propone realizar la extensión Cader identificada por el logotipo que se muestra en la Figura # 5, es una extensión de la herramienta “Visual Paradigm for UML”, la cual estará orientada a la obtención de las clases de acceso a datos directamente desde su interfaz, pasando por un proceso compuesto por tres etapa para su desarrollo.

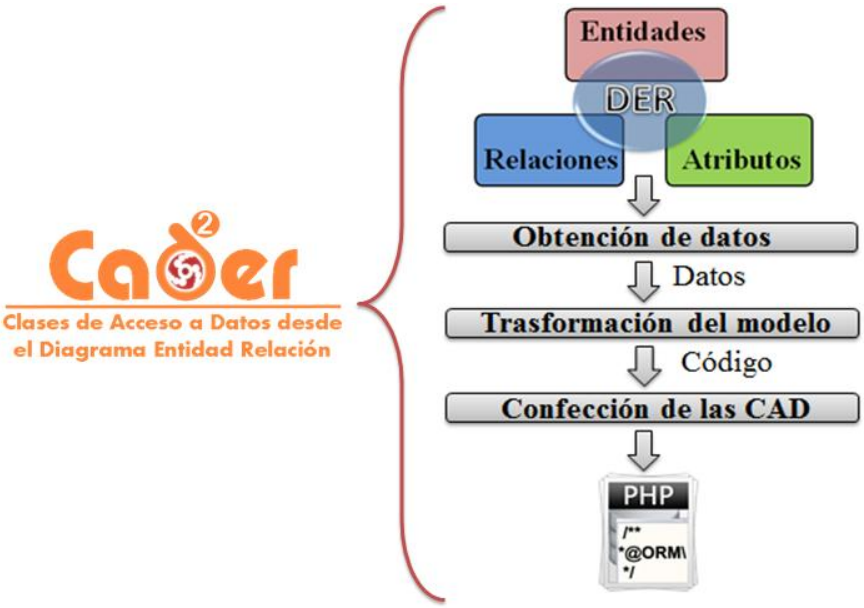


Figura # 5: Proceso de generación de las CAD.

Dichos procesos transcurren una vez realizado el diseño de la base de datos a través del Diagrama Entidad Relación para modelar los componentes necesarios para su comprensión. Seguidamente se procede a la obtención parcial o total de las tablas modeladas como requisito necesario para la recogida de los datos y así comenzar la etapa de transformación del modelo. En esta etapa se realiza una conversión entre las propiedades obtenidas de las entidades y la forma de su representación, que permita

ser interpretado por el ORM Doctrine para efectuar el mapeo de la base de datos. Esto incluye el uso de anotaciones Docblock en forma de comentario desde el lenguaje de programación de PHP, para poder especificar las características definidas con anterioridad de cada entidad o relaciones entre ellas. Luego de la transformación y según el directorio seleccionado por el desarrollador, se confeccionan las CAD como etapa final. La confección estará representada por un fichero llamado como "nombreEntidad.php" en correspondencia con las entidades seleccionadas inicialmente, además de las clases Repositorios[8] por entidad ("nombreEntidadRepository.php").

2.3 Especificación de los Requisitos del sistema

A partir de la descripción de las clases más importantes dentro del contexto de la extensión representado en el Modelo de Dominio, se realiza el levantamiento de requisitos. Esto proporciona a los desarrolladores una mejor comprensión de las funcionalidades y fronteras que posee la extensión. Establecen una base para la planificación del contenido técnico de desarrollo y definición de una interfaz para el usuario, enfocado en los requisitos a utilizar para la realización de las funcionalidades de la extensión. Para ello se muestra a continuación los requisitos funcionales y no funcionales que se encuentran plasmados en el documento de Especificación de requisitos de la extensión.

Requisitos funcionales

Los requisitos funcionales (RF) son capacidades o condiciones que el sistema debe cumplir. Definen las funciones que el sistema será capaz de realizar. Expresan la naturaleza del funcionamiento del sistema, cómo interacciona el sistema con su entorno y cuáles van a ser su estado y funcionamiento. (14) Entre los requisitos funcionales identificados para la realización extensión se encuentran:

RF1. Listar los Diagramas de Entidad Relación

Descripción: La extensión mostrará en un listado, los nombres de los Diagramas Entidad Relación que han sido modelado por el diseñador de base de datos en el Visual Paradigm.

[8] Estos archivos se utilizan para organizar sentencias DQL de una entidad en cuestión. Dentro de estas clases ubican cada uno de los métodos que serán las consultas a utilizar.

Entrada: Utiliza un objeto creado a partir del proyecto en que se trabaja en el Visual Paradigm, para así poder obtener todos los diagramas modelados y entre ellos los del Diagrama Entidad Relación.
Salida: Un listado de todos los Diagramas Entidad Relación que posea el proyecto en que se trabaja desde el Visual Paradigm.

RF2. Listar entidades del Diagrama Entidad Relación seleccionado.

Descripción: La extensión mostrará en un listado, todos los nombres de las entidades que posea el Diagrama Entidad Relación seleccionado por el desarrollador desde la interfaz visual.
Entrada: Un valor numérico que indica la posición en que se encuentra el Diagrama Entidad Relación seleccionado del listado mostrado por la extensión.
Salida: Un listado con todas las entidades que posee el Diagrama Entidad Relación seleccionado.

RF3. Adicionar entidades al listado de generación.

Descripción: Se adicionan al listado de entidades a generar, el nombre de las entidades que han sido seleccionadas por el desarrollador para su posterior generación.
Entrada: Valor numérico que indica el tipo (parcial o total) de agregación a establecer hacia el listado de entidades a generar.
Salida: Actualiza los cambios a ambos listados, luego de la operación (parcial o total) de agregación realizada.

RF4. Remover entidades del listado de generación.

Descripción: Se remueve del listado de entidades a generar las entidades que han sido seleccionadas por el diseñador, regresándolas hacia el listado de entidades por diagramas.
Entrada: Valor numérico que indica el tipo (parcial o total) de eliminación de las entidades en el listado de entidades a generar.
Salida: Actualiza los cambios en ambos listados, luego de la operación (parcial o total) de eliminación realizada.

RF5. Obtener dirección del directorio.

Descripción: Se selecciona por parte del desarrollador la dirección en donde se desee guardar las clases de acceso a datos a generar por la extensión.
Entrada: Selección realizada por el desarrollador desde una ventana que permite la navegación por los directorios para elegir así el lugar a guardar los archivos generados por la extensión Cader.
Salida: Valor booleano que permite identificar de forma correcta o incorrecta la selección del directorio realizada por parte del desarrollador.

RF6. Validar Diagrama Entidad Relación.

Descripción: Se comprueba el correcto diseño del Diagrama Entidad Relación en cuanto a tipo de dato usado en las columnas y la estrategia de generación de ID (identificador único a cada fila) según decisión del usuario.
Entrada: El listado de entidades a generar seleccionado previamente por el desarrollador.
Salida: Variable que identifica como valido o invalido el Diagrama Entidad Relación para realizar la generación de las clases de acceso a datos para Doctrine 2.0.

RF7. Generar código de declaración de entidad

Descripción: Se genera el código para la declaración de la entidad el cual incluye namespace[9] y nombre de la entidad según la estructura de formación de las clases de acceso a datos para el ORM Doctrine 2.0.
Entrada: Objeto de tipo entidad que se encuentra en el listado de entidades a generar confeccionado por el desarrollador desde la interfaz de la extensión.
Salida: Cadena de texto que guarda el código generado de declaración de entidad.

RF8. Generar código de los atributos por columna.

Descripción: Se genera el código que recoge las características de la columna en cuanto a nombre, tipo de dato, llave primaria, valores nulos y únicos.

[9] Un *namespace* o espacio de nombres es un medio para organizar clases dentro de un entorno, agrupándolas de un modo más lógico y jerárquico (22)

Entrada: Objeto de tipo entidad que se encuentra en el listado de entidades a generar y un objeto de tipo columna de los que posee el objeto entidad mencionado.
Salida: Cadena de texto que guarda el código generado de los atributos que posea el objeto de tipo columna.

RF9. Generar código para las relaciones de entidades.

Descripción: Se genera el código referente a las asociaciones existentes entre las entidades modeladas en el Diagrama Entidad Relación creado en el Visual Paradigm.
Entrada: Objeto de tipo entidad que se encuentra en el listado de entidades a generar confeccionado por el desarrollador.
Salida: Cadena de texto que guarda el código generado a partir de las relaciones que posea con otras entidades.

RF10. Generar código de los métodos Get y Set.

Descripción: Se generan los métodos de obtención (Get) y transformación (Set) de datos por cada columna que posea la entidad.
Entrada: Lista de objeto de tipo columna que posee la entidad a la cual se le generan los métodos Get y Set.
Salida: Cadena de texto que guarda el código generado de los métodos Get y Set por columna.

RF11. Generar código de clase Repositorio.

Descripción: Se genera el código referente a la clase repositorio por cada entidad que se encuentre en el listado de entidades a generar.
Entrada: El listado de entidades a generar seleccionado previamente por el desarrollador.
Salida: Lista que contendrá los códigos para la creación de las clases Repositorio posteriormente.

RF12. Generar las clases de acceso a datos.

Descripción: Se construyen los archivos que contendrán los códigos correspondientes a cada clase de acceso a datos guardada según el directorio seleccionado por el desarrollador desde la interfaz de la extensión.

Entrada: Lista que contendrá el listado de los nombres de las entidades y códigos correspondientes a cada entidad modelada en el Diagrama Entidad Relación.
Salida: Se obtienen los archivos con extensión en PHP que conforman las clases de acceso a datos.

Requisitos no funcionales

Los requisitos no funcionales (RNF) son propiedades o cualidades que el producto debe tener. Estas se refieren a las características que hacen al producto atractivo, usable, rápido o confiable. Por lo general los requisitos no funcionales son fundamentales en el éxito del producto; normalmente están vinculados a los requisitos funcionales, es decir, una vez que se conoce lo que el sistema debe hacer se puede determinar cómo ha de comportarse, qué cualidades o propiedades debe tener. (15) Entre los requisitos no funcionales necesarios para la realización de la extensión se encuentran:

Requisitos de Software

- El diseñador podrá realizar el modelo de la base de datos en La herramienta “Visual Paradigm for UML” 6.4 y deberá tener instalado el JRE (*Java Runtime Environment*) 6.0 o superior para que permita la ejecución de programas Java.

Requisitos de Hardware

- Se requiere para la ejecución de la extensión un procesador Intel Pentium III a 1.0 GHz o superior. Un mínimo 512 MB de RAM (*Random Access Memory*), y como mínimo de espacio libre en disco: 1GB (Gigabyte).

Restricciones del diseño e implementación

- Se deben utilizar las herramienta “Visual Paradigm for UML” 6.4 y el IDE NetBeans 7.1.
- El lenguaje de programación que será usado para la implementación es Java y para la confección de las clases de acceso a datos el de PHP con el uso de las anotaciones Docblock.

Requisitos de Usabilidad

- Facilidad de uso por parte de los usuarios: La interfaz debe ser lo más descriptiva posible, permitiendo que las operaciones a realizar por los usuarios estén bien descritas, de manera que se puedan entender claramente.
- La interfaz debe tener mensajes contextuales asociados a los objetos.

Requisitos de Portabilidad

- La extensión una vez integrada a la herramienta “Visual Paradigm for UML” podrá ser instalada y disponer de la misma, en diferentes sistemas operativos por ser “Visual Paradigm for UML” una herramienta multiplataforma.

Requisitos de Licencia

- No se posee ningún requisito de licencia o restricción en el uso del software puesto que se desarrolló con la utilización de herramientas libres como: Visual Paradigm 6.4 y Netbeans 7.1.

2.4 Modelo de Casos de Uso del Sistema

Concluido el proceso de levantamiento de requisitos del sistema declarados anteriormente, se propone a continuación la realización del modelo de casos de uso del sistema. Un modelo de casos de uso describe la funcionalidad propuesta del nuevo sistema, representa una unidad discreta de interacción entre un usuario (humano o máquina) y el sistema. (16) Este modelo permite a partir de la captura de los requisitos por casos de uso, simplificar la construcción del modelo de objetos y servir de base a la implementación de la extensión, por representar las relaciones existentes entre actores y casos de uso. Para un mejor entendimiento sobre los elementos que los componen, se realiza la siguiente descripción de ellos:

Casos de Uso del Sistema

Cada caso de uso tiene una descripción que especifica la funcionalidad que se incorporará al sistema propuesto. (16) Estas descripciones se encuentran recogidas por tabla, en la cual incluye aspectos como el nombre del CU descrito y un resumen del funcionamiento del CU, entre otras exigencias para la realización de sus operaciones. Las descripciones se encuentran representadas por las acciones del actor

al interactuar con el sistema y las respuestas que este le proporciona, a través de los distintos escenarios que componen el CU.

Un caso de uso puede “incluir” la funcionalidad de otro caso de uso o puede “extender” a otro, con su propio comportamiento. (16) Particularmente en el diagrama de casos de uso de la extensión se utiliza la relación de inclusión (Include) agregando determinadas funcionalidades o comportamiento al CU base. La representación se realiza desde el CU base a uno de inclusión, que especifica cómo el comportamiento definido para el CU de inclusión se inserta explícitamente dentro del comportamiento definido para el CU base.

Actor del sistema

Un actor es un usuario del sistema, que usa un caso de uso para desempeñar alguna porción de trabajo que es de valor para el negocio. El conjunto de casos de uso al que un actor tiene acceso define su rol global en el sistema y el alcance de su acción. Además son generalmente responsables de realizar actividades que serán automatizadas en el futuro sistema. (17)

Actor del Sistema identificado	Definición	Descripción
Desarrollador de software	Desarrollador de software es un programador que se dedica a una o más tareas del proceso de desarrollo de software, realiza programas o aplicaciones en uno o varios lenguajes de programación.	Es el encargado de generar las CAD por conocer las clases necesarias para el desarrollo del software y el área de trabajo en donde se realiza la implementación.

Tabla # 2: Descripción del actor del sistema identificado.

Diagrama de Casos de Uso del Sistema

A partir de los requisitos funcionales identificados con anterioridad, estos son agrupados según el siguiente diagrama de casos de uso de la extensión:

Figura # 6: Diagrama de Casos Uso del Sistema.

Descripción textual del caso de uso del sistema

Caso de Uso:	Administrar clases de acceso a datos.
Actor:	Desarrollador.
Resumen:	El caso uso se inicia cuando el actor seleccione la opción de "Generar las CAD para Doctrine 2.0". El sistema muestra el listado de los Diagrama Entidad Relación modelados en la herramienta Visual Paradigm, permitiendo seleccionar las entidades y las opciones de generación, además de una dirección para guardar los archivos que conforman las clases de acceso a datos creadas inmediatamente después de haber seleccionado la opción de "Generar".
Precondiciones:	Debe existir al menos un Diagrama Entidad Relación modelado en el Visual Paradigm.
Referencias	RF1, RF2, RF3, RF4, RF5, RF6, RF12 y CUS Generar Código <<incluido>>
Prioridad	Crítico
Flujo Básico de Eventos	
Acción del Actor	**Respuesta del Sistema**
1. El desarrollador selecciona la opción "Generar CAD para Doctrine 2.0" que se encuentra en el menú Tool/Object Relational Mapping (ORM) de Visual Paradigm.	2. El sistema muestra la interfaz de la extensión.
Sección 1: "Listar Diagrama Entidad Relación"	
Acción del Actor	**Respuesta del Sistema**
	1. El sistema muestra un listado con los Diagrama Entidad Relación creados en VP.

Flujos Alternos al paso 1 "Ausencia de Diagrama Entidad Relación en el Visual Paradigm"	
Acción del Actor	**Respuesta del Sistema**
	1a. El sistema muestra un mensaje informado de que no existen Diagrama Entidad Relación modelado en la herramienta de VP.
Sección 2: "Listar entidades del Diagrama Entidad Relación seleccionado"	
Acción del Actor	**Respuesta del Sistema**
1. El desarrollador selecciona el Diagrama Entidad Relación del listado mostrado por el sistema.	
	2. El sistema muestra el listado de entidades referente al Diagrama Entidad Relación seleccionado.
Flujos Alternos al paso 2 "Ausencia de entidades en el Diagrama Entidad Relación seleccionado"	
Acción del Actor	**Respuesta del Sistema**
	2a. El sistema muestra un mensaje informado de que no existen entidades en el Diagrama Entidad Relación seleccionado.
Sección 3: "Adicionar entidades al listado de generación"	
Acción del Actor	**Respuesta del Sistema**
1. El desarrollador selecciona de forma parcial o total las entidades a generar a través del botón ">" o ">>" respectivamente.	
	2. El sistema adiciona la entidad al listado para la generación de las CAD de forma parcial o total.
Flujos Alternos al paso 2 "No existen elementos seleccionados"	
Acción del Actor	**Respuesta del Sistema**
	2a. El sistema informa que debe seleccionar un elemento de la lista.

Sección 4: "Remover entidades del listado de generación"	
Acción del Actor	**Respuesta del Sistema**
2. El desarrollador remueve de forma parcial o total las entidades a generar a través del botón "<" o "<<" respectivamente.	
	3. El sistema remueve la entidad del listado para la generación de las CAD de forma parcial o total.
Flujos Alternos al paso 2 "No existen elementos seleccionados"	
Acción del Actor	**Respuesta del Sistema**
	2a. El sistema informa que debe seleccionar un elemento de la lista.
Sección 5: "Obtener dirección del directorio"	
Acción del Actor	**Respuesta del Sistema**
1. El desarrollador selecciona la opción "Examinar".	
	2. Es sistema muestra una interfaz para seleccionar el lugar a generar las CAD
3. El desarrollador selecciona la dirección destino de la carpeta para la generación y hace clic en el botón "Guardar"	
	4. El sistema muestra el directorio seleccionado por el desarrollador en un campo de texto.
Flujos Alternos al paso 3 "No se ha podido obtener la dirección"	
Acción del Actor	**Respuesta del Sistema**
3a. El desarrollador selecciona la dirección destino de la carpeta para la generación y hace clic en el botón "Cancelar"	

	3b. El sistema muestra un mensaje informando que no se ha podido obtener la dirección del directorio.
Sección 6: "Generar las clases de acceso a datos"	
Acción del Actor	**Respuesta del Sistema**
1. El desarrollador selecciona la opción de generar las clases de acceso a datos, haciendo clic sobre el botón "Generar"	
	2. El sistema valida que el tipo de datos del Diagrama Entidad Relación sea correcto para Doctrine. 3. El sistema valida que la estrategia de id sea correcta para Doctrine. 4. El sistema inicia el CUS "Generar Código" (ver descripción de CUS) obteniendo los códigos del listado de entidades para generar. 5. El sistema informa de su correcta ejecución de la generación, terminado así el CUS.
Flujos Alternos al paso 2 y 3"Error"	
Acción del Actor	**Respuesta del Sistema**
	2a. El sistema muestra un mensaje informando los tipos de datos que no son correctos para Doctrine 3a. El sistema muestra un mensaje informando error en la estrategia seleccionada.

Prototipo de Interfaz

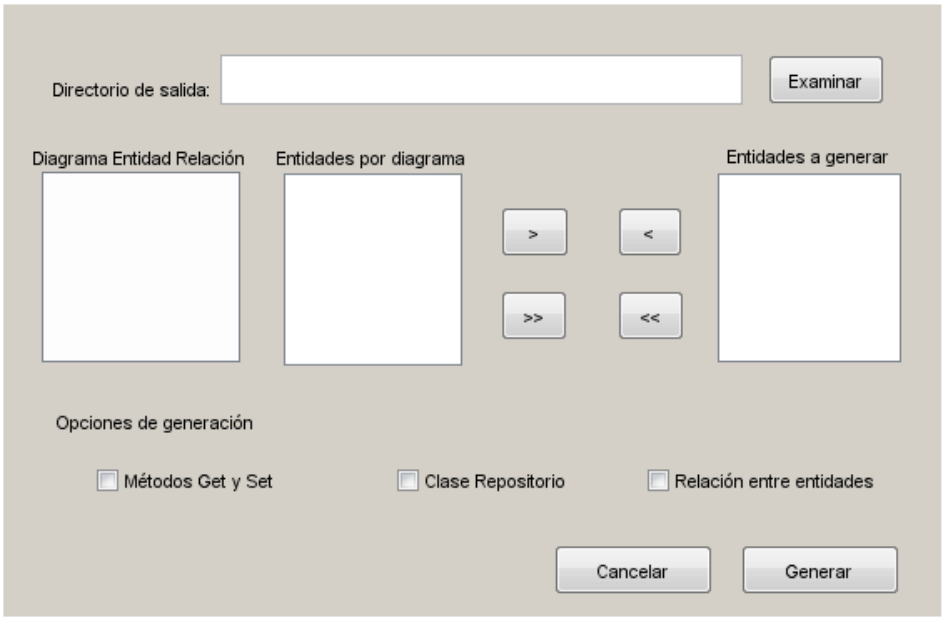



Figura # 7: Prototipo de interfaz de usuario “Generar CAD para Doctrine”.
Poscondiciones | Quedaron generadas las CAD de las entidades seleccionadas.

Tabla # 3: Descripción del CU “Administrar clases de acceso a datos”.

2.5 Modelo de diseño

Luego de la descripción del funcionamiento de cada caso de uso, se hace necesario conocer las bases de la arquitectura que permiten las tareas descritas entre las acciones realizadas por el actor y las respuestas del sistema, a través de la creación del modelo de diseño de la extensión.

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

Diagrama de Clase del Diseño

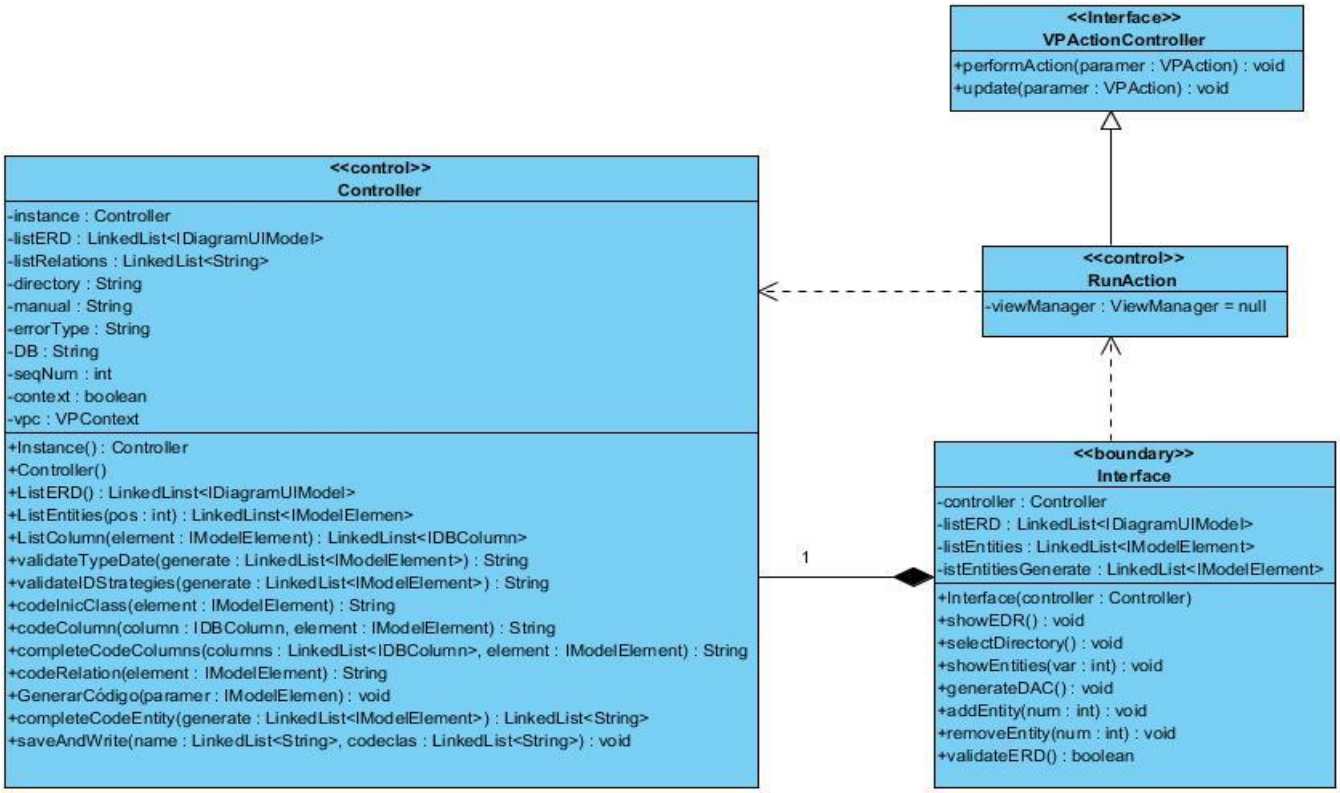


Figura # 8: Diagrama de Clase del Diseño.

Clases	Descripción
VPActionController	Es la clase que brinda el Open API por defecto para permitir la actualización de las acciones realizadas a través de la herramienta.
RunAction	Es la clase que permite crear la interfaz visual de la extensión y junto con ella el objeto de la controladora que será utilizado durante todo el proceso que se realice desde la interfaz.
Controller	Es la clase que contiene todas las funcionalidades que permiten desde la obtención de los datos del Diagrama Entidad Relación, hasta su transformación y creación de las CAD.
Interface	Interfaz visual que permite la interrelación con el desarrollador y la extensión para realizar las operaciones necesarias en la generación de las CAD.

Tabla # 4: Descripción de las clases del diagrama de diseño.

2.6 Patrones utilizados

Para realizar un diseño eficiente durante el desarrollo de la extensión se propone el uso de los patrones al permitir capturar, reutilizar y transmitir la experiencia obtenida por otros desarrolladores en situaciones similares. La guía que representa el empleo de los patrones para describir las formas de solucionar los problemas que se presentan de forma recurrente en los desarrollos de software, constituye un aspecto significativo a destacar durante la implementación de la extensión.

Como características deseables para la aplicación de un patrón se encuentran: (18)

- **Debe solucionar un problema:** los patrones capturan soluciones, no solo principios o estrategias abstractas
- **Debe ser un concepto probado:** ser soluciones demostradas, no teorías o especulaciones.
- **La solución no es obvia:** muchas técnicas de solución de problemas tratan de hallar soluciones por medio de principios básicos. Los mejores patrones generan una solución a un problema de forma indirecta.
- **Describe participantes y relaciones entre ellos:** los patrones no sólo describen módulos sino estructuras del sistema y mecanismos más complejos.

Los elementos esenciales que debe contener un patrón son: (18)

- **Nombre:** Tiene que tener un nombre significativo. Sería muy incontrolable tener que describir el patrón cada vez que se utiliza en una discusión.
- **Problema:** Describe cuándo aplicarlo, explica el problema y su contexto y la lista de precondiciones que deben encontrarse, si las hubiera.
- **Solución:** Describe los elementos que lo componen: clases, objetos, relaciones, responsabilidades y colaboraciones.
- **Consecuencias:** Describe los costos y beneficios de aplicarlo.

Una vez analizadas las principales particularidades de los patrones de software y teniendo en cuenta las clasificaciones en que pueden dividirse, resulta necesario analizar con mayor profundidad los patrones utilizados para el desarrollo de la solución. Estos se encuentran agrupados en dos categorías: Patrón arquitectónico por capas y Patrones de diseño, como se describe a continuación.

Patrón arquitectónico por capas

Los patrones arquitectónicos o patrones de arquitectura ofrecen soluciones a problemas de arquitectura de software, al proporcionar una descripción del tipo de relación y restricciones entre los elementos que conforman el diseño. Un patrón arquitectónico expresa un esquema de organización estructural esencial para un sistema de software, que consta de subsistemas, sus responsabilidades e interrelaciones. Son vistos con un nivel de abstracción mayor que los patrones de diseño, porque muchas arquitecturas diferentes pueden implementar el mismo patrón y por lo tanto compartir las mismas particulares.

La selección de un patrón de arquitectura está en correspondencia con las particularidades que posea el sistema y ajustándose a sus necesidades. Estas pueden variar en dependencia de la conformación del mecanismo de almacenamiento, presentación, controlador o negocio, entre otros componentes que pueda manejar el sistema.

Según las características de la extensión, se propone el uso del estilo arquitectónico por capas. Esta arquitectura tiene como objetivo principal el de separar los diferentes aspectos del desarrollo en que está conformado el sistema y permite además la construcción de sistemas débilmente acoplados, minimizando las dependencias entre capas. Al dividir un sistema en capas, cada una puede tratarse de forma independiente, sin tener que conocer los detalles de las demás. Según David Garlan y Mary Shaw en el libro "An introduction to Software Architecture", definen este estilo como una organización jerárquica tal que cada capa proporciona servicios a la capa inmediatamente superior y se sirve de las prestaciones que le brinda la inmediatamente inferior.

La organización de las capas en la extensión quedaría agrupada en dos grupos compuesta por la capa Controladora y Presentación. La capa Controladora se encuentra representada principalmente por la clase Controller que contiene la mayoría de las operaciones implementadas para el funcionamiento de la extensión. La capa de Presentación constituida por la clase Interface, permite al desarrollador la selección y configuración de la generación de las clases de acceso con el uso de las funcionalidades implementadas en la clase Controller. De esta manera se plantea que la capa de Presentación interactúa con la capa Controladora y desde la filosofía de arquitectura en capas, esto significa que la capa de Controladora presenta un conjunto de funcionalidades para brindar servicios a la capa inmediatamente superior de Presentación.

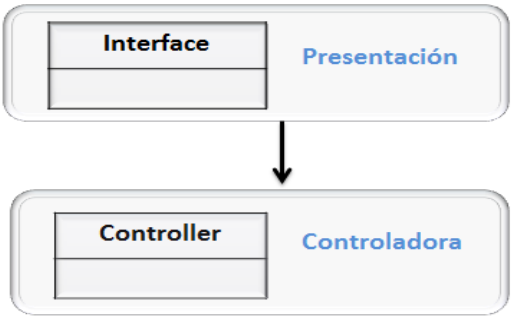


Figura # 9: Arquitectura por capa de la extensión.

Patrones de diseño

La aplicación de los patrones consiste en identificar el patrón que resuelve el problema de diseño encontrado y aplicar la solución abstracta prescrita por el patrón a dicho problema. (18) Con el uso de estos patrones se puede lograr un mejor entendimiento y documentación de diseños orientados a objeto. Mejora en el mantenimiento de sistemas, al proveer una especificación explícita de clases e interacción entre objetos. Ayuda a prevenir los fallos inesperados durante la ejecución de software al proporcionar un vocabulario para exponer decisiones de diseño en término de estructuras de clases en lugar de objetos, aumentando la legibilidad del código para implementadores y arquitectos.

Existen varios patrones de diseño utilizados en el proceso de desarrollo del software que describen los principios fundamentales del diseño de objetos. Estos son agrupados fundamentalmente por dos grandes grupos conocidos como Patrones de Software para la asignación General de Responsabilidad (GRASP - *General Responsibility Assignment Software Patterns*) y “La Banda de los cuatro” (GOF - *Gang of Four*).

Patrones GRASP

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

- Experto

La responsabilidad de asignar una labor a la clase que tiene o puede tener los datos necesarios para cumplir determinada responsabilidad, es la solución que pretende dar este patrón ante el problema de cómo realizar la asignación de la forma más eficiente posible. Si estas asignaciones de responsabilidades se hacen adecuadamente, los sistemas tienden a ser más fáciles de entender, mantener y ampliar, lo que nos ofrece la garantía de poder reutilizar los componentes en futuras aplicaciones.

En el diagrama de clases del diseño de la extensión el patrón experto se ve reflejado en la clase Controller, utilizada por la clase interfaz para asignarle la responsabilidad de las funcionalidades a realizar durante la ejecución de la extensión. Esta clase se identifica como la experta en la información por permitir obtener, almacenar y procesar los datos del Diagrama Entidad Relación modelado en el Visual Paradigm, que serán utilizados para la generación de las clases de acceso a datos para Doctrine 2.0.

- Controlador

Es el encargado de asignar la responsabilidad del manejo de uno de los eventos del sistema, a una clase facultada de atender determinada funcionalidad, solucionando así el problema fundamental de este patrón, al saber quién debería ocuparse de dicho evento. En todos los casos, si se recurre a un diseño orientado a objetos, hay que elegir los controladores que manejen esos eventos de entrada.

En el diagrama de clases del diseño de la extensión se ve reflejado el patrón Controlador en la clase RunAction al extender del comportamiento y redefinir las funcionalidades que permiten el control de los eventos proporcionados por la clase VPActionController. Esta clase provista por la librería OpenAP, otorga a la clase RunAction la capacidad de comunicar el evento generado por el desarrollador al hacer clic en el menú de Visual Paradigm en donde se encuentra la opción de “Generar CAD para Doctrine 2.0”. Este evento genera como respuesta la ejecución de la implementación realizada en el método performAction(), que permite crear la interfaz visual de la extensión.

Patrones GOF

Los patones GOF se revelan como una forma indispensable de enfrentarse a la programación a raíz del libro “Design Patterns: Elements of Reusable Object Oriented Software” escrito por los ahora famosos

"Gang of Four" formada por Erich Gamma, Richard Helm, Ralph Johnson y John Vlissides. Ellos recopilaron y documentaron 23 patrones de diseño aplicados usualmente por expertos diseñadores de software orientado a objetos. Desde luego que no son los inventores ni los únicos involucrados, pero inmediatamente de la publicación del libro es que empezó a difundirse con más fuerza la idea de patrones de diseño.

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

- Singleton o Solitario

Este patrón es de tipo creacional a nivel de objetos. Su principal objetivo es garantizar que una clase solo tenga una única instancia, proporcionando un punto de acceso global a la misma. Permite realizar refinamientos en las operaciones y en la representación, mediante la especialización por herencia de "Solitario". Es fácilmente modificable para permitir más de una instancia y para controlar el número de las mismas (incluso si es variable) [24].

En la extensión es utilizado este patrón directamente en la implementación de la clase Controller para mantener la constancia entre los objetos obtenido del DER modelado por el diseñador de base de datos. Esta única instancia de la clase pone de manifiesto el uso del patrón Singleton y su ventaja de evitar la duplicidad a través del punto de acceso global que es capaz de definir. El siguiente código muestra la aplicación en la extensión del patrón descrito.

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

Figura # 10: Ejemplo de método Singleton en la extensión.

- Iterator o Iterador

El patrón Iterador es de tipo comportamiento a nivel de objetos. A través de su utilización es posible acceder de forma secuencial a cada uno de los elementos de un objeto agregado sin exponer su representación interna. Además, permite realizar recorridos sobre objetos compuestos independientemente de la implementación de estos. Su utilización tributa al incremento de la flexibilidad porque es posible utilizar nuevas formas de recorrer una estructura con solo modificar el iterador en uso, cambiarlo por otro o definir uno nuevo. También facilita el paralelismo y la concurrencia, pues es posible que dos o más iteradores recorran una misma estructura simultánea o solapadamente.

En la extensión propuesta, es aplicado este patrón directamente en la implementación de la clase controladora cuando se necesita proporcionar una interfaz uniforme para recorrer las diferentes estructuras proporcionadas por Visual Paradigm. En la captura de todos los Diagrama Entidad Relación creados, el Iterator permite obtener las tabas contenidas por diagramas, las columnas por tablas, los atributos de las columnas y almacenarlos en un listado, la información para su posterior transformación hacia las clases de acceso a datos. En el siguiente código se muestra la aplicación en la extensión del patrón descrito.

```
public LinkedList<IDBColumn> listColumn(IModelElement element) {

    LinkedList<IDBColumn> list = new LinkedList<IDBColumn>();
    Iterator childElement = element.childIterator();

    while (childElement.hasNext()) {

        IDBColumn column = (IDBColumn) childElement.next();
        list.add(column);
    }
    return list;
}
```

Figura # 11: Ejemplo de método Iterator en la extensión.

2.7 Diagrama de Secuencia

Un diagrama de secuencia muestra la interacción de un conjunto de objetos en una aplicación a través del tiempo y se modela para cada caso de uso. Mientras que el diagrama de casos de uso permite el modelado de una vista de negocio del escenario, el diagrama de secuencia contiene detalles de implementación del escenario, incluyendo los objetos y clases que se usan para implementar el escenario, y mensajes intercambiados entre los objetos. (19)

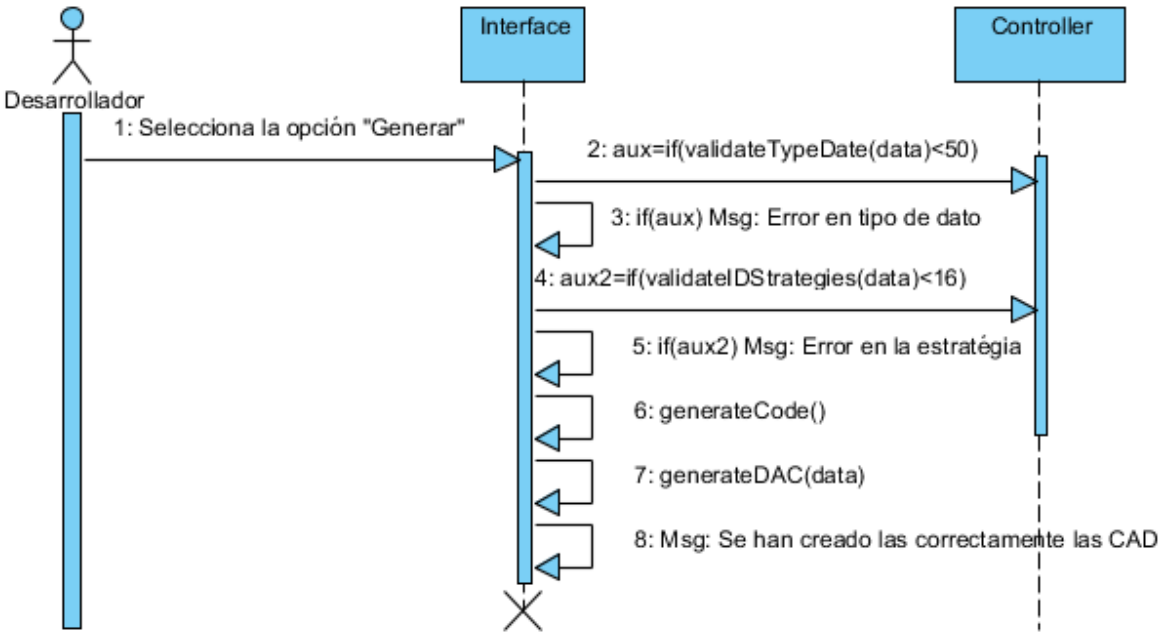


Figura # 12: Diagrama de Secuencia. CU uso “Administrar clases de acceso a datos”.

El presente diagrama se corresponde con el escenario seis del caso uso “Administrar clases de acceso a datos” y muestra el flujo de acciones que ocurren cuando el desarrollador selecciona en la clase interfaz la opción Generar. Lo primero que se realiza es el llamado al método validateTipeDate() de la clase Controller para garantizar el correcto mapeo de los datos en la base de datos. La segunda validación examina que el tipo de estrategia seleccionada corresponde con el sistema de base de datos a utilizar para evitar errores de validación. En caso de que no se cumpla con las condiciones programadas, las variables definirán el tipo de error cometido informándolo al desarrollador a través de un mensaje. Luego se podrá generar el código y las clases de acceso a datos a través de los métodos generateCode() y

generateDAC() respectivamente. Al finalizar se informará al desarrollador de la correcta implementación de las clases terminadas.

2.8 Conclusiones

En este capítulo se expuso los elementos fundamentales que ayudan a los usuarios, desarrolladores e interesados a entender el análisis y diseño para la implementación de la extensión. Entre los elementos obtenidos a lo largo del capítulo se encuentra el modelo de dominio como ente de partida para el diseño de la extensión nombrada como Cader. Para el desarrollo de extensión se propuso como solución un proceso de tres sesiones que reúne el modo de implementación y funcionamiento de la extensión. La descripción de las funcionalidades a implementar, quedaron establecidas a través de los once requisitos funcionales identificados así como los requisitos no funcionales de software, hardware, usabilidad, portabilidad, licencia y las restricciones de diseño e implementación. Finalmente se establecieron la base de la arquitectura de la extensión compuesta por el Diagrama de Clases del Diseño y la aplicación de los patrones de diseño GRASP (Experto y Controlador), GOF (Singleton e Iterator) y el arquitectónico por capas para garantizar un mejor entendimiento y documentación de diseños orientado a objeto .

CAPÍTULO 3: IMPLEMENTACIÓN Y PRUEBA

En el siguiente capítulo se abordará sobre el modelo de implementación realizado para la confección de la extensión. Los pasos fundamentales para el desarrollo de la extensión a través de los estándares de codificación y pruebas que se realicen. El proceso de prueba de software será llevado a cabo para la detección y corrección de los errores que pueda presentar la aplicación con el objetivo de darles solución y respuesta a los mismos. Terminada las pruebas se describe el proceso de integración y utilización del método Delphi para demostrar el cumplimiento del problema planteado.

2.1 Modelo de implementación

El modelo de implementación describe cómo los elementos del diseño se implementan en términos de componentes, que pueden ser ficheros de código fuente, ejecutables, entre otros. Este modelo describe cómo se organizan los componentes de acuerdo con los mecanismos de estructuración y modularización disponibles en el entorno de implementación. Representa los lenguajes de programación utilizados y cómo dependen los componentes unos de otros. Describe una jerarquía de subsistemas de implementación que contiene componentes e interfaces. (20)

El modelo de implementación se encuentra compuesto por el diagrama de despliegue y de componente que se describen a continuación:

Diagrama de despliegue

El Modelo de Despliegue o diagrama de despliegue muestra las relaciones físicas entre los componentes de hardware y software utilizados en las implementaciones de sistemas y las relaciones entre sus componentes. Está compuesto por los distintos nodos que componen el sistema y el reparto de los componentes por nodo estará en correspondencia según las características del sistema.

Para la representación del diagrama de despliegue de la extensión se utilizó un único nodo de procesamiento en representación de la PC Cliente. Este nodo corresponde con el componente de hardware en el cual se encuentra instalado el software de Visual Paradigm y este a su vez, integra la extensión implementada.

Diagrama de componentes

Es el encargado de modelar la vista estática de un sistema y describir los elementos físicos compuesto por archivos, librerías, módulos, ejecutables o paquetes. Pueden ser usados para modelar y documentar cualquier arquitectura del sistema. Los diagramas de componentes son empleados para estructurar el modelo de implementación en términos de subsistemas y mostrar las relaciones entre sus elementos. Presentan las restricciones impuestas por los lenguajes de programación y las herramientas utilizadas en el desarrollo en cada subsistema de implementación.

Entre los diferentes tipos de componentes utilizados en la extensión se encuentran:

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

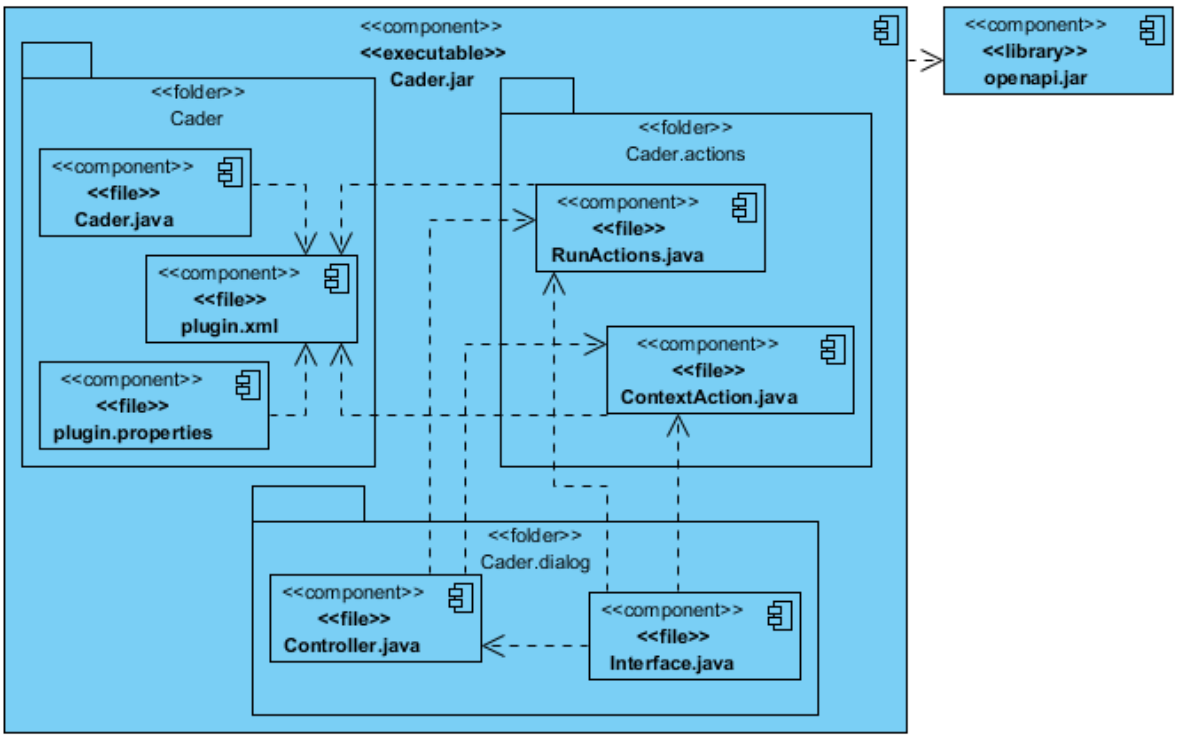


Figura # 13: Diagrama de componentes de la extensión.

Descripción de los componentes del diagrama:

Componentes	Descripción
Paquete Cader	Paquete que contiene las clases relacionadas con la configuración de la extensión: plugin.xml, Cader.java y plugin.properties
Paquete Cader.actions	Paquete que agrupa las clases referentes a las acciones que son accesibles a través contexto (área de diseño del diagrama en Visual Paradigm) o del menú herramientas. Ellas son: RunAction.java, ContexAction.java y Controller.java
Paquete Cader.dialog	Paquete que contiene el formulario de la aplicación a través de la clase Interface.java y la clase Controller.java que utiliza para la realización de las funcionalidades de la extensión.
Cader.java	Carga y descarga el plugin mediante la clase VPPluginInfo que provee el openapi.jar, capturando la información definida en el archivo plugin.xml.
Plugin.xml	Su función consiste en la configuración del plugin, define el nombre de la extensión, descripción, proveedor y las acciones a ejecutar.
Plugin.properties	Define propiedades asociados a los eventos y acciones de la extensión.

Tabla # 5: Descripción de los componentes.

2.2 Implementación de la extensión Cader

La implementación de la extensión Cader se alcanzó a partir de la ejecución de las tres etapas de obtención, transformación y generación de las clases de acceso. La realización de estas etapas depende del Diagrama Entidad Relación, previamente creado por el diseñador de base de datos en la fase de análisis y diseño. Para la conformación de este diagrama, el diseñador debe haberse regido según los tres pasos que propone la metodología de desarrollo de base de datos (modelo conceptual, diseño lógico y diseño físico), en pos de garantizar una mayor calidad en la representación y organización de los datos. Luego de asegurado estos aspectos de buenas prácticas para la correcta confección del Diagrama Entidad Relación se prosiguen a describir cada uno de las etapas siguientes:

Obtención de los datos

En la etapa de obtención de los datos del Diagrama Entidad Relación modelado, es necesario el uso de un conjunto de componentes de los ofrecidos por la librería openapi.jar, de los cuales son utilizados los que se describen a continuación:

Componentes	Descripción
ApplicationManager	Interfaz que permite el control de la aplicación.
IProject	Interfaz que define el proyecto en que se trabaja.
IDiagramUIModel	Interfaz que define los diagramas modelados.
IDiagramElement	Interfaz que define los elementos de un diagrama.
IModelElement	Interfaz que define un elemento del Diagrama Entidad Relación.
IDBColumn	Interfaz que define una columna dentro de una tabla en un Diagrama Entidad Relación.
IDBForeignKey	Interfaz que define en una tabla la llave foránea para las relaciones entre tablas.
IRelationShip	Interfaz que define una relación entre cualquier objeto IModelElement.
IAssociationEnd	Interfaz que define el tipo de relación entre clases.

Tabla # 6: Descripción de los componentes utilizados en Visual Paradigm.

La obtención de los datos del modelo comienza cuando el desarrollador accede a través del clic derecho sobre el contexto del Diagrama Entidad Relación o por la barra de menú en la ruta de acceso: Herramientas\Mapeo Relacional de Objeto (ORM)\Generar CAD para Doctrine. Para ambos casos, se crea la instancia única de la clase controladora (Controller) pasada como parámetro a la interfaz visual (Interface) en el momento de ser mostrada. Esto es permitido por la clase RunAction al implementar los métodos performAction() y update(), de la clase VPActionController ofrecida por el API para actualizar las acciones que se realizan en la herramienta. La creación de la interfaz incluye la llamada al método ListERD() encargado de obtener el listado de los diagramas entidad relación (IDiagramUIModel) modelado en el proyecto en que se trabaja, por medio del componente IProject. Luego el desarrollador podrá seleccionar entre todos los diagrama modelados en proyecto, haciendo clic sobre él. La interfaz obtendrá

todo el listado de entidades (IModelElement) que contiene el DER seleccionado a través del método listEntities(). Por último, el desarrollador en dependencia de las necesidades que posea sobre las clases que desea generar, podrá adicionar (addEntity()) de forma parcial o total las entidades hacia el listado para su posterior generación.

Transformación del Modelo

Para la etapa de transformación del modelo, luego de haber realizado la obtención de los datos modelados en el Diagrama Entidad Relación, se ejecutan los métodos que van creando de manera independiente el código de las clases de acceso a datos. Ese código es conformado por las diferentes anotaciones Docblock utilizadas por Doctrine para representar los metadatos de las clases a generar. A continuación se presenta en la tabla las anotaciones empleadas y sus atributos con las respectivas descripciones:

Anotación	Descripción	Atributos	Descripción
@Table	Anotación que configura las características de la tabla.		
@Entity	Anotación requerida para marcar a una clase PHP como entidad.	Repository Class	Especifica la dirección en donde se encuentra la clase repositorio
@Column	Anotación que identifica a una variable o columna como persistente	type	Tipo de dato de la variable.
		Name	Nombre de la columna de la base de datos
		length	Utilizado por el tipo de dato “String”, para determinar su máxima longitud en la base de datos.
		precision	Utilizado para definir la precisión del tipo de dato “decimal”.

		scale	Utilizado para definir la escala del tipo de dato “decimal”.
		unique	Determinar si el valor de la columna debe ser único en todas las filas de la tabla.
		nullable	Determina si el valor NULL es permitido para la columna donde se utilice.
@Id	Anotación que indica a una columna como identificador único de la entidad, es la clave principal en la base de datos.		
@Generated Value	Anotación que especifica la estrategia utilizada para la generación del identificador de una variable que es identificada con la anotación @Id. De instancia que es anotado por @ Id.	strategy	Establece el tipo de estrategia de generación del id, entre los valores validos se encuentran: **AUTO:** (Opción predeterminada) Ofrece una portabilidad completa al indicarle a Doctrine la estrategia preferida por la plataforma de base de datos utilizada. **SEQUENCE:** Le dice a Doctrine que utilice una secuencia de base de datos para la generación de id. Compatible solo con Oracle y PostgreSql. **IDENTITY:** Le dice a Doctrine que debe utilizar columnas de identidad especiales en la base de datos que generen un valor al insertar una fila. Compatible solo con MySQL/SQLite (AUTO_INCREMENT), MSSQL (IDENTITY) y PostgreSQL (SERIAL). **TABLE:** Esta estrategia no se encuentra implementada todavía por Doctrine.

			NONE: Es lo mismo que omitir el @GeneratedValue.
@SequenceGenerator	Anotación que permite especificar detalles acerca de la secuencia, como el tamaño de incremento y los valores iniciales de la secuencia.	Sequence Name	Nombre de la secuencia.
		Allocation Size	Utilizado para incrementar el tamaño de la secuencia de asignación.
		initialValue	Utilizado para indicar el valor inicial de la secuencia. Por defecto se inicia con valor 1.
@OneToOne @OneToMany @ManyToOne	Anotaciones que indican las relaciones que son utilizadas entre dos entidades modeladas en el DER.	targetEntity	Determina la entidad que es objeto de referencia.
		inversedBy	Designa el campo en la entidad que es el lado inverso de la relación.
@JoinColumn	Anotación utilizada por las relaciones @ManyToOne y @OneToOne	name	Nombre de la columna que contiene el identificador de clave externa para esta relación.
		referenced Column Name	Nombre del identificador de clave principal que se utiliza para la unión de esta relación.

Tabla # 7: Descripción de las anotaciones utilizadas en Doctrine 2.0

La etapa de transformación del modelo comienza con la llamada en la clase Controller al método completeCodeEntity(). Este método recibe el listado de entidades seleccionadas por el desarrollador para la generación y retorna el listado con el código generado por cada entidad, en posiciones distintas de la lista. La generación del código para cada posición de la lista empieza con la inicialización de la clase en PHP, el namespace y las anotaciones necesarias para configuración (@Table) e identificación (@Entity) de la clase como entidad. Luego se obtiene el código referente a cada columna que posea la entidad a través del método codeColum(). Este método es el encargado de asignar las características que posee la

columna a los atributos que conforman el código. Para el caso de que la columna sea llave primaria el método codePrimaryKey() incluirá la notación @Id y el tipo de generador de valor (@GeneratedValue) seleccionado. En caso de que el generador sea de tipo secuence, se utilizara @SequenceGenerator para establecer los valores de inicio e incremento a utilizar. Finalizado la confección del código de las columnas se prosigue utilizar el método codeRelation() para crear las relaciones (@OneToOne, @OneToMany y @ManyToOne) que se efectúan desde una entidad hacia otra. De igual forma pero en sentido opuesto a la relación entre entidades, se crean las relaciones bidireccionales con las entidades enlazadas y para el caso de @ManyToOne o @OneToOne se utiliza @JoinColum para establecer la relación. Las confección de los código de las relaciones, como de los métodos Get() y Set() pude ser opcional, al igual que los códigos de cada clase Repositorio.

Generación de las CAD

La etapa de generación de las CAD comienza luego de haberse transformado el modelo y obtener con él, los códigos necesarios para concebir las entidades a generar. La llamada al método saveAndWrite() en la clase Controller, es el encargado de llevar a cabo la confección de las clases de acceso a datos y las de Repositorio. Estas son creadas en una carpeta “Entity” ubicada según el directorio seleccionado por el desarrollador. A cada clase se le escribe el código correspondiente generado, terminando así el método al mostrar un mensaje de la creación de la carpeta Entity según el directorio entrado por el desarrollador.

2.3 Estándares de codificación

Los estándares de codificación son pautas de programación que no están enfocadas a la lógica del programa, sino a su estructura y apariencia física del código. Las normas de codificación definidas por la línea de arquitectura están enfocadas a lograr una mejor comprensión y mantenimiento del código que responda a la complejidad dada por la cantidad de componentes y el alto nivel de integración que puede existir en el desarrollo del software.

Las realizaciones de revisiones frecuentes al código, la aplicación de forma continua de técnicas y estándares de codificación definidos, junto al empleo de buenas prácticas de programación, garantizan una alta calidad en el desarrollo de la extensión además de proporcionar un código legible y reutilizable. Entre los estándares utilizados durante la implementación se encuentran:

Tamaño y organización de las líneas de código

La longitud de línea es aproximadamente de 80 caracteres. En caso de que una expresión ocupe más de este rango, podrá romper o dividirse en una nueva línea y deberán estar todas alineadas con respecto a la anterior, para mantener la legibilidad del código

Declaraciones

Toda variable local tendrá que ser inicializada en el momento de su declaración, salvo que su valor inicial dependa de una llamada a otro método en determinado momento de ejecución de método. Las declaraciones deben situarse al principio de cada bloque principal en el que se utilicen y nunca en el momento de su uso.

Llaves de apertura y cierre

Se utiliza siempre llaves de apertura y cierre, incluso en situaciones en las que técnicamente son opcionales. Esto aumenta la legibilidad y disminuye la probabilidad de errores lógicos.

Nomenclaturas Utilizadas

PascalCase establece que los nombres de los identificadores, las variables, métodos y clases están compuestos por una o más palabras juntas, iniciando cada palabra con letra mayúscula y el resto en minúscula. Todas las clases están nombradas siguiendo el estándar PascalCase, nombrándolas de acuerdo al propósito y la función que corresponda. Ejemplo: las clases Controller y RunAction

CamelCase es similar a PascalCase con diferencia en la letra inicial del identificador no comienza con mayúscula. Esta notación se utilizó para el nombre de funciones, atributos y variables.

Nomenclatura de las funciones: El identificativo de todas las funciones o métodos se escribe con la primera palabra en minúscula de acuerdo a la función que realizan. Ejemplo: listEntities() y listColumn()

Nomenclatura de los atributos: El identificativo de los atributos se escribe de acuerdo a su objetivo con la primera letra en minúscula. Ejemplos: errorCASEType y seqNum

Notación húngara

Notación también conocida como REDDICK por el nombre de su autor está basada en definir prefijos para cada tipo de datos según el ámbito de las variables, con la finalidad de lograr mayor comprensión del nombre de la variable, método o función.

Nomenclatura de las variables. El identificativo de los atributos se escribe atendiendo a su objetivo y de acuerdo al estándar CamelCase, con la primera letra en minúscula. Ejemplo: codeList, que define una lista de código. columList, que define una lista de columnas.

Nomenclatura de las constantes. El identificativo de las constantes se realiza utilizando todas las letras en mayúscula. Ejemplo: DB (Base de Datos) y ERD (Diagrama Entidad Relación).

2.4 Ejemplo de código

El siguiente fragmento de código corresponde al escenario número uno del caso uso Listar DER y su implementación se encuentra dentro de la etapa de obtención de los datos. En el código se evidencia el uso de los estándares de codificación aplicados como PascalCasing, CamelCase, notación húngara y otras técnicas de las mencionadas anteriormente.

```
public LinkedList<IDiagramUIModel> listERD() {

    IProject project = ApplicationManager.instance().getProjectManager().getProject();

    Iterator diagramIter = project.diagramIterator();

    LinkedList<IDiagramUIModel> listDiagram = new LinkedList<IDiagramUIModel>();

    String ERD = DiagramManager.DIAGRAM_TYPE_ENTITY_RELATIONSHIP_DIAGRAM;

    while (diagramIter.hasNext()) {

        IDiagramUIModel diagram = (IDiagramUIModel) diagramIter.next();

        if (diagram.getType().equals(ERD)) {

            listDiagram.add(diagram);
        }
    }
    return listDiagram;
}
```

Figura # 14: Ejemplo de código de la extensión implementada.

2.5 Interfaz de la extensión Cader

Luego de la implementación de las tres etapas y con el uso de los estándares de codificación utilizados se obtuvo la extensión Cader, la cual quedo representada a través de la siguiente interfaz gráfica:

Figura # 15: Interfaz de la extensión Cader

2.6 Prueba de software

La prueba del software es un elemento crítico para la garantía de la calidad del software. El objetivo de la etapa de pruebas es garantizar la calidad del producto desarrollado. Las pruebas son un proceso que se enfoca sobre la lógica interna del software y las funciones externas, es un proceso de ejecución de un programa con la intención de descubrir un error. Una prueba tiene éxito si descubre un error no detectado hasta entonces. (21)

Las pruebas son consideradas un pilar indispensable para la evaluación del software, al comprobar los distintos productos o modelos que se van generando durante todo el proceso de desarrollo. La verificación de los requisitos que debe cumplir el software a través del análisis a las posibles combinaciones de entradas y de las salidas de datos, además de la comprobación de que el software trabaje como fue diseñado, son algunos de los objetivos que se persigue con la realización de las pruebas. Como resultado permitirá un mayor control e identificación temprana de los defectos y fallos garantizando la calidad del desarrollo con una reducción notable de los costos necesarios para corregir los errores.

En la evaluación dinámica del sistema se aplicó durante todo el ciclo de desarrollo de la extensión mediante el **nivel de prueba**: Pruebas de Desarrollador, la cual está diseñada e implementada por el equipo de desarrollo. Cada nivel de prueba engloba una técnica de prueba específica según los atributos de calidad que se deseen verificar con las pruebas al software. La **técnica de prueba** seleccionada es la Prueba Funcional, la cual tiene como principal objetivo asegurar el trabajo apropiado de los requisitos funcionales, incluyendo la navegación, entrada de datos, procesamiento y obtención de resultados, a través del **método de prueba** de Caja Negra.

Este método es conocido también como Pruebas de Comportamiento y se ejecuta por cada caso de uso usando datos válidos e inválidos, para verificar que los resultados esperados se correspondan con los mensajes apropiados de error o precaución respectivamente. Las pruebas se realizan sobre la interfaz del software y son completamente indiferentes al comportamiento interno y la estructura del programa. Los casos de prueba de Caja Negra constituyen un conjunto de condiciones de ejecución desarrollado para cumplir un objetivo en particular o una función esperada y demostrar el estado de operatividad en que se encuentra el software.

Para confeccionar los casos de prueba de Caja Negra se utilizó el criterio por técnica de Partición de Equivalencia, donde se divide el campo de entrada de un programa en variables con juegos de datos que tienden a ejercitar determinadas funciones del software. En esencia, esta técnica intenta dividir el dominio de entrada de un programa en un número finito de variables de equivalencia. De tal modo que se pueda asumir razonablemente que una prueba realizada con un valor representativo de cada variable es equivalente a una prueba realizada con cualquier otro valor de dicha variable. Las variables de equivalencia representan un conjunto de estados válidos y no válidos para las condiciones de entrada de un programa. (21)

Caso de prueba

Un caso de prueba se diseña según las funcionalidades descritas en los casos de uso. El propósito que se persigue con este artefacto es lograr una comprensión común de las condiciones específicas que la solución debe cumplir. Se comienza con la descripción de los casos de uso del sistema, como apoyo para las revisiones. Cada planilla de caso de prueba recoge la especificación de un caso de uso, dividido en secciones y escenarios, detallando las funcionalidades descritas en él y describiendo cada variable que recoge el caso de uso en cuestión. (21) De esta forma quedaron plasmadas en los artefactos de Diseño de Casos de Prueba basado en Casos de Uso a las revisiones realizadas a “Administrar clases de acceso a datos” y “Generar código”.

Resultados de pruebas

Después de realizar las pruebas de desarrollador mediante la técnica de prueba funcional y con el uso del método de Caja Negra, se verificó el correcto funcionamiento de la extensión al cumplir con todos sus requisitos funcionales. La realización de las pruebas permitió identificar los errores no detectados que hasta entonces, poseía la extensión. La siguiente tabla muestra el resultado obtenido durante las cuatro iteraciones de pruebas efectuadas. En ella se recogen la cantidad de requisitos que intervienen durante la ejecución de las pruebas así como el número de no conformidades, las cuales fueron resueltas al final de cada iteración a través de las pruebas de regresión.

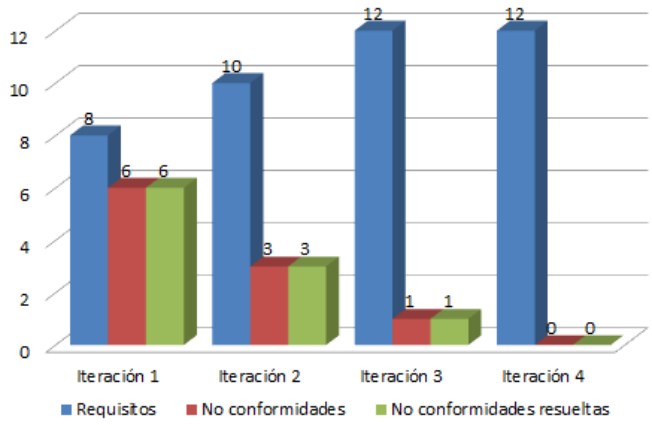


Figura # 16: Resultados de pruebas.

2.7 Integración con Doctrine 2.0

Luego de verificar el correcto funcionamiento de la extensión Cader mediante las pruebas realizadas, se prosigue a efectuar la integración de las clases de acceso a datos con el ORM Doctrine 2.0. Para la ejecución de esta integración se hace necesaria la existencia de dos condiciones previamente realizadas que se enumeran a continuación:

1. La creación desde el framework Symfony 2.0 del Bundle[10] en donde se ubicará la carpeta Entity.
2. La creación de una nueva base de datos (para este ejemplo se llamará: “cupón”).

Posteriormente se podrá utilizar la extensión Cader para la generación de las clases de acceso a datos a partir del Diagrama Entidad Relación del caso de estudio nombrado como “Cupón”. Este caso de estudio está conformado por las seis relaciones que se establecen entre las cinco entidades y las columnas pertenecientes a cada entidad, las cuales abarcan la mayoría de los tipos de datos utilizados por Doctrine 2.0. La siguiente figura muestra la representación del caso de estudio descrito:

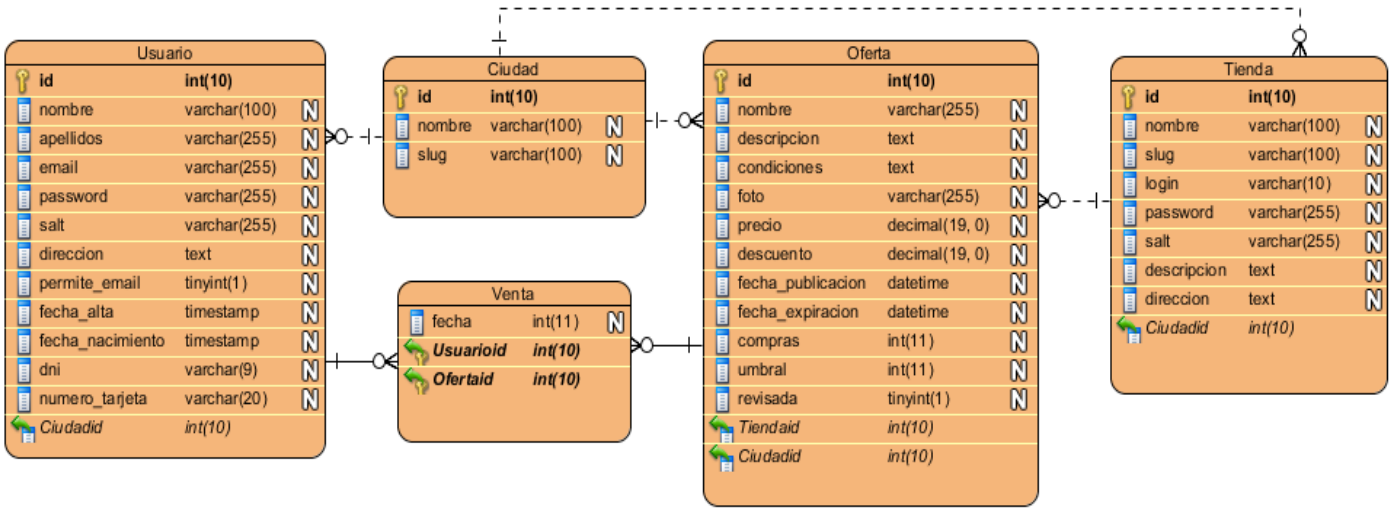


Figura # 17: Caso de estudio “Cupón”.

La generación de las entidades se realizará de forma total y el directorio de salida estará en correspondencia según la ubicación del Bundle creado. Entre las opciones de configuración de la

[10] Los *bundles* son la base de la nueva filosofía de trabajo de Symfony2. Técnicamente, un *bundle* es un directorio que contiene todo tipo de archivos dentro una estructura jerarquizada de directorios.

generación de las clases de acceso a datos ofrecida por la extensión Cader, se utilizarán las que aparecen seleccionadas por defecto. Esto posibilita que se creen los códigos de relaciones entre las entidades y se eviten errores por incompatibilidad entre el tipo de base de datos a utilizar y la estrategia de generación de ID escogida, además de los errores que puede incluir establecer de forma incorrecta el área de trabajo o namespace de las entidades mediante la forma manual.

Después de generadas las entidades, se accede a la consola y se introduce la dirección en donde se encuentra el framework Symfony. Desde esta ubicación, se crea el esquema de la base de datos mediante el comando: *php app\console doctrine:schema:create,* como se muestra en la siguiente figura:

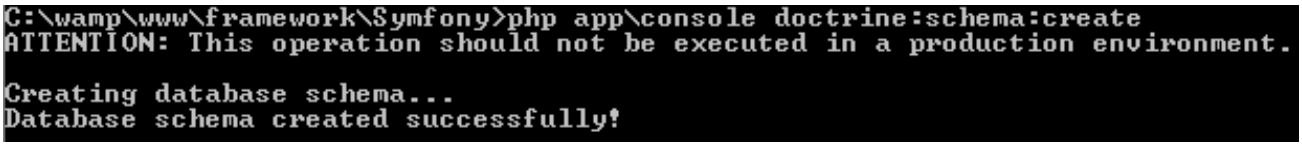


Figura # 18: Creación del esquema de la base de datos.

La creación del esquema de forma satisfactoria a partir de las clases de acceso a datos generadas, confirman la correcta integración que se establece entre los código de las entidades modeladas en el Diagrama Entidad Relación y el ORM Doctrine 2.0. La realización de la integración demuestra el reconocimiento íntegro de cada anotación Docblock utilizada para las declaraciones de las entidades, atributos por columnas y las relaciones que se establecen entre las entidades.

Ahora se puede revisar la base de datos mediante la interfaz web de phpMyAdmin, para verificar visualmente la realización de la integración con las entidades, columnas y relaciones generadas del caso estudio “Cupón”, luego de ser reconocidas por Doctrine. En las siguientes figuras se muestran cada uno de los aspectos mencionados tras la integración:

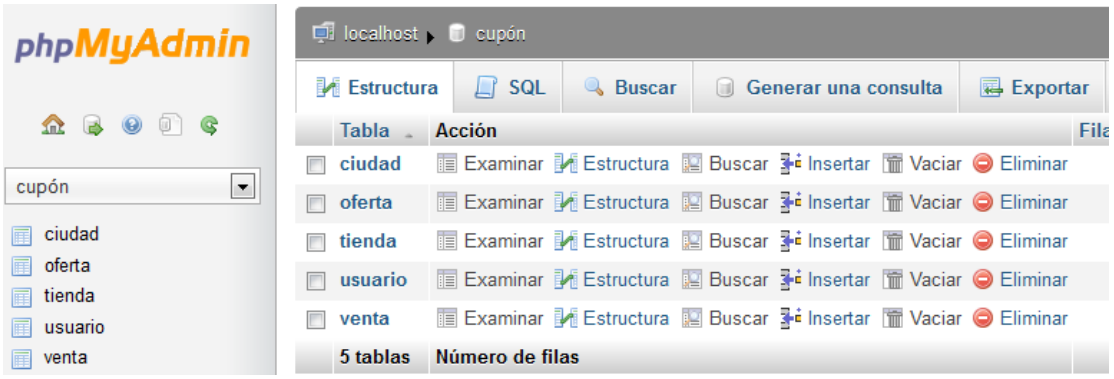


Figura # 19: Entidades modeladas.

Figura # 20: Columnas de la entidad tienda.

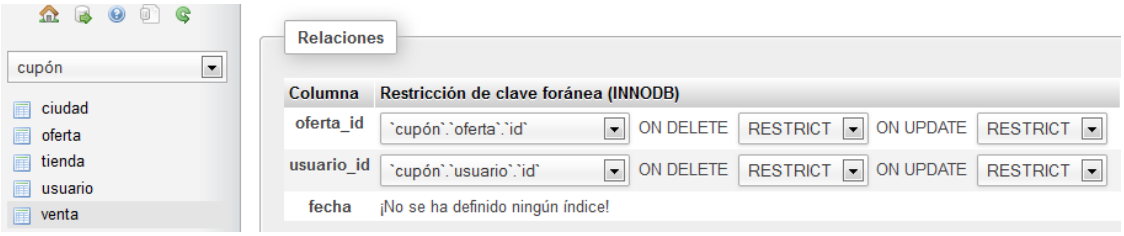


Figura # 21: Relaciones de la entidad venta.

2.8 Evaluación de la solución

Al Verificarse el correcto funcionamiento de la extensión mediante las pruebas y la integración del código generado con el ORM Doctrine 2.0 se procede finalmente a comprobar el cumplimiento del problema de la investigación mediante la solución obtenida. Para demostrar la confiabilidad de lo antes planteado, se desarrollará a continuación una variante de las cuatro fases del método Delphi.

Formulación del problema:

La extensión de la herramienta Visual Paradigm desarrollada, propone la reducción del tiempo de desarrollo del software para la construcción de la capa de acceso a datos con Doctrine 2.0 mediante un Diagrama Entidad Relación previamente modelado en dicha herramienta. Para llevar a cabo la medición del tiempo en la generación de las clases de acceso a datos entre el proceso utilizado actualmente (comando indicado por consola) y el propuesto por la extensión Cader, se hace necesario definir los atributos (A) a evaluar por los especialistas:

A1: Tiempo de confección de las clases de acceso a datos utilizando el generador de entidades de Symfony.

A_C: Tiempo de generación de las clases de acceso a datos mediante los comandos indicados por consola.

A_R: Tiempo empleado en reflejar por cada entidad, los códigos que establecen las relaciones modeladas en el diagrama.

$A1 = A_C + A_R$

A2: Tiempo de confección de las clases de acceso a datos utilizando la extensión Cader.

La evaluación de los atributos será realizada por cada experto mediante el mismo caso de estudio (“Cupón”) utilizado durante la integración con Doctrine 2.0.

Selección de los expertos.

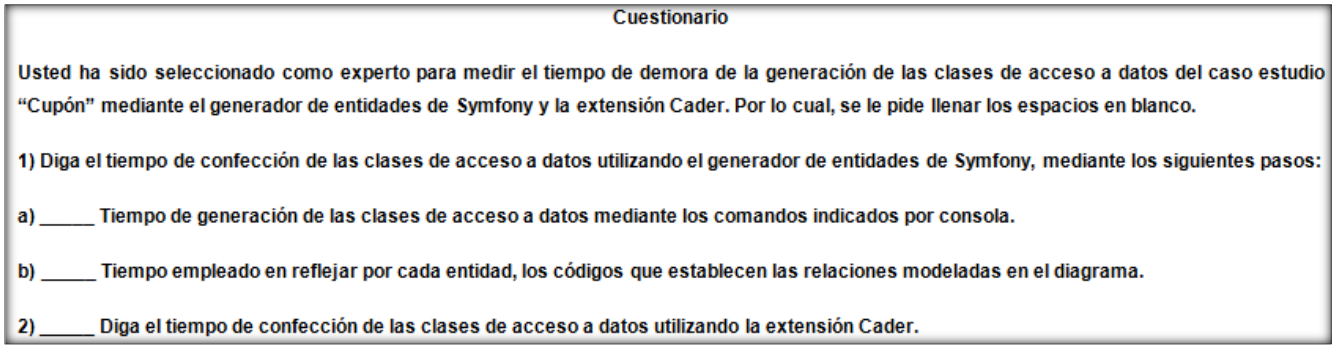
La selección del grupo de expertos fue conformado por siete de los especialistas del Departamento de Soluciones Integrales perteneciente al Centro de Tecnologías de Gestión de Datos. La selección se realizó de forma anónima a personas experimentadas e interesadas a participar en el cuestionario, así como otros aspectos importantes a tener en cuenta, que se enumeran a continuación:

- Graduado de nivel superior.
- Un año de experiencia como mínimo.
- Vinculación al desarrollo de productos informáticos.

- Tener conocimiento y experiencia en las tecnologías utilizadas (Symfony 2.0 y Doctrine 2.0).

Elaboración y envío de los cuestionarios:

Para la medición del tiempo de generación entre las vías mencionadas, se elaboró un cuestionario en el que se les pide a los expertos evaluar cada uno de los criterios definidos en la fase de formulación del problema, a través del caso de estudio “Cupón”. De esta forma se puede obtener el tiempo empleado por cada experto para la creación de las clases de acceso a datos. La siguiente figura muestra cada una de las preguntas efectuadas a los expertos.



Cuestionario

Usted ha sido seleccionado como experto para medir el tiempo de demora de la generación de las clases de acceso a datos del caso estudio “Cupón” mediante el generador de entidades de Symfony y la extensión Cader. Por lo cual, se le pide llenar los espacios en blanco.

1) Diga el tiempo de confección de las clases de acceso a datos utilizando el generador de entidades de Symfony, mediante los siguientes pasos:

a) _____ Tiempo de generación de las clases de acceso a datos mediante los comandos indicados por consola.

b) _____ Tiempo empleado en reflejar por cada entidad, los códigos que establecen las relaciones modeladas en el diagrama.

2) _____ Diga el tiempo de confección de las clases de acceso a datos utilizando la extensión Cader.

Figura # 22: Cuestionario aplicado a los expertos.

Análisis de los resultados:

Luego de la realización de los cuestionarios se confeccionó la siguiente tabla que agrupa las preguntas realizadas y los resultados obtenidos por los expertos. Esta tabla muestra por cada experto la notable diferencia entre el tiempo de confección de las clases de acceso a datos utilizando el generador de entidades de Symfony (A1) y la extensión Cader (A2). La diferencia creada entre estas dos vía y ante el mismo caso de estudio, demuestra que la solución implementada da cumplimiento al problema planteado, al reducir el tiempo en más de 32 minutos como promedio.

Expertos	AC	AR	A1 (A1=AC+AR)	A2
Exp. 1	9:35	19:15	28:50	0:35

Exp. 2	10:05	20:26	30:31	0:27
Exp. 3	8:55	28:49	37:44	0:45
Exp. 4	6:49	26:09	32:58	0:41
Exp. 5	9:28	24:06	33:34	0:39
Exp. 6	8:26	26:45	35:11	0:27
Exp. 7	9:08	20:53	30:01	0:30
Promedio	8:26	23.57	**32:47**	**0:35**

Tabla # 8: Resultados de los cuestionarios

2.9 Conclusiones

En este capítulo se realizó el modelo de implementación con el propósito de mostrar la organización de los componentes de la extensión Cader y las relaciones que se establecen a través de los diagrama de componentes y despliegue confeccionados. Luego se describieron e implementaron las tres etapas del proceso definido en la propuesta de solución y expusieron los estándares de codificación identificados para las declaraciones, tamaño y organización de las líneas de código además del uso de llaves y nomenclaturas utilizadas. Posteriormente se realizó la prueba de Caja Negra mediante los casos de pruebas asociados a casos de uso, para garantizar el correcto funcionamiento de la extensión al suprimir las no conformidades detectadas en cada iteración efectuada. Finalmente se verificó la correcta integración de las clases de acceso a datos generadas con Doctrine 2.0 y comprobó el cumplimiento del problema de la investigación con el uso del método Delphi.

CONCLUSIONES

Con la realización del trabajo investigativo se logró adicionar la funcionalidad que permite generar desde la herramienta Visual Paradigm las clases de acceso a datos para Doctrine 2.0 a partir de un Diagrama Entidad Relación. La implementación de la extensión resolvió los problemas de incompatibilidad que existían con la versión anterior de Doctrine y las consecuencias que traían las tareas repetitivas de conversión entre los distintos paradigmas a la hora de transformar el diseño del sistema, además de la reducción del tiempo de desarrollo en la construcción de la capa de acceso a datos.

Para logar este resultado final se cumplieron los objetivos específicos trazados:

- El estudio de las características para la realización del mapeo objeto relacional con el ORM Doctrine 2.0, permitió conocer sobre la estructura y anotaciones utilizadas para la representación de los metadatos en las entidades.

- El análisis y diseño de la extensión permitió fundamentar y regir el proceso de implementación con los diferentes modelos, requisitos y patrones aplicados.

- La implementación de la extensión se obtuvo según el análisis y diseño realizado y los tres procesos definidos en la propuesta de solución de la generación de forma automática de las clases de acceso a datos a partir de un Diagrama Entidad Relación.

- Las pruebas funcionales permitieron evaluar la extensión en cuatro iteraciones y dar solución a las dificultades encontradas para garantizar una mejor calidad del producto final.

RECOMENDACIONES

A partir del estudio realizado se recomienda:

- Desarrollar nuevas versiones que incluya la generación de las clases de acceso a datos en archivos YAML y XML.

REFERENCIAS BIBLIOGFRÁFICAS

1. Doctrine. *doctrine2-preview-release.* [En línea] [Citado el: 5 de 11 de 2012.] http://www.doctrine-project.org/blog/doctrine2-preview-release.html.
2. doctrine2-released. *Doctrine.* [En línea] [Citado el: 5 de 11 de 2012.] http://www.doctrine-project.org/blog/doctrine2-released.html.
3. **Pacheco, Nacho.** *Doctrine 2 ORM Documentation.* 2011.
4. **Valencia, Néstor Díaz.** Proceso Unificado Abierto. *Centro de Competencia Morfeo.* [En línea] [Citado el: 12 de 10 de 2012.] http://formacion.morfeo-project.org/wiki/index.php/PT3:GPSL:UF6.
5. Lenguaje de Programación. *EcuRed.* [En línea] [Citado el: 10 de 11 de 2012.] http://www.ecured.cu/index.php/Lenguaje_de_Programaci%C3%B3n.
6. Programación Orientada a Objetos. *EcuRed.* [En línea] [Citado el: 10 de 11 de 2012.] http://www.ecured.cu/index.php/Programaci%C3%B3n_Orientada_a_Objetos.
7. 22 ventajas de la programacion orientada a objetos. *Android Linux.* [En línea] 23 de 5 de 2012. [Citado el: 10 de 11 de 2012.] http://android-linux.net/desarrollo/web/22-ventajas-de-la-programacion-orientada-a-objetos-poo.
8. PHP. *EcuRed.* [En línea] [Citado el: 10 de 11 de 2012.] http://www.ecured.cu/index.php/Php.
9. Herramientas CASE. *Visual Paradigm For UML.* [En línea] [Citado el: 15 de 11 de 2012.] http://www.visual-paradigm.com.
10. Visual Paradigm For UML. *Visual Paradigm For UML.* [En línea] [Citado el: 15 de 11 de 2012.] http://www.visual-paradigm.com/product/vpsuite.
11. Dirección de Información. *repositorio_institucional.uci.cu.* [En línea] [Citado el: 15 de 11 de 2012.] http://repositorio_institucional.uci.cu/jspui/bitstream/ident/TD_04358_11/1/TD_04358_11.pdf.
12. Dirección de Información . *repositorio_institucional.uci.cu.* [En línea] [Citado el: 2013 de 6 de 22.] http://repositorio_institucional.uci.cu/jspui/bitstream/ident/TD_02944_10/1/TD_02944_10.pdf.
13. Dirección de Información. *repositorio_institucional.uci.cu.* [En línea] [Citado el: 15 de 2 de 2013.] http://repositorio_institucional.uci.cu/jspui/bitstream/ident/TD_04357_11/1/TD_04357_11.pdf.
14. [En línea] [Citado el: 2013 de 2 de 20.] http://elvex.ugr.es/idbis/db/docs/design/2-requirements.pdf.
15. Requisitos_No_Funcionales. *EcuRed.* [En línea] [Citado el: 2013 de 1 de 20.] http://www.ecured.cu/index.php/Requisitos_No_Funcionales.
16. [En línea] [Citado el: 2013 de 2 de 20.] http://sis.senavirtual.edu.co/infocurso.php?semid=652&areaid=10.
17. [En línea] [Citado el: 2013 de 2 de 20.] http://www.exa.unicen.edu.ar/catedras/modysim/teoria/casos_de_uso_a.pdf.
18. Dirección de Información. *repositorio_institucional.uci.cu.* [En línea] [Citado el: 25 de 2 de 2013.] http://repositorio_institucional.uci.cu/jspui/bitstream/ident/TD_1724_08/1/TD_1724_08.pdf.
19. [En línea] [Citado el: 2013 de 2 de 20.] http://www.slideshare.net/omarzon/modelo-relacional-202868.
20. [aut. libro] Grady Booch, James Rumbaugh Ivar Jacobson. *El Proceso Unificado de Desarrollo de Software.* Madrid : s.n.
21. Dirección de Información. *repositorio_institucional.uci.cu.* [En línea] [Citado el: 10 de 4 de 2013.] http://repositorio_institucional.uci.cu/jspui/bitstream/ident/TD_2457_09/1/TD_2457_09.pdf.
22. **Lapuente, María Jesús Lamarca.** [En línea] [Citado el: 2013 de 3 de 2.] http://www.hipertexto.info/documentos/namespaces.htm.

BIBLIOGRAFÍA

1. Doctrine. *doctrine2-preview-release.* [Online] [Cited: 11 5, 2012.] http://www.doctrine-project.org/blog/doctrine2-preview-release.html.
2. doctrine2-released. *Doctrine.* [Online] [Cited: 11 5, 2012.] http://www.doctrine-project.org/blog/doctrine2-released.html.
3. **Pacheco, Nacho.** *Doctrine 2 ORM Documentation.* 2011.
4. **Valencia, Néstor Díaz.** Proceso Unificado Abierto. *Centro de Competencia Morfeo.* [Online] [Cited: 10 12, 2012.] http://formacion.morfeo-project.org/wiki/index.php/PT3:GPSL:UF6.
5. Lenguaje de Programación. *EcuRed.* [Online] [Cited: 11 10, 2012.] http://www.ecured.cu/index.php/Lenguaje_de_Programaci%C3%B3n.
6. Programación Orientada a Objetos. *EcuRed.* [Online] [Cited: 11 10, 2012.] http://www.ecured.cu/index.php/Programaci%C3%B3n_Orientada_a_Objetos.
7. 22 ventajas de la programacion orientada a objetos. *Android Linux.* [Online] 5 23, 2012. [Cited: 11 10, 2012.] http://android-linux.net/desarrollo/web/22-ventajas-de-la-programacion-orientada-a-objetos-poo.
8. PHP. *EcuRed.* [Online] [Cited: 11 10, 2012.] http://www.ecured.cu/index.php/Php.
9. Herramientas CASE. *Visual Paradigm For UML.* [Online] [Cited: 11 15, 2012.] http://www.visual-paradigm.com.
10. Visual Paradigm For UML. *Visual Paradigm For UML.* [Online] [Cited: 11 15, 2012.] http://www.visual-paradigm.com/product/vpsuite.
11. Dirección de Información. *repositorio_institucional.uci.cu.* [Online] [Cited: 11 15, 2012.] http://repositorio_institucional.uci.cu/jspui/bitstream/ident/TD_04358_11/1/TD_04358_11.pdf.
12. Dirección de Información . *repositorio_institucional.uci.cu.* [Online] [Cited: 6 2013, 22.] http://repositorio_institucional.uci.cu/jspui/bitstream/ident/TD_02944_10/1/TD_02944_10.pdf.
13. Dirección de Información. *repositorio_institucional.uci.cu.* [Online] [Cited: 2 15, 2013.] http://repositorio_institucional.uci.cu/jspui/bitstream/ident/TD_04357_11/1/TD_04357_11.pdf.
14. [Online] [Cited: 2 2013, 20.] http://elvex.ugr.es/idbis/db/docs/design/2-requirements.pdf.
15. Requisitos_No_Funcionales. *EcuRed.* [Online] [Cited: 1 2013, 20.] http://www.ecured.cu/index.php/Requisitos_No_Funcionales.
16. [Online] [Cited: 2 2013, 20.] http://sis.senavirtual.edu.co/infocurso.php?semid=652&areaid=10.
17. [Online] [Cited: 2 2013, 20.] http://www.exa.unicen.edu.ar/catedras/modysim/teoria/casos_de_uso_a.pdf.
18. Dirección de Información. *repositorio_institucional.uci.cu.* [Online] [Cited: 2 25, 2013.] http://repositorio_institucional.uci.cu/jspui/bitstream/ident/TD_1724_08/1/TD_1724_08.pdf.
19. [Online] [Cited: 2 2013, 20.] http://www.slideshare.net/omarzon/modelo-relacional-202868.
20. [book auth.] Grady Booch, James Rumbaugh Ivar Jacobson. *El Proceso Unificado de Desarrollo de Software*. Madrid : s.n.
21. Dirección de Información. *repositorio_institucional.uci.cu.* [Online] [Cited: 4 10, 2013.] http://repositorio_institucional.uci.cu/jspui/bitstream/ident/TD_2457_09/1/TD_2457_09.pdf.
22. Visual Paradigm For UML. *Slideshare.* [Online] 11 15, 2007. [Cited: 10 19, 2012.] www.slideshare.net/vanquishdarkenigma/visual-paradigm-for-uml.
23. Visual Paradigm. *EcuRed.* [Online] [Cited: 12 5, 2012.] http://www.ecured.cu/index.php/Visual_Paradigm.
24. Desarrollo Dirigido por Modelo y Generacion Automática de Código. *Universidad Politécnica de Vaéncia.* [Online] [Cited: 11 26, 2012.] http://www.pros.upv.es/index.php/es/lineas/69-lineaddm.
25. **Novas, Teresa Fernández.** DISEÑO FÍSICO DE LA BASE DE. *ITESCAM.* [Online] 4 6, 200. [Cited: 12 8, 2012.] www.itescam.edu.mx/principal/sylabus/fpdb/recursos/r12292.PDF.
26. **Cedillo, Sergio.** Desarrollo dirigido por modelos. *SG.* [Online] [Cited: 11 26, 2012.] sg.com.mx/content/view/219.

27. **Amaya, Rodrigo.** ¿Qué es ORM? *Sr.Byte.* [Online] 10 2, 2009. [Cited: 10 20, 2012.] http://www.srbyte.com/2009/09/que-es-orm.html.
28. ¿Qué es UML? *DocIRS.* [Online] [Cited: 11 10, 2012.] http://www.docirs.cl/uml.htm.
29. ¿Qué es Doctrine ORM? *TecnoRetales.* [Online] 7 2009. [Cited: 10 20, 2012.] http://www.tecnoretales.com/programacion/que-es-doctrine-orm.
30. OpenUp. *EcuRed.* [Online] [Cited: 11 15, 2012.] http://www.ecured.cu/index.php/OpenUp.
31. Open API. *Visual Paradigm For UML.* [Online] 5 4, 2011. [Cited: 11 20, 2012.] http://knowhow.visual-paradigm.com/openapi/how-to-deploy-plugins-to-vp-application.
32. Object Management Group - UML. *UML.* [Online] 11 28, 2012. [Cited: 12 5, 2012.] www.uml.org.
33. MODELOS DE PROCESOS DE INGENIERIA DE SOFTWARE. *Scribd.* [Online] 5 27, 2012. [Cited: 12 6, 2012.] http://es.scribd.com/doc/94934934/MODELOS-DE-PROCESOS-DE-INGENIERIA-DE-SOFTWARE.
34. Modelado de Sistemas con UML. *tldp.* [Online] [Cited: 11 25, 2012.] http://es.tldp.org/Tutoriales/doc-modelado-sistemas-UML/doc-modelado-sistemas-uml.pdf.
35. Metodologias de desarrollo de Software. *EcuRed.* [Online] [Cited: 11 1, 2012.] http://www.ecured.cu/index.php/Metodologias_de_desarrollo_de_Software.
36. **Lapuente, María Jesús Lamarca.** [Online] [Cited: 3 2013, 2.] http://www.hipertexto.info/documentos/namespaces.htm.
37. **L. Cuaderno, E. Di Lorenzo, A. Gaig, D. García, R. Giandini, L. Nahuel, L. Ocaranza, M. Pinasco, C. Pons, F. Salvatierra.** *Herramientas de soporte al proceso de desarrollo dirigido por modelos y su implementación con DSL Tools.* Buenos Aires : s.n., 2011. 45308ce78aafdf2092719a59d01c.
38. **Longoria, J.F.** La Educación en línea: El uso de la TIC en el PEA. [Online] [Cited: 1 15, 2013.] http://www2.fiu.edu/~longoria/publications/enlinea.pdf.
39. Introducción a Doctrine 2. *chuso.* [Online] 2 2011. http://chusete.es/2011/02/19/introduccion-a-doctrine-2/.
40. Introduccion a object relational mapping. *Web.Ontuts.* [Online] 5 5, 2010. [Cited: 10 20, 2012.] http://web.ontuts.com/tutoriales/introduccion-a-object-relational-mapping-orm/.
41. Estudio comparativo de sistemas de mapeo objeto realcional desarrollado en plataforma open source. *Scribd.* [Online] 7 2011. [Cited: 10 20, 2012.] http://es.scribd.com/doc/70397732/Articulo-Cientifico-ORM.
42. El método Delphi. *Asociacion Colombiana de Psiquiatria.* [Online] 2 3, 2009. [Cited: 5 20, 2013.] http://www.scielo.org.co/scielo.php?pid=S0034-74502009000100013&script=sci_arttext.
43. Educación en Línea. *Universidad de Los Andes.* [Online] [Cited: 1 15, 2013.] http://www.ceidis.ula.ve/index.php?option=com_content&view=section&id=7&Itemid=30.
44. **Silva, Edison Alfredo Suárez.** *Analisis comparativo de los framework.* Quito : s.n., 2011.
45. NetBeans. *EcuRed.* [Online] [Cited: 11 15, 2012.] http://www.ecured.cu/index.php/NetBeans.
46. IDE de Programación. *EcuRed.* [Online] [Cited: 11 15, 2012.] http://www.ecured.cu/index.php/IDE_de_Programaci%C3%B3n.
47. Lenguaje de programación Java. *EcuRed.* [Online] [Cited: 11 10, 2012.] http://www.ecured.cu/index.php/Lenguaje_de_programaci%C3%B3n_Java.
48. Doctrine. *EcuRed.* [Online] [Cited: 11 9, 2012.] http://www.ecured.cu/index.php/Doctrine.
49. API. *EcuRed.* [Online] [Cited: 11 20, 2012.] http://www.ecured.cu/index.php/API.
50. **Carrero, Angel.** Conceptos básicos de ORM. *Programacion en castellano.* [Online] [Cited: 10 25, 2012.] http://www.programacion.com/articulo/conceptos_basicos_de_orm_object_relational_mapping_349.
51. **Villa, Angy Alexandra Borja.** Mapeo de objeto relacional. *Slideshare.* [Online] 10 24, 2010. [Cited: 10 19, 2012.] http://www.slideshare.net/inspirateunaula/mapeo-de-objeto-relacional-5925742.
52. **Calleja, Manuel Arias.** Estándares de codificación. *cisiad.uned.es.* [Online] [Cited: 4 15, 2013.] http://www.cisiad.uned.es/carmen/estilo-codificacion.pdf.