Universidad de las Ciencias Informáticas

"FACULTAD 3"

MIGRACIÓN DE LA CAPA DE ACCESO A DATOS DEL COMPONENTE NOMENCLADOR DE CUENTAS DEL SUBSISTEMA CONTABILIDAD DE XEDRO-ERP.

Trabajo de Diploma para optar por el título de

Ingeniero en Ciencias Informáticas

Autores: Lisandra Pérez Zaldivar
Dargel Compta Dominguez
Tutores: Ing. Yanay Hernández Sosa
Ing. Juan Alberto Caballero Portelles

Ciudad Habana, Cuba

Junio, 2014

"Año 56 de la Revolución"

DECLARACIÓN DE AUTORÍA

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

Para que así conste firmo la presente a los ____ días del mes de _Junio_ del año __2014__.

Lisandra Pérez Zaldivar

_____
Firma del Autor 1

Yanay Hernández Sosa

_____
Firma del Tutor 1

Dargel Compta Dominguez

_____
Firma del Autor 2

Juan Alberto Caballero Portelles

_____
Firma del Tutor 2

"La imaginación es más importante que el conocimiento. El conocimiento es limitado, mientras que la imaginación no."

Albert Einstein

DATOS DE CONTACTO

Tutora: Ing. Yanay Hernández Sosa

Edad: 28 años

Ciudadanía: cubana

Institución: Universidad de las Ciencias Informáticas (UCI)

Título: Ingeniero en Ciencias Informáticas

Categoría docente: Instructor

E-mail: yhsosa@uci.cu

Ingeniero en Ciencias Informáticas, graduada en 2009 en la Universidad de las Ciencias Informáticas. Instructor. Cinco años de experiencia en el desarrollo de software. Cinco años de graduada.

Tutor: Ing. Juan Alberto Caballero Portelles

Edad: 28 años

Ciudadanía: cubano

Institución: Universidad de las Ciencias Informáticas (UCI)

Título: Ingeniero en Ciencias Informáticas

E-mail: jacaballero@uci.cu

Ingeniero en Ciencias Informáticas, graduado en 2011 en la Universidad de las Ciencias Informáticas. Tres años de experiencia en el desarrollo de software. Tres años de graduado.

AGRADECIMIENTOS Y DEDICATORIA

Agradezco:

…ante todo a mi mamá que me ha dado su apoyo incondicional durante toda mi vida en especial en estos cinco años, que ha sido esa llama que me ilumina el camino y que me ha ayudado a levantarme cuando me caigo, que me ha aportado las fuerzas para seguir adelante con su carácter inquebrantable y ese valor para luchar por la vida que no se si algún día llegaré a tener. Quiero agradecerle a mi papá que ha sido mi ejemplo a seguir y sin él no podría realizar este sueño que es su sueño también.

…a mi familia en general y en especial a mi abuelito Pipo que siempre creyó en mí y sé que se siente muy orgulloso de tener a una ingeniera más en la familia.

…a mi tía Tata, mi tío Roly, a mis primos en especial a mi primo Rogelito, Rolandito y Yaimara por ser tan dedicados. También agradecer a mi segunda familia Adelfo, Mary, Arianne y a la más linda de la casa Adrianita.

…a mi compañero de tesis por su arduo trabajo en el estudio y elaboración de este trabajo de diploma. A los tutores, por el apoyo y guía en la realización de la investigación.

…a todos mis compañeros de proyecto por haber hecho de este último año, el mejor. Les agradezco a todos aquellos profesores que me han enseñado a lo largo de estos cinco años contribuyendo con mi formación profesional en especial a Yoansi.

…a mis compañeros de aula que sin ellos las clases hubieran sido aburridas y monótonas, en especial a Randy, Rene Olazabal, Andres, Jose Daniel y muy muy especial a José Miguel y William mis mejores amigos y compañeros.

…a mis amigas que siempre las voy a recordar con mucho cariño y aprecio, en especial a Rosaly y a Jaca. También le quiero agradecer a la persona que siempre ha estado en los momentos buenos y malos, brindándome su mano desinteresadamente, poniéndome su hombro para llorar o brindándome su sonrisa para alegrarme el día, mi hermanita Irene.

Por último a una persona que no por mencionarla al final es menos importante, mi novio y compañero Arian Daniel que ha sabido aguantar mis pesadeces, que en este tiempo ha sido un gran apoyo, ha sabido guiarme y ayudarme en las decisiones más difíciles, gracias por todo: Mi amor "te amo"

Dedicatoria:

Dedicada a mi hermano Roberto: El último que llegó siempre es el más querido. Pensando en ser tu ejemplo son todos mis esfuerzos. Tú eres el mejor.

AGRADECIMIENTOS Y DEDICATORIA

Agradezco:

...A mis padres, por existir, por darme vida, por educarme, por hacer de mí la persona que soy hoy y por tantas cosas más que no cabrían en mil páginas solo mencionarlas.

...A mis abuelos que es lo que más amo, y mi hermano Tito, con el que más me he fajado, pero del que menos quiero estar lejos.

...A todo mi familión, por apoyarme siempre, en especial a Abad que es como un padre para mí, y tonito, mi cuarto hermano.

...A mi segunda familia, que me acogió como un hijo Pilar, Reino, Alfredo, Xiomara y Titi.

...A mi tercera familia, la más diversa, mi grupo, el 3503, especialmente a Carlos Miguel, Gendry, El Pulga y Arturo.

…A los amigos que han perdurado a pesar de la distancia y el tiempo, en especial a Yule que la tengo siempre como una mosca a mi alrededor. A Disnel, Andrés, y a todo el equipo de pelota, mi equipo.

...A los profesores, por los conocimientos que me brindaron, por su paciencia, por su ecuanimidad.

...A mi compi Lisandra, a René, Eloy, Yoandy, Carlos Javier, Inoelkis y a los tutores, por el apoyo y guía en la realización de la investigación, mil gracias Yordano, usted es un crack.

...Al pilar fundamental de mi vida en estos últimos 4 años, mi novia, mi amiga, mi confidente, mi profesora, mi contrincante, te quiero Lianet.

…En fin, a todos los que de una forma u otra me ayudaron con la realización de este trabajo.

Dedicatoria:

…a mis padres, a mis abuelos, a mis hermanos, a mis amigos, y muy especial a Nane, te quiero mucho rani.

Dargel

RESUMEN

Uno de los elementos principales a tener en cuenta para el desarrollo de un sistema informático es el tiempo de respuesta del mismo. CEIGE emplea en el desarrollo del producto Xedro-ERP el marco de trabajo Sauxe, éste utiliza actualmente el marco de persistencia de datos Doctrine en su versión 1.2.2. La generación de ficheros utilizando este ORM[1] presenta una serie de deficiencias que atentan contra el tiempo de respuesta del sistema y la seguridad de sus procesos.

El presente trabajo está centrado en la solución de estas deficiencias mediante la migración a la versión 2.2.0 de Doctrine para mejorar el rendimiento del componente Nomenclador de cuentas del subsistema Contabilidad del producto Xedro-ERP. El objetivo principal que se persigue es lograr una mejor generación de los ficheros de mapeo necesarios para la persistencia de objetos relacionales mediante el uso de este ORM. Para lograrlo se desarrolla el marco teórico donde se estudian las herramientas y tecnologías a utilizar en la migración, con vista al desenlace de las etapas de análisis, implementación y validación de la solución.

Para la validación de la solución se evaluaron los resultados de las pruebas de rendimiento, lo que permitió demostrar que como resultado de la migración se obtuvo un aumento del rendimiento del componente Nomenclador de cuentas.

PALABRAS CLAVE
Sauxe, marco de trabajo, Doctrine, migración, rendimiento, ficheros de mapeo, validación.

[1] ORM: del inglés Object-Relational Mapping. Es la interfaz que traduce la lógica de los objetos a la lógica relacional.

ÍNDICE

ÍNDICE

INTRODUCCIÓN

Las empresas desarrolladoras de software independientemente de su tamaño, capital o presencia en el mercado persiguen un objetivo común: desarrollar software de calidad a un costo adecuado y en tiempo reducido. Este es un reto difícil de lograr debido a que las demandas de los clientes en muchos casos son superiores a las capacidades productivas de las empresas. La solución a este conflicto se ha buscado en todas las direcciones: se han perfeccionado los procesos y las metodologías de desarrollo, se trabaja en la capacitación de los recursos humanos y constantemente se desarrollan nuevas tecnologías y marcos de trabajo.

Cuba no se ha quedado atrás en el avance y desarrollo de las Tecnologías de la Información y las Comunicaciones (TICs), hoy día son numerosas las instituciones que se dedican al desarrollo de software entre las que se encuentra la Universidad de las Ciencias Informáticas (UCI). Dentro de esta radican proyectos productivos que tienen como tarea principal la producción de aplicaciones y servicios informáticos, a partir de la vinculación estudio-trabajo como modelo de formación.

El Centro de informatización de entidades (CEIGE) que pertenece a la Facultad 3 de la UCI desarrolla el producto informático Xedro-ERP haciendo uso del marco de trabajo Sauxe. Este está basado en el marco de trabajo Zend y utiliza en la capa de acceso a datos el Lenguaje de Consulta de Datos (DQL) que implementa Doctrine. Además, utiliza ExtJS en la capa de presentación, por la amplia gama de componentes que se pueden reutilizar y para mostrarle al usuario una interfaz más agradable [1].

Pruebas de rendimiento realizadas sobre Xedro-ERP evidenciaron que se obtenía un rendimiento reducido del subsistema Contabilidad. El uso del marco de persistencia de datos de Doctrine en la versión 1.2.2 provoca que al ejecutar consultas que necesiten manejar gran cantidad de información o que necesiten hacer referencia a varias tablas, se eleven los tiempos de respuesta. Por ejemplo, al realizarse una consulta al sistema gestor de base de datos, el sistema tendrá que convertir la consulta al lenguaje SQL[2] del proveedor usado; enviar la consulta, leer los registros, limpiarlos y convertirlos a objetos [2].

También al ejecutar funciones tales como: cargar los elementos del menú donde no se maneja gran cantidad de datos y la relación entre tablas es mínima, el tiempo de ejecución corresponde

[2] SQL: acrónimo del inglés Structured Query Language.

con mediciones superiores a los 2.3 segundos, considerándose este no aceptable para aplicaciones web ya que el 95% de las transacciones han de tener un tiempo de respuesta inferior a 2 segundos [3] y las funciones para cargar los elementos del menú apenas llegan al 60% de las transiciones del sistema.

Igualmente, debido a que en Xedro-ERP existen varios componentes y que la base de datos trabaja con esquemas independientes relacionados a cada uno de estos, al realizar llamadas a funciones entre varios componentes el marco de trabajo de persistencia de datos Doctrine 1.2.2 tiene que abrir una nueva conexión, realizar la consulta y cerrar la conexión por cada llamada que se realice siendo este un procedimiento repetitivo. Este proceso, ligado a otras dificultades entra las que se encuentra la forma en que se maneja la herencia donde se debe hacer referencia a numerosas tablas para obtener los datos, provoca que la aplicación tenga un rendimiento reducido.

Otro inconveniente que presenta el uso de esta versión de Doctrine es que a partir del 1 de junio de 2011 se dejó de dar soporte [4], lo que conlleva a que no se corrijan vulnerabilidades en el código que puedan atentar contra el correcto funcionamiento de los sistemas que lo utilizan. Por lo anteriormente expuesto la Subdirección de producción de CEIGE, respaldado en la decisión de los arquitectos principales de Xedro-ERP, informa a la línea de desarrollo Contabilidad la tarea de migrar paulatinamente sus componentes a Doctrine 2.2.0 y debido al flujo del uso del subsistema se decide comenzar por el componente Nomenclador de Cuentas.

De acuerdo con la problemática planteada se identifica como **problema a resolver:** ¿Cómo mejorar el rendimiento del componente Nomenclador de Cuentas del subsistema Contabilidad de Xedro-ERP?

Por tanto, para el desarrollo de la investigación se precisó como **objetivo general**: Migrar el componente Nomenclador de Cuentas del subsistema Contabilidad de Xedro-ERP haciendo uso de la versión 2.2.0 de Doctrine para mejorar el rendimiento del mismo.

Se define como **objeto de estudio**: la migración en el proceso de desarrollo de software y como **campo de acción**: la migración de la capa de acceso a datos del componente Nomenclador de Cuentas del subsistema Contabilidad de Xedro-ERP.

Para darle cumplimiento al objetivo general se desglosan los siguientes **objetivos específicos:**

1. Fundamentar la investigación mediante la elaboración del Marco Teórico para sustentar los conceptos y la propuesta de desarrollo del componente.

2. Realizar el diseño e implementación del componente Nomenclador de Cuentas para mejorar el rendimiento del mismo.

3. Medir el rendimiento del componente Nomenclador de Cuentas para validar la migración realizada.

La investigación parte de la siguiente **Idea a defender**: Si se migra el componente Nomenclador de Cuentas del subsistema Contabilidad de Xedro-ERP haciendo uso de la versión 2.2.0 de Doctrine mejorará el rendimiento del mismo.

En el desarrollo de la investigación se han utilizado los siguientes métodos científicos:

Métodos Teóricos: los métodos teóricos son factibles en el estudio de las características poco observables del objeto de investigación. Dentro de este grupo se utilizan:

- ✓ **Histórico - Lógico:** Permite realizar un estudio del estado del arte de la versión 1.2.2 del ORM Doctrine y la versión 2.2.0. Además, facilita el análisis evolutivo de dicha herramienta en materia de diseño y su tendencia actual.
- ✓ **Analítico - Sintético:** Se utilizó para el procesamiento de toda la información relacionada con la investigación, analizando los documentos que permitieron extraer los elementos más significativos relacionados con el objeto de estudio.

Métodos Empíricos: los métodos empíricos tienen gran importancia pues permiten efectuar el análisis preliminar de la información, así como verificar y comprobar las concepciones teóricas. Dentro de este grupo se utiliza:

- ✓ **Entrevista:** Este método brinda apoyo a la incorporación de conocimientos mediante las entrevistas planificadas efectuadas a los especialistas de áreas de CEIGE.
- ✓ **Observación:** Con su utilización se centralizó toda la atención en la situación actual existente en el marco de trabajo Sauxe y en las inconvenientes de la utilización de la versión 1.2.2 de Doctrine.

CAPÍTULO I: FUNDAMENTACIÓN TEÓRICA

1.1. Introducción

En este capítulo se reflejan los temas fundamentales que sustentan la investigación. En él se abordan varios conceptos asociados a los marcos de trabajo de persistencia de datos, específicamente Doctrine. Debido a que el problema a resolver está enmarcado en las soluciones desarrolladas sobre el marco de trabajo Sauxe, se seleccionan las tecnologías y herramientas informáticas utilizadas para el mismo.

1.2. Proceso de Migración

Los procesos de migración informática se refieren en su mayoría a los procesos asociados a las migraciones de los sistemas informáticos, desde plataformas rentadas o privativas, hacia las nuevas tendencias del software libre (Da Rosa, 2008).

Desde el punto de vista informático se puede decir que la migración de software es el procedimiento mediante el cual se cambian los sistemas actuales de información de una organización, por otros productos o versiones que se adapten mejor a las necesidades actuales y/o futuras de dicha organización.

Migrar es también elevar una versión de un producto informático a otra de más alto nivel, o bien el movimiento de una arquitectura a otra, por ejemplo, de un sistema centralizado a otro con una estructura basada en el modelo cliente/servidor.

En el presente trabajo se realizará la migración de la versión 1.2.2 del marco de persistencia de datos Doctrine a la versión 2.2.0 por el conjunto de beneficios que posee la utilización de esta nueva versión del ORM, entre ellas las mejoras en cuanto a la generación de ficheros de mapeo y el buen rendimiento en tiempo de ejecución.

1.3. Marco de trabajo de persistencia de datos

Es una estructura de software orientada a mejorar la productividad a través de la reutilización de código. Es un conjunto de clases creadas para apoyar la escritura de código en un contexto determinado. “Un marco de trabajo de persistencia es una biblioteca de clases que facilita la tarea del programador al permitirle guardar objetos en bases de datos relacionales de manera lógica y eficiente, de otra manera se debería hacer manualmente, y este es un proceso tedioso, repetitivo y propenso a errores” [5].

¿Qué es un ORM?

Un ORM (Object Relation Mapper) es una técnica de programación que nos permite convertir datos entre el sistema de tipos utilizado en un lenguaje de programación orientado a objetos y el utilizado en una base de datos relacional, es decir, las tablas de nuestra base de datos pasan a ser clases y los registros objetos que podemos manejar con facilidad [6].

Utilizar un ORM tiene una serie de ventajas que nos facilitan tareas comunes y de mantenimiento:

Reutilización: La principal ventaja que aporta un ORM es la reutilización de códigos permitiendo llamar a los métodos de un objeto de datos desde distintas partes de la aplicación e incluso desde diferentes aplicaciones.

Encapsulación: La capa ORM encapsula la lógica de los datos pudiendo hacer cambios que afectan a toda la aplicación únicamente modificando una función.

Portabilidad: Utilizar una capa de abstracción nos permite cambiar en mitad de un proyecto de una base de datos MySQL a una Oracle sin ningún tipo de complicación. Esto es debido a que no utilizamos una sintaxis MySQL, Oracle o SQLite para acceder a nuestro modelo, sino una sintaxis propia del ORM utilizado que es capaz de traducir a diferentes tipos de bases de datos.

Seguridad: Los ORM suelen implementar mecanismos de seguridad que protegen nuestra aplicación de los ataques como inyecciones SQL y Secuencia de comandos a sitios cruzados.

Mantenimiento del código: Gracias a la correcta ordenación de la capa de datos, modificar y mantener nuestro código es una tarea sencilla.

¿Qué es Doctrine?

Doctrine es un ORM para la persistencia de datos que trabaja con lenguaje de programación PHP 5.2.3 o superior, situándose sobre su DBAL[3]. Es uno de los más conocidos que interactúa con este lenguaje y su utilización brinda facilidades a los desarrolladores a la hora de su trabajo con los datos, automatizando la generación de ficheros que responden a las tablas relacionales de la base de datos [7].

[3] DBAL: del inglés Database Abstraction Layer o Capa de Abstracción de la base de datos.

Principales características de Doctrine

Una característica de Doctrine es el bajo nivel de configuración que necesita para empezar un proyecto. Doctrine puede generar clases a partir de una base de datos existente y después el programador puede especificar relaciones y añadir funcionalidad extra a las clases autogeneradas. No es necesario generar o mantener complejos esquemas XML de base de datos como en otros frameworks.

Doctrine se divide en dos capas fundamentales que interactúan en conjunto, la DBAL y la compuesta por el ORM reflejadas en la siguiente imagen.

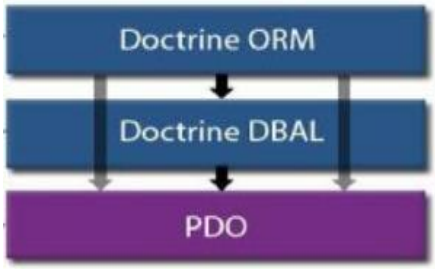


Figura1. Estructura lógica del marco de trabajo de persistencia Doctrine [5].

Este marco de trabajo tiene la facultad de construir consultas a la base de datos relacional utilizando un lenguaje OO[4] denominado DQL[5]. Además, implementa un patrón CRUD (Crear, Recuperar, Actualizar y Eliminar) para operaciones comunes, ya sea desde crear un nuevo registro o actualizar los registros existentes y dentro de los ORM que dispone PHP.

1.3.1. Doctrine 1.2.2

Doctrine 1.2.2 es una implementación de ActiveRecord (registro activo), que propone hacer la persistencia en el mismo objeto de negocio como se muestra en la **Figura 2**, esta característica influye negativamente en su rendimiento debido al alto acoplamiento de los objetos en la capa de negocio, otra inconveniente es que utiliza métodos mágicos y estáticos que hace más difícil la comprensión de sus funciones.

[4]OO: Orientado a Objetos.

[5]DQL: del inglés Doctrine Query Language.

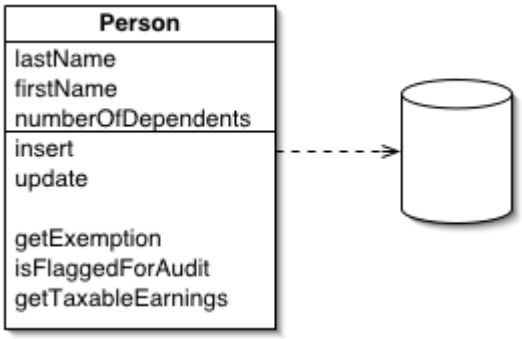


Figura2. Representación de un objeto de negocio en Doctrine 1.2.2.

Otro de los aspectos analizados de Doctrine 1.2.2 se refiere al manejo de los datos donde al realizar una consulta al sistema gestor de base de datos, el sistema tiene que convertir la consulta al lenguaje SQL, enviar la consulta, leer los registros, limpiarlos y convertirlos a objetos.

Esta versión también genera 2 ficheros de mapeo, el fichero Base que extiende de Doctrine_Record y otro fichero que extiende del fichero Base lo que trae consigo un alto acoplamiento por el grado de dependencia entre una clase y otra.

Los estudios concernientes al rendimiento realizados sobre el componente Nomenclador de cuentas del subsistema Contabilidad de Xedro-ERP evidenciaron que el uso de la versión 1.2.2 de Doctrine provocaba un rendimiento reducido del sistema, arrojando tiempos de respuesta superior a los 10 segundos al insertar y/o consultar grandes volúmenes de datos o al realizar consultas sobre múltiples tablas en la base de datos.

1.3.2. Doctrine 2.2.0

Doctrine 2.2.0 es una implementación del patrón DataMapper (mapa de datos), no fuerza a las clases a extender de Active Record, ni contiene detalles sobre el modelo relacional, como son las claves externas **figura 3**.

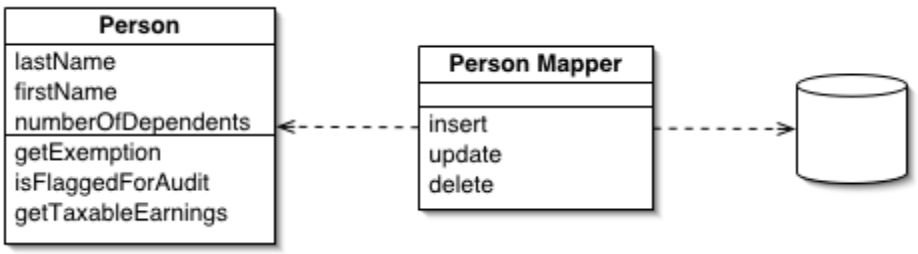


Figura3: Modelo negocio Doctrine2

Doctrine 2 es la versión superior de este marco de trabajo de persistencia de datos para PHP 5.3.0+ que proporciona persistencia transparente de objetos PHP. Se sitúa en la parte superior de una poderosa capa de abstracción de base de datos [8].

Doctrine 2 implementa el diseño Mapa de Datos proponiendo que los objetos de negocio sean POCO[6], o sea que no tengan nada de código que los acople a una tecnología de persistencia, ya que no es su función manejarla. Los objetos de negocio solo deben preocuparse de cumplir con el modelo del dominio del diagrama de clases y con las asociaciones pertinentes.

Dentro de las ventajas que traen consigo los cambios es 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 a continuación:

- 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) para apoyar a diferentes proveedores y brinda facilidades a 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 2.2.0) no están obligados a extender más de una clase base abstracta. Doctrine 2.2.0 permite el uso de simples objetos de PHP.
- El UnitOfWork es el patrón más céntrico de Doctrine 2.2.0. En lugar de llamar a guardar () o borrar () en Doctrine_Record ahora se pasan los objetos al encargado de asignar los datos denominado EntityManager 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 1.2.2 en términos de rendimiento y facilidad de uso.
- No hay medidas de generación de código de YAML5 a PHP. YAML, XML6, PHP y Anotaciones Docblock 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 2.2.0 utilizar los metadatos en tiempo de ejecución sin depender de la generación de código.

[6]POCO: Componentes Portables (en inglés Portable Components).

- 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.
- No posee código que lo acople a una tecnología de persistencia, ya que no es su función manejarla. Los objetos de negocio solo deben preocuparse de cumplir con el modelo del dominio del diagrama de clases con las asociaciones pertinentes, eliminándose el innecesario instanciado de la conexión.

Doctrine 2.2.0 sin almacenamiento en caché es más rápido que Doctrine 1.2.2 con el caché. La memoria caché de consulta en Doctrine 2.2.0 es más eficaz y elimina casi completamente toda la sobrecarga de DQL. La caché de consultas es lo que permite ofrecer esta potente abstracción que es inmensamente flexible.

Rendimiento del uso de Doctrine 2.2.0: [8]

- ✓ El uso de PHP 5.3.0+ brinda un buen rendimiento.
- ✓ Mejor hidratación de objetos y algoritmo optimizado.
- ✓ Nuevas implementaciones de consultas y un mejor resultado del caché.
- ✓ El código más explícito y menos mágico, como resultados: el código mejor y más rápido.

Compatibilidad con el gestor de base de datos PostgreSQL:

Doctrine 2.2.0 soporta múltiples gestores de base de datos tales como MySQL, PostgreSQL, SQLite, MSSQL y Oracle.

Uso del Lenguaje PHP:

Este está escrito en PHP pero solo es compatible con las versiones superiores a 5.3.0.

Compatibilidad con el marco de trabajo Sauxe:

La compatibilidad con Sauxe está garantizada debido a que este utiliza en su capa de acceso a datos la versión 1.2.2 de Doctrine.

Comunidad que respalde su soporte:

Cuenta con una amplia comunidad y colaboradores alrededor de todo el mundo, es uno de los marcos de trabajo de persistencia más usados y con mayor prestigio a escala mundial.

Buenas prácticas:

- ✓ No utilizar propiedades públicas en las entidades.
- ✓ Limitar las relaciones tanto como sea posible.
- ✓ Se deben evitar las asociaciones bidireccionales, si es posible.
- ✓ Eliminar las asociaciones que no sean esenciales.

1.3.3. Ventajas de Doctrine 2.2.0

Una de las ventajas de utilizar estas capas de abstracción de objetos/relacional es que evita utilizar una sintaxis específica de un sistema de bases de datos concreto. Esta capa transforma automáticamente las llamadas a los objetos en consultas SQL optimizadas para el sistema gestor de bases de datos que se está utilizando en cada momento. De esta forma, es muy sencillo cambiar a otro sistema de bases de datos completamente diferente en mitad del desarrollo de un proyecto. Estas técnicas son útiles por ejemplo cuando se debe desarrollar un prototipo rápido de una aplicación y el cliente aún no ha decidido el sistema de bases de datos que más le conviene. El prototipo se puede realizar utilizando SQLite y después se puede cambiar fácilmente a MySQL, PostgreSQL u otro gestor cuando el cliente se haya decidido.

La capa de abstracción utilizada encapsula toda la lógica de los datos. El resto de la aplicación no tiene que preocuparse por las consultas SQL y el código SQL que se encarga del acceso a la base de datos es fácil de encontrar. Los desarrolladores especializados en la programación con bases de datos pueden localizar fácilmente el código.

Utilizar objetos en vez de registros y clases en vez de tablas es otra de las ventajas de Doctrine 2.2.0, debido a que se pueden definir nuevos métodos de acceso a las tablas. Por ejemplo, si se dispone de una tabla llamada Cliente con 2 campos, Nombre y Apellido, puede que sea necesario acceder directamente al nombre completo (NombreCompleto).

Con la programación orientada a objetos, este problema se resuelve añadiendo un nuevo método de acceso a la clase Cliente de forma sencilla.

La principal ventaja de este ORM es el rendimiento en ejecución y en la forma tan concisa en la que se pueden escribir consultas muy complejas.

Resumen del estudio del marco de trabajo Doctrine 2.2.0:

El estudio del marco de trabajo para la persistencia de datos Doctrine 2.2.0 demostró las potencialidades que representa el uso del mismo, en operaciones comunes como crear, eliminar, buscar y actualizar un objeto del sistema, es superior en cuanto al tiempo de respuesta obtenido con la versión 1.2.2.

1.3.4. Procedimiento para la migración

El procedimiento a seguir para realizar la migración lo constituyó la investigación titulada Procedimiento para la actualización de la CAD del MT Sauxe a D2 elaborada por la Ingeniera Lisandra Cordero y el Ingeniero Inoelkis Velázques, en la misma se definen 3 etapas(Inicial, Integración e Implementación), mostrando una secuencia lógica y organizada para su ejecución. En el trabajo antes mencionado se definen las directrices mediante la cual los proyectos implementados en Sauxe pueden realizar la migración de forma exitosa.

Etapa Inicial

Se realiza un estudio de la arquitectura del marco de trabajo Sauxe, donde se valora el contenido de cada una de las capas que contiene y la estructura de carpetas que utiliza este para representar cada uno de sus componentes.

Etapa de Integración

En esta etapa se realizará la integración de Doctrine 2 con el marco de trabajo Sauxe y como consecuencia se hace necesaria la creación de directorios donde se añaden las clases y funcionalidades para la integración.

Etapa de Implementación

En esta etapa se aplicarán nuevos cambios propuesto por Doctrine 2.2.0 en el diseño de la arquitectura base del componente Nomenclador de cuentas.

1.4. Tecnologías y herramientas para el desarrollo

La correcta selección de las herramientas y tecnologías que se utilizan en el desarrollo de un software se traduce en ahorro de tiempo y trabajo dentro de cualquier proyecto. A continuación se realiza una breve descripción de las tecnologías y herramientas que serán utilizadas.

1.4.1 Marco de trabajo Sauxe v2.0

Sauxe fue creado por el departamento de Tecnología del centro CEIGE de la Universidad de las Ciencias Informáticas. Contiene un conjunto de componentes reutilizables que provee la estructura genérica y el comportamiento para una familia de abstracciones, logrando una mayor estandarización, flexibilidad, integración y agilidad en el proceso de desarrollo [8]. Con la utilización del mismo desde el inicio del desarrollo de un proyecto se tienen garantizado aspectos como la seguridad, la multi-entidad, el multi-tema, el multi-idioma, la auditoría, la integración, la interoperabilidad y la concurrencia. Por las ventajas que proporciona, mencionadas anteriormente, es utilizado en más de 20 proyectos de la UCI y 10 entidades desarrolladoras de software [9].

Siguiendo el paradigma de soberanía tecnológica por el cual apuesta el país, se utilizan las siguientes tecnologías.

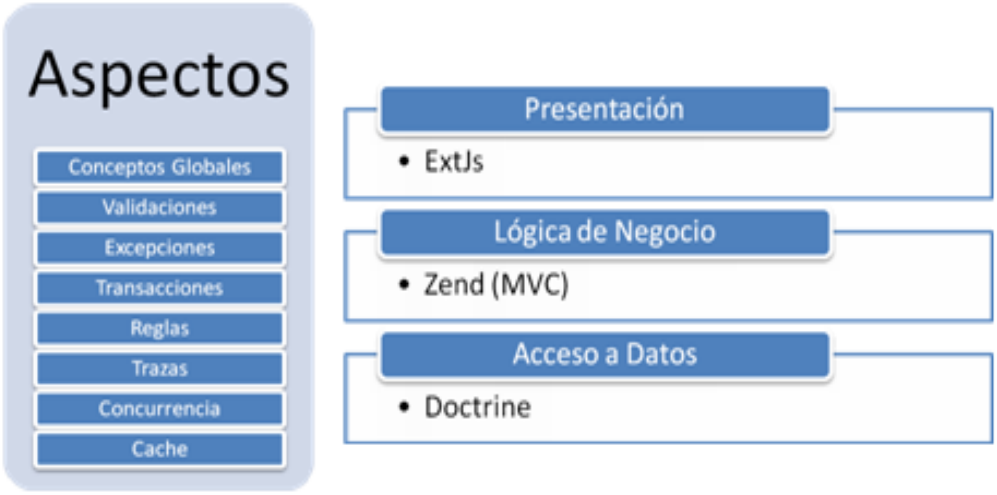


Figura4. Arquitectura de Sauxe [10].

ExtJs 2.2

ExtJs es una biblioteca de JavaScript para el desarrollo de aplicaciones web interactivas usando tecnologías como AJAX[7], DHTML[8] y DOM[9]. Incluye un alto rendimiento, interfaces de

[7] AJAX: Asynchronous JavaScript and XML (JavaScript asíncrono y XML).

[8] DHTML: Dynamic Hypertext Markup Language (Lenguaje de Marcado de Hipertexto Dinámico).

[9] DOM: Document Object Model (Modelo de Objetos del Documento).

usuario personalizables, bien diseñado y extensible modelo de componentes, una interfaz intuitiva y fácil de utilizar con licencias de código abierto y comercial. Brinda soporte para construir interfaces gráficas complejas y dinámicas y comunicar datos de forma asíncrona con el servidor.

Una de las ventajas de utilizar ExtJs es que permite crear aplicaciones complejas utilizando componentes predefinidos, así como un manejador de capas similar al que provee Java Swing, gracias a esto provee una experiencia consistente sobre cualquier navegador, evitando el tedioso problema de validar que el código escrito funcione bien en cada uno.

Usar ExtJs permite tener además otros beneficios entre los que se encuentran:

- ✓ Existe un balance entre cliente–servidor. La carga de procesamiento se distribuye, permitiendo que el servidor, al tener menor carga, pueda manejar más clientes al mismo tiempo.
- ✓ Comunicación asíncrona. Este tipo de aplicación puede comunicarse con el servidor sin necesidad de estar sujeta a un clic o una acción del usuario, dándole la libertad de cargar información sin que el cliente se dé cuenta.
- ✓ Eficiencia de la red. El tráfico de red puede disminuir al permitir que la aplicación elija qué información desea transmitir al servidor y viceversa.

Zend 1.11

Zend es un marco de trabajo de código abierto para desarrollar aplicaciones y servicios web con PHP5. Este es una implementación que usa un código totalmente orientado a objetos. Su estructura de componentes es única; cada componente está construido con una baja dependencia de otros componentes. Esta arquitectura débilmente acoplada, permite a los desarrolladores utilizar los componentes por separado. A menudo se refiere a este tipo de diseño como “uso a voluntad” (del inglés “use at will").

Aunque se pueden utilizar de forma individual, los componentes de la biblioteca estándar de Zend conforman un potente y extensible marco de trabajo de aplicaciones web al combinarse. Este ofrece un gran rendimiento y una robusta implementación del patrón Modelo-Vista-Controlador (MVC), una abstracción de base de datos fácil de usar y un componente de formularios que implementa la prestación de formularios HTML[10] [10].

[10]HTML: siglas de HyperText Markup Language (Lenguaje de Marcado de Hipertexto).

1.4.2. Lenguajes de Programación para la Web

Hypertext Preprocessor 5.3.0+ (PHP)

PHP es un lenguaje de programación interpretado, originalmente diseñado para la creación de páginas web dinámicas. Sigue un estilo clásico ya que es un lenguaje de programación con variables, sentencias condicionales, bucles y funciones.

Contiene una biblioteca nativa de funciones sumamente amplia e incluida, no requiere definición de tipos de variables ni manejo detallado del bajo nivel, presenta mejoras de rendimiento, es un lenguaje multiplataforma. No es un lenguaje de marcas y su principal meta es permitir rápidamente a los desarrolladores la generación dinámica de páginas. Soporta el uso de otros servicios que usen protocolos como SNMP[11] y HTTP[12] [11].

1.4.3. Subversion v1.5

Subversion es un sistema de control de versiones de código abierto. Este maneja ficheros y directorios a través del tiempo. Posee un árbol de ficheros en un repositorio central y el mismo funciona como un servidor de ficheros ordinario, exceptuando que guarda todos los cambios hechos a sus ficheros y directorios. Esto permite recuperar versiones antiguas de los datos, o examinar la historia de cómo cambiaron sus datos.

Subversion puede acceder al repositorio a través de redes, lo que permite que pueda ser utilizado por personas en diferentes equipos. A cierto nivel, la capacidad para que varias personas puedan modificar y administrar el mismo conjunto de datos desde sus respectivas ubicaciones fomenta la colaboración [12].

1.4.4. RapidSVN v0.12.0

RapidSVN es un cliente gráfico multiplataforma que se distribuye bajo licencia pública. Es bastante simple, pues proporciona una interfaz fácil de usar para las características del sistema de control de versiones Subversion. Es un sistema eficiente, debido a su simplicidad para los principiantes, pero lo suficientemente flexible como para aumentar la productividad para los usuarios con experiencia. Es portable, rápido, multilingüe y posee soporte completo para el estándar de codificación denominado Unicode [13].

[11]SNMP: Protocolo simple de Administración de Red.

[12]HTTP: Protocolo de transferencia de hipertexto.

1.4.5. Servidor de Base de Datos

PostgreSQL v8.3

PostgreSQL es un Sistema de Gestión de Bases de Datos Objeto-Relacionales (ORDBMS). Este está ampliamente considerado como el sistema de bases de datos de código abierto más avanzado del mundo. El proyecto PostgreSQL sigue actualmente un activo proceso de desarrollo a nivel mundial gracias a un equipo de desarrolladores y contribuidores de código abierto. Este proporciona un gran número de características que normalmente solo se encontraban en las bases de datos comerciales tales como DB2 u Oracle. La siguiente es una breve lista de algunas de esas características [14]:

- ✓ DBMS Objeto-Relacional: PostgreSQL aproxima los datos a un modelo objeto-relacional, y es capaz de manejar complejas rutinas y reglas.
- ✓ Altamente_Extensible: PostgreSQL soporta operadores, funciones, métodos de acceso y tipos de datos definidos por el usuario.
- ✓ Lenguajes Procedurales: PostgreSQL tiene soporte para lenguajes procedurales internos, incluyendo un lenguaje nativo denominado PL/pgSQL.

1.4.6. Servidor de aplicaciones web

Apache 2.2

Apache es un servidor web gratuito, potente y que ofrece un servicio estable y sencillo de mantener y configurar.

Entre sus principales características se encuentran [15]:

- ✓ Corre en una multitud de Sistemas Operativos, lo que lo hace prácticamente universal.
- ✓ Es una tecnología gratuita de código fuente abierto.
- ✓ Es un servidor altamente configurable de diseño modular. Actualmente existen muchos módulos para Apache que son adaptables a este.
- ✓ Permite personalizar la respuesta ante los posibles errores que se puedan dar en el servidor.
- ✓ Tiene una alta configurabilidad en la creación y gestión de logs.

1.4.7. Entorno de Desarrollo Integrado (IDE)

Un entorno o ambiente integrado de desarrollo (Integrated Development Environment) es un programa compuesto por un conjunto de herramientas para un programador, que puede dedicarse a un solo lenguaje de programación o utilizarse para varios. Estos suelen incluir en una misma suite, un buen editor de código, administrador de proyectos y archivos, enlace transparente a compiladores y debuggers e integración con sistemas controladores de versiones o repositorios [20].

NetBeans v7.4

La plataforma de NetBeans es un producto libre y gratuito sin restricciones de uso que permite a los desarrolladores crear rápida y fácilmente aplicaciones de gran calidad. Es principalmente un marco de trabajo de aplicación genérica para las aplicaciones de escritorio en Java aunque también puede usarse para aplicaciones web reconociendo lenguajes como PHP, JavaScript, C y C++. El beneficio principal de esta plataforma es su arquitectura modular predefinida. Los beneficios secundarios son las soluciones reusables en combinación con las herramientas proporcionadas por su SDK[13], NetBeans IDE[14][16].

Dentro de sus características se pueden encontrar:

- ✓ Mejoras en el editor de código.
- ✓ Características visuales para el desarrollo web.
- ✓ Soporte para PHP.
- ✓ Permite desarrollar aplicaciones de escritorio, web, Mobile, Enterprises.

Para el desarrollo de la investigación se selecciona el NetBeans por sus características, entre ellas que brinda soporte para PHP y además presenta mejoras para el mismo. Además este IDE es mucho más ágil y a la vez robusto, contiene más ayuda en línea, reconocimiento de sintaxis y todo lo que provee la última versión de este lenguaje.

[13]SDK: Kit de desarrollo de software o SDK (siglas en inglés de Software Development Kit).

[14]IDE: Un entorno de desarrollo integrado (de su acrónimo en inglés de Integrated Development Environment).

1.4.8. Navegador Web

Mozilla Firefox 25.0

Mozilla Firefox es un navegador de código abierto, multiplataforma, basado en el código de base de Mozilla que proporciona una navegación más rápida, segura y eficiente que otros navegadores. Entre sus principales características se encuentran:

- ✓ Velocidad: las páginas se abren en menor tiempo y se navega con comodidad en ellas.
- ✓ Seguridad: es un software de código abierto, esto permite que las fallas de seguridad se corrijan al instante, tienen un grupo de colaboradores encargados de esta cuestión.
- ✓ Se pueden usar varios perfiles de usuario con configuraciones diferentes para cada uno.

Este navegador consta además de un innumerable conjunto de extensiones y temas dentro de su paquete de complementos. En el desarrollo de aplicaciones web es muy utilizado el complemento firebug, a través del cual los desarrolladores pueden llevar un control del tráfico de información desde el cliente hasta el servidor, brindando grandes facilidades al conectar la programación de la capa de presentación con la del negocio.

Resumen del estudio de las herramientas:

Las herramientas antes mencionadas son las definidas en el documento *"Vista de Entorno de Desarrollo Tecnológico del proyecto Sauxe_v2.2"* y a las cuales se les realizó un estudio para destacar algunas de sus potencialidades.

1.5. Validación de la solución

JMeter Apache 2.3.1

Herramienta Java dentro del proyecto de Jakarta, que permite realizar pruebas de rendimiento y pruebas funcionales sobre aplicaciones web y bases de datos. JMeter se destaca por su versatilidad, estabilidad y por ser de uso gratuito.

El JMeter muestra los resultados de las pruebas en una amplia variedad de informes y gráficas. Además facilita una rápida detección de los cuellos de botella existentes debido al tiempo de respuesta excesivo [17].

1.6. Conclusiones parciales

Se realizó un estudio del marco de trabajo Doctrine destacando las ventajas que presenta la versión 2.2.0 en comparación con la versión 1.2.2. Entre ellas su mejor rendimiento en ejecución y la forma tan concisa en la que se pueden escribir consultas muy complejas.

Se determinó que la aplicación debe ser desarrollada basándose en las siguientes tecnologías y herramientas: marco de trabajo Sauxe, gestor de base de datos PostgreSQL, servidor web Apache, entorno de desarrollo NetBeans y como navegador web Mozilla Firefox, debido a que son las más idóneas dentro de la plataforma tecnológica que ofrece el centro CEIGE teniendo en cuenta las características del sistema.

CAPÍTULO II: DESARROLLO DE LA MIGRACIÓN

2.1. Introducción

En el presente Capítulo se describe la solución propuesta que consiste en la migración de la capa de acceso a datos del componente Nomenclador de Cuentas del subsistema Contabilidad. Se ofrece una explicación de la arquitectura empleada por el ORM Doctrine y sus librerías, especificando además la nueva estructura de carpetas para la versión 2.2.0 de este marco de trabajo de persistencia de datos y las configuraciones necesarias en los ficheros de Xedro-ERP para garantizar la integración de la nueva solución a través de los pasos de esta migración.

2.2. Etapa Inicial

Inicialmente se lleva a cabo el estudio de la arquitectura del marco de trabajo Sauxe, donde se valora el contenido de cada una de las capas que contiene y la estructura de carpetas que utiliza este para representar cada uno de sus componentes.

2.2.1. Estudio de la arquitectura de Sauxe

La arquitectura de Sauxe está basada en capas aunque en una de sus capas implementa el patrón Modelo Vista Controlador. Está compuesta básicamente por cinco niveles o capas estructurales [18]:

1. Capa de Presentación: En esta capa se emplean las facilidades que brinda el marco de trabajo ExtJs para la construcción de interfaces agradables a la vista de los usuarios. ExtJs centra su desarrollo en tres componentes fundamentales JS-File, CSS-File, Client-page y Server-Page.
2. Capa de Control o Negocio: En esta capa se encuentran las reglas obtenidas a partir del análisis funcional del proyecto. Responde a eventos, usualmente acciones del usuario e invoca peticiones al modelo cuando se hace alguna solicitud sobre la información y notifica a la vista. En esta capa también estarán los aspectos fundamentales del sistema así como el encargado del tratamiento de la seguridad a nivel de aplicación Web.
3. Capa de Acceso a Datos: En esta capa estará presente el marco de persistencia de datos de Doctrine, para la comunicación con el servidor de datos mediante el protocolo

PDO[15], también estará un Persistidor de Configuración que es el encargado de comunicarse vía XML con los ficheros de configuración del sistema denominado FastResponse.

4. Capa de Datos: En esta capa estará ubicado como servidor de base de datos PostgreSQL y un conjunto de Ficheros de Configuración de la arquitectura tecnológica.
5. Capa de Servicio: En esta última capa se encuentran todos los subsistemas que prestan y consumen servicios entre sí.

2.2.2. Estudio de la estructura de carpetas

El subsistema Xedro-ERP está basado en el marco de trabajo Sauxe, en el cual los componentes utilizan la misma estructura de carpetas definida en la capa de datos. En la **Figura 5** se muestra el componente Nomenclador de Cuentas con su estructura de carpetas utilizando la versión 1.2.2 de Doctrine.

Figura5. Estructura de carpetas del componente Nomenclador de Cuentas en Doctrine 1.2.2.

2.3. Etapa de Integración

El propósito de la etapa de integración es lograr la correcta comunicación de Doctrine 2 con el marco de trabajo Sauxe y como consecuencia se hace necesaria la creación de directorios donde se añaden las clases y funcionalidades para la integración. Además se adicionan los parámetros de configuración y las nuevas librerías.

[15]PDO: del inglés PHP Data Objects, es una extensión que provee una capa de abstracción de acceso a datos para PHP5.

2.3.1. Integración de Doctrine 2.2.0 con el marco de trabajo Sauxe

El proceso de integración posee gran importancia, debido a que es la actividad que crea las bases para realizar la implementación de la migración. Se parte del estudio y selección de las nuevas librerías que se utilizan en el acceso a datos por parte de Doctrine 2.2.0. Además se crean los directorios donde se ubicarán las clases que se modifican en la integración, conteniendo éstas sus funcionalidades actualizadas y adaptadas a los nuevos requerimientos del ORM.

2.3.2. Estudio de las librerías seleccionadas

Las librerías escogidas son las propuestas por Doctrine 2.2.0 (Common, DBAL y ORM); sus estudios resultan imprescindibles debido a la necesidad de conocer cuándo, cómo y para qué se utilizan cada una de estas. En la **Figura 6** se puede apreciar su ubicación en el marco de trabajo Sauxe.

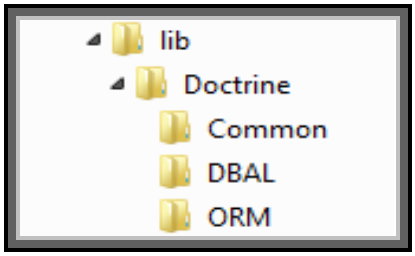


Figura6. Ubicación de las nuevas librerías de Doctrine.

Common: contiene los componentes altamente reutilizables que no tienen dependencias más allá del propio paquete y PHP.

DBAL: es una capa de abstracción de base de datos mejorada basada en PDO, a partir de esta capa la interacción con cualquier base de datos es totalmente transparente.

ORM: contiene las herramientas necesarias que proporcionan una persistencia relacional transparente de objetos PHP sencillos.

2.3.3. Estudio y actualización de las clases seleccionadas

Para lograr la integración de Doctrine 2.2.0 con el marco de trabajo Sauxe se hace necesaria la creación del fichero **Doctrine.php**. Para la ubicación de dicho fichero se crea en **lib/ZendExt/App/** la carpeta **Resource** y dentro de esta se implementa la clase **ZendExt_App_Resource_Doctrine**, en la **Figura 7** se observa lo descrito anteriormente.

Figura7. Ubicación de la clase Doctrine.php.

Para que la nueva clase **Doctrine.php** sea reutilizable en todas las aplicaciones dentro del marco de trabajo Sauxe se procede a crearla como un recurso del marco de trabajo Zend, por lo que la misma debe extender de **Zend_Application_Resource_ResourceAbstract.** La función principal de este recurso será la configuración de los parámetros necesarios para crear una instancia del EntityManager[16] logrando el acceso a las funcionalidades del marco de persistencia de datos de Doctrine. En la **Figura 8** se evidencia la implementación de la clase creada con sus parámetros de configuración.

```
class ZendExt_App_Resource_Doctrine
extends Zend_Application_Resource_ResourceAbstract {

    public function init() {

        $options = Zend_Registry::get('configdoct');
```

[16] EntityManager: punto de acceso a las funcionalidades del marco de persistencia Doctrine.

```
        $classLoader = new \Doctrine\Common\ClassLoader('Doctrine');
        $classLoader->register();

        $classLoader = new \Doctrine\Common\ClassLoader(
                        $options['entitiesPathNamespace'],
                        $options['entitiesPath']
        );
        $classLoader->register();

        $classLoader = new \Doctrine\Common\ClassLoader(
                        $options['proxiesPathNamespace'],
                        $options['proxiesPath']
        );
        $classLoader->register();
        $classLoader = new \Doctrine\Common\ClassLoader(
                        $options['repositoriesPathNamespace'],
                        $options['repositoriesPath']
        );
        $classLoader->register();
    $cacheClass = 'Doctrine\Common\Cache\ArrayCache';

    $cache = new $cacheClass();
    $config = new \Doctrine\ORM\Configuration();
    $config->setMetadataCacheImpl($cache);
    $driverImpl = $config->newDefaultAnnotationDriver($options['module']);
    $config->setMetadataDriverImpl($driverImpl);
    $config->setQueryCacheImpl($cache);

    $config->setProxyDir($options['proxiesPath']);
    $config->setProxyNamespace($options['proxiesPathNamespace']);
    $config->setAutoGenerateProxyClasses($options['autoGenerateProxyClasses']);

    $em = \Doctrine\ORM\EntityManager::create(
                    $options['conection'], $config
    );

    Zend_Registry::set('em', $em);
    return $em;

}
```

Figura8. Configuración de la clase Doctrine.php.

Otro de los elementos a modificar son los parámetros de configuración del fichero **app/config.php** para lograr un correcto funcionamiento del ORM Doctrine 2.2.0. La función principal de este fichero es garantizar la localización de los directorios de las entidades y los namespaces[17]. En la **Figura 9** se puede observar la correcta configuración de los nuevos parámetros a utilizar por Doctrine.

[17] Namespaces: espacios de nombres utilizados por Doctrine 2.2.0 para el manejo de los ficheros del mapeo.

```
$config['configdoct']['driver'] = "pdo_pgsql";
$config['configdoct']['entitiesPath'] = $_SERVER['DOCUMENT_ROOT'] . "/../apps/";
$config['configdoct']['entitiesPathNamespace'] = "";
$config['configdoct']['proxiesPath'] = $_SERVER['DOCUMENT_ROOT'] . "/../apps/";
$config['configdoct']['proxiesPathNamespace'] = "";
$config['configdoct']['repositoriesPath'] = $_SERVER['DOCUMENT_ROOT'] . "/../apps";
$config['configdoct']['repositoriesPathNamespace'] = "";
$config['configdoct']['cacheClass'] = "Doctrine\Common\Cache\ApcCache";
$config['configdoct']['autoGenerateProxyClasses'] = true;
$config['configdoct']['Path'] = $_SERVER['DOCUMENT_ROOT'] . "/../lib/ZendExt/";
```

Figura9. Configuración de los nuevos parámetros de Doctrine.

Una vez creado el recurso y ubicados los parámetros de configuración de Doctrine 2.2.0 será necesario modificar la funcionalidad **OpenConnections()** de la clase **ZendExt_Aspect_TransactionManager.php**, que es la encargada de configurar y manejar las conexiones del marco de persistencia de datos de Doctrine en el marco de trabajo Sauxe. En la **Figura 10** se muestra la ubicación de dicha clase.

Figura10. Ubicación de la clase TransactionManager.

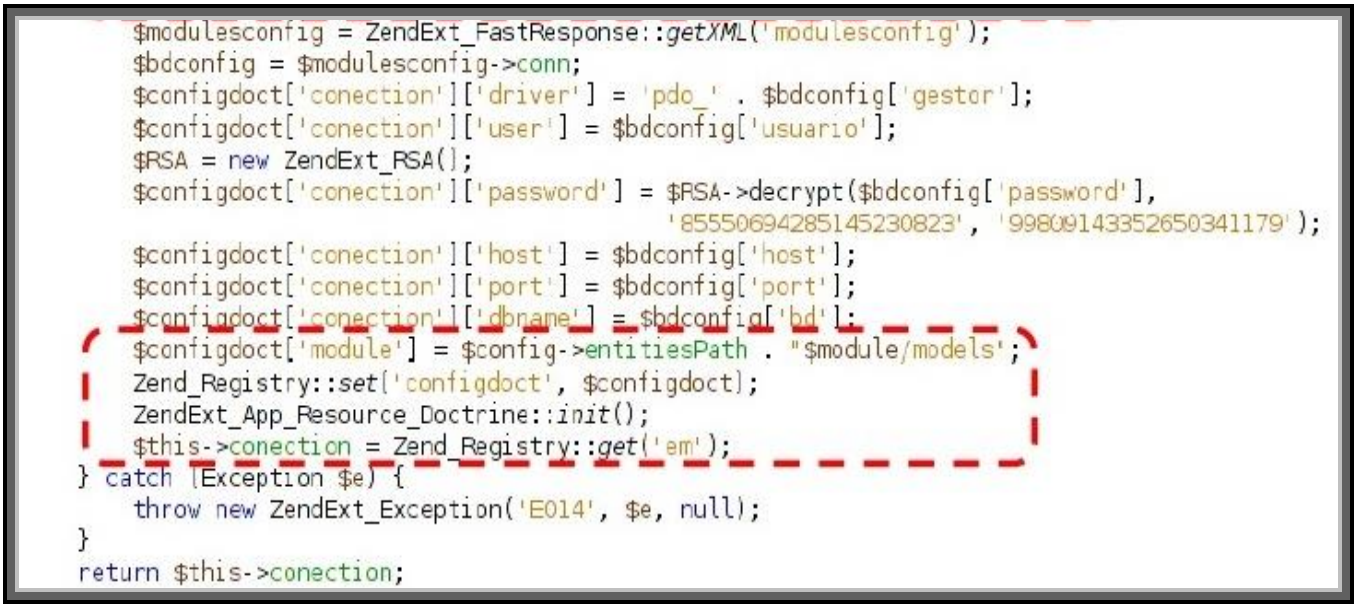
Para lograr una correcta configuración a través de la funcionalidad **openConections()** será necesario hacer uso de los parámetros anteriormente creados en el fichero de configuración **config.php.**

A continuación se listan los nuevos parámetros de configuración que se utilizarán:

1. driver: tipo de controlador que implementan los PDO de interfaz para permitir el acceso de PHP a las bases de datos. Debido a que el marco de trabajo Sauxe utiliza PostgreSQL como gestor de base de datos se debe usar pdo_pgsql.
2. entitiesPath: ruta en la que se encontrarán las entidades del mapeo.
3. entitiesPathNamespace: nombre del namespace que debe estar asociado a la ruta de las entidades del mapeo.
4. proxiesPath: ruta en la que se encontrarán las clases proxy pertenecientes a las entidades del mapeo.
5. proxiesPathNamespace: nombre del namespace de las clases proxy.
6. repositoriesPath: ruta en las que se encontrarán las clases repository pertenecientes a las entidades del mapeo.
7. repositoriesPathNamespace: nombre del namespace de las clases repository.
8. cacheClass: tipo de cache que utilizará el ORM Doctrine, en este caso: *Doctrine\Common\Cache\ApcCache*.
9. autoGenerateProxyClasses: parámetro de tipo boolean para indicar si se generan o no de manera automática las clases proxy.

A continuación se muestra en la **Figura 11** la actualización de la funcionalidad **OpenConections()** una vez adaptada a la nueva estructura de Doctrine.

```
public function openConections($module = null, $current = false) {
    try {
        $config = Zend_Registry::getInstance()->config->configdoct;
        $config->entitiesPathNamespace = $module . "\models";
        $config->proxiesPath = $config->proxiesPath . $module . '/models/Proxies";
        $config->proxiesPathNamespace = $module . "\models\Proxies";
        $config->repositoriesPathNamespace = $module . "\models\Repositories";
        if ($module == "Trace") {
            $config->proxiesPath = $config->Path . $module . "/models/Proxies";
            $config->entitiesPath = $config->Path;
            $config->repositoriesPath = $config->Path;
        }
        $configdoct = $config->toArray();
```



```
        $modulesconfig = ZendExt_FastResponse::getXML('modulesconfig');
        $bdconfig = $modulesconfig->conn;
        $configdoct['conection']['driver'] = 'pdo_' . $bdconfig['gestor'];
        $configdoct['conection']['user'] = $bdconfig['usuario'];
        $RSA = new ZendExt_RSA();
        $configdoct['conection']['password'] = $RSA->decrypt($bdconfig['password'],
                                               '8555069428514523O823', '9980914335265O341179');
        $configdoct['conection']['host'] = $bdconfig['host'];
        $configdoct['conection']['port'] = $bdconfig['port'];
        $configdoct['conection']['dbname'] = $bdconfig['bd'];
        $configdoct['module'] = $config->entitiesPath . "$module/models";
        Zend_Registry::set('configdoct', $configdoct);
        ZendExt_App_Resource_Doctrine::init();
        $this->conection = Zend_Registry::get('em');
    } catch (Exception $e) {
        throw new ZendExt_Exception('E014', $e, null);
    }
    return $this->conection;
```

Figura11. Método OpenConections( ) de la clase TransactionManager.

Una vez creados y configurados los parámetros necesarios se inicializa Doctrine para poder acceder a las funcionalidades que este brinda, para ello se hace una llamada al recurso anteriormente configurado **ZendExt_App_Resource_Doctrine::init( )**.

2.3.3. Pruebas de Integración

Para verificar si es correcta la integración que se desarrolla en esta etapa de la migración es de suma importancia aplicarle pruebas al marco de trabajo Sauxe con la nueva configuración. Entre ellas se encuentra la prueba de conectividad a la base de datos donde se verifica la correcta conexión a la base de datos introduciéndole los parámetros de forma correcta e incorrecta. Otra prueba que se aplica es la configuración de los namespaces donde se acredita que las mismas respondan de forma precisa mediante las llamada a entidades con namespaces existentes e inexistentes.

2.4. Etapa de Implementación

En la fase de Implementación se aplican nuevos cambios en el diseño de la arquitectura base del sistema propuestos por Doctrine 2.2.0, se parte de la modificación de la estructura de carpetas y archivos para la capa de acceso a datos en el componente Nomenclador de Cuentas. Otra de las actividades de esta fase se refiere al rediseño y estructuración de los métodos y consultas que responden a las relaciones y funcionalidades del componente.

2.4.1. Redefinir la Capa de Acceso a Datos

La capa de acceso a datos presenta transformaciones debido a las diferencias entre las versiones 1.2.2 y 2.2.0 de Doctrine. Entre los cambios que se realizan se encuentra la nueva estructura de carpetas donde se re implementan los métodos y las consultas, además es necesario llevar a cabo el mapeo de las tablas con el fin de integrarlas a la nueva estructura de Doctrine 2.2.0 junto a la migración manual de las clases ya mapeadas con Doctrine 1.2.2.

2.4.2. Estructura de carpetas del componente Nomenclador de Cuentas

Para las clases del modelo la nueva estructura de carpetas que propone Doctrine 2.2.0 se generan 3 carpetas (Entity, Proxy y Repository), con el fin de organizar los ficheros generados por su tipo o función.

Figura12. Nueva estructura de carpetas de Doctrine 2.2.0.

1. En la carpeta bussines se encuentran los ficheros encargados de manejar el negocio de la capa de acceso a datos.
2. Los ficheros que representan las tablas mapeadas ubicados en generated son generados con su nueva estructura en la Carpeta Entity. Estas clases permiten la realización del mapeo objeto relacional que consiste en el trabajo con entidades o clases persistentes que representan las tablas con sus campos o columnas
3. Los ficheros generados como resultado del mapeo que contienen las clases Proxy usadas por Doctrine se ubican en la Carpeta Proxy. Estas clases no son más que objetos que se ponen en el lugar del objeto real, y se utilizan para realizar varias funciones, pero principalmente, para la transparencia de carga diferida. Los objetos proxy con sus instalaciones de carga diferida ayudan a mantener el subconjunto de objetos que ya están en la memoria conectada con el resto de los objetos.

4. Las funcionalidades con las consultas que se encontraban en los ficheros localizados en la carpeta domain la deben realizar los ficheros ubicados dentro de la carpeta Repository. Por cada tabla en la base de datos se genera un fichero Repository, su nomenclatura está compuesta por el nombre del fichero de mapeo de la tabla y Repository al final. Estos ficheros son clases que extienden de la clase Doctrine\ORM\EntityRepository.

2.4.3. Transformación del modelo

El proceso de transformación del modelo comienza con el listado de entidades seleccionadas por el desarrollador para la generación del código por cada entidad utilizando las anotaciones Docblock[18] propuesta por Doctrine 2.2.0.

Las anotaciones Docblock son utilizadas con el objetivo de representar la estructura de las entidades de las clases de acceso a datos 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.

La estructura para cada entidad se realiza inicializando la clase en PHP, el namespace y luego las anotaciones necesarias para la configuración (@Table*)* e identificación (@Entity) de la clase como entidad. Luego se obtiene el código referente a cada columna que posea la entidad con la anotación correspondiente. Finalizada la confección del código de las columnas se crean las relaciones (@OneToOne, @OneToMany y @ManyToOne, @ManyToMany) existentes entre cada entidad y en caso de ser seleccionada la opción se generan los métodos Get y Set por columna.

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. El generar entidades consiste en un grupo de preguntas que se realizarán para conformar las entidades en

[18]Docblock: es un tipo especial de comentario en el código que puede proporcionar información detallada acerca de un elemento.

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 una instrucción de comando indicada por consola para obtener cada entidad.

La **Figura13** muestras las anotaciones Docblock utilizada por Doctrine para representar los metadatos de las clases a generar.

Anotación	Descripción	Atributos	Descripción
@Table	Anotación que configura las características de la tabla.		
@Entity	Anotación requerida para marcar a una clase PHP como entidad.	RepositoryClass	Especifica la dirección 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.		
@GeneratdValue	Anotación que especifica la estrategia utilizada para la generación del identificador de una variable que es identificada con la anotación @Id.	strategy	Establece el tipo de estrategia de generación del id, entre los valores válidos se encuentran: AUTO: (Opción predeterminada) Ofrece una portabilidad completa al indicarle a Doctrine la estrategia preferida por la plataforma de BD utilizada. SEQUENCE: Le indica a Doctrine que utilice una secuencia de BD para la generación de id. Compatible solo con Oracle y PostgreSql.

			IDENTITY: Le indica a Doctrine que debe utilizar columnas de identidad especiales en la base de datos que generen un valor al insertar una fila.
@Sequence Generator	Anotación que permite especificar detalles acerca de la secuencia, como el tamaño de incremento y los valores iniciales de la secuencia.	sequenceName	Nombre de la secuencia.
		allocationSize	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 @ManyToMany	Anotaciones que indican las relaciones que son utilizadas entre dos entidades modeladas en el Diagrama Entidad Relación.	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.
		referencedColumn Name	Nombre del identificador de clave principal que se utiliza para la unión de esta relación.

Figura13. Anotaciones Docblock.

2.4.4. Transformación de las Formas de mapeo

Las formas de mapeo entre Doctrine 1.2.2 y Doctrine 2.2.0 se realizan de un modo diferente. En Doctrine 1.2.2 se definen las tablas y columnas a partir de funciones predefinidas dentro del marco de persistencia de datos. A diferencia de Doctrine 2.2.0 que define su propio conjunto de anotaciones docblock en el código para mapear la información y suministrar la asignación de metadatos objeto relacional. A continuación se evidencia la estrategia de mapeo que utilizan ambas versiones de Doctrine, especificando sus principales componentes para estructurar las entidades o clase persistentes.

Definir Tablas

La estrategia que utiliza Doctrine 1.2.2 para definir el mapeo de una tabla es generar las entidades que extienden de una clase base llamada Doctrine_Record y una funcionalidad llamada setTableDefinition.

```
abstract class BaseConfApertura extends Doctrine_Record
  public function setTableDefinition()
  {
    $this->setTableName('conf_apertura');
  }
```

Por su parte, en la versión 2.2.0 de Doctrine se mapea la información a partir de anotaciones Docblock en el código, que se muestran en las clases PHP en forma de comentarios. El mapeo de una tabla a través de la anotación @Table y se debe indicar además que esta es una entidad @Entity.

```php
<?php
namespace nomenclador_cuentas\models\Entity;
/**
 * ConfApertura
 *
 * @Table(name="mod_contabilidad.conf_apertura")
 * @Entity(repositoryClass="contabilidad\models\Repository\ConfAperturaRepository")
 */
class ConfApertura{
```

Definir Columnas

A la hora de representar una columna en Doctrine 1.2.2 se hace una llamada a la función hasColumn y se definen cada uno de los parámetros de la misma.

```php
$this->hasColumn('idconfapertura', 'decimal', null,
      array('notnull' => true, 'primary' => true, 'sequence' => 'mod_contabilidad.sec_configapertura'));
```

Doctrine 2.2.0 realiza la definición de columnas a través de las anotaciones Docblock como se muestra a continuación.

```php
/**
 * @var decimal
 *
 * @Column(name="idconfapertura", type="decimal", nullable="false")
 * @Id
 * @GeneratedValue(strategy="SEQUENCE")
 * @SequenceGenerator(sequenceName=" mod_contabilidad.sec_configapertura",
 * allocationSize="", initialValue="1")
 */
private $idconfapertura = null;
```

Formas de Representación de las relaciones entre Tablas en las clases Entity

En la versión 1.2.2 de Doctrine se generan 2 ficheros de Mapeo, el fichero Base que extiende de Doctrine_Record y el fichero que extiende del fichero Base, en este último se definen las relaciones entre tablas se realiza a través de la funcionalidad setUp. En el caso que se muestra a continuación se está especificando que una DatRefApertura tiene una única instancia del objeto mapeado por la entidad NomCuenta. Estas declaraciones pertenecen a las clases mapeadas como DatRefApertura.

```
public function setUp(){
  $this->hasOne('NomCuenta', array('local'=>'idcuenta','foreign'=>'idcuenta'));
  parent::setUp();
                          }
```

En Doctrine 2.2.0 La anotación @ManyToOne indica que se tiene una única instancia del objeto mapeado por la entidad NomCuenta pero un NomCuenta puede ser usado por muchos a la vez.

```
/**
 * @var \NomCuenta $idcuenta
 *
 * @ManyToOne(targetEntity="NomCuenta")
 * @JoinColumns({
 *   @JoinColumn(name="idcuenta", referencedColumnName="idcuenta")
 * })
 */
private $idcuenta;
```

Otro tipo de relación entre tablas que puede existir son las de tipo ManyToMany, para ello Doctrine 1.2.2 lo define su estructura como se muestra a continuación.

```
public function setUp() {
    $this->hasMany('DatRefapertura', array('local' => 'idcuenta', 'foreign' => 'idcuenta'));
    parent::setUp();
}
```

La particularidad del mapeo en Doctrine 2.2.0 está dada debido a que no se genera un fichero por la tabla en la base de datos que guarda las llaves primarias de las tablas a unir, en este caso se hace referencia a la entidad de mapeo con que se posee la relación y mediante la anotación @JoinTable se le indica cuál es la tabla a través de la cual se relacionan. Otro

aspecto a tener en cuenta es que en estos casos se genera el constructor y dentro se inicializa la variable de la relación ManyToMany como un ArrayCollection definido por Doctrine.

```
/**
 * @var \Doctrine\Common\Collections\Collection
 *
 * @ManyToMany(targetEntity="DatRefapertura", inversedBy="idcuenta")
 * @JoinTable(name="mod_contabilidad.dat_refapertura",
 *      joinColumns={@JoinColumn(name="idcuenta", referencedColumnName="idcuenta")},
 *      inverseJoinColumns={@JoinColumn(name="idcuenta", referencedColumnName="idcuenta")}
 *      )
 */
private $refapertura;
```

Funcionalidades en las clases Model

Las clases Model son las encargadas de manejar el negocio de la capa de acceso a datos, Doctrine 2.2.0 define nuevas estructuras y formas de hacer sus funciones. A la hora de hacer una llamada a las funcionalidades implementadas en las clases Repository se debe crear una instancia de la clase **TransactionManager** y hacer una llamada al método **openConnection** para obtener la instancia del **EntityManager,** una vez hecho esto se llama la funcionalidad **getRepository** con el parámetro namespace de la clase que se quiere obtener el repositorio y finalmente se llama la funcionalidad.

```
<?php

class ConfAperturaModel extends ZendExt_Model
{
    public function ConfApertura() {
        parent::ZendExt_Model('nomenclador_cuentas');
                                        }

    public function ModificarNomenclador(Confapertura $obj){
        $mg = ZendExt_Aspect_TransactionManager::getInstance();
        $_em = $mg->getConnection('nomenclador_cuentas');
        $result = $_em->getRepository("nomenclador_cuentas\models\Entity\ConfApertura")
                      ->ModificarNomenclador($obj, $_em);
        return $result;
                                                            }
        }
```

En este caso se hace referencia a la funcionalidad ModificarNomenclador de la clase que tiene como namespace "nomenclador_cuentas\models\Entity\ConfApertura".

Consultas en las clases Repository

Una vez mapeadas las tablas de la base de datos se procede a modificar las funcionalidades y consultas que se estaban ubicadas en los ficheros de la carpeta domain; estas funcionalidades se deben ubicar en los ficheros dentro de la carpeta Repository. En la versión 1.2.2 de Doctrine las funcionalidades quedaban de la manera que se evidencia a continuación:

```
static public function obtenerCuentaPorNom($idBuscado) {
    $query = new Doctrine_Query();
    return ZendExt_ClassStandard::Object($query->select("n.idcuenta, n.clavecuenta, n.ordenizq,
                                    n.ordender, n.fechainicio, n.fechafin, n.descripcion, n.nivel,
                                    n.concatcta, n.concatctax, n.idcuentapadre, n.idcontenidoe,
                                    n.idnaturaleza, n.idestructurae, n.idtipocuenta")
                                   ->from('NomCuenta n')
                                   ->innerJoin('n.DatRefapertura rf')
                                   ->where("rf.codigo = ?", $idBuscado)
                                   ->setHydrationMode(Doctrine :: HYDRATE_ARRAY)
                                   ->execute());
     }
```

En Doctrine 2.2.0 las funcionalidades deben tener como parámetro además de los necesarios para la ejecución de las consultas, una variable que posea una instancia de la entidad EntityManager ($_em).

```
static public function obtenerCuentaPorNom($idBuscado, $_em) {
  $result = $_em
           ->createQuery('select n.idcuenta, n.clavecuenta, n.ordenizq, n.ordender, n.fechainicio,
                          n.fechafin, n.descripcion, n.nivel, n.concatcta, n.concatctax,
                          n.idcuentapadre, n.idcontenidoe, n.idnaturaleza, n.idestructurae,
                          n.idtipocuenta from nomenclador_cuentas\models\Entity\NomCuenta n
                         Join n.DatRefapertura rf
                         where rf.codigo = ?1')
                         ->setParameter(1, $idBuscado)
                         ->getArrayResult();
           return $result;
    }
```

2.5. Conclusiones parciales.

La migración se dividió en tres etapas: Inicial, Integración e Implementación, cumpliéndose las actividades específicas en cada una de ellas para alcanzar los objetivos definidos.

Se realizaron las configuraciones necesarias en los ficheros del subsistema Contabilidad para garantizar la integración y el correcto funcionamiento del marco de trabajo Doctrine en su versión 2.2.0.

Se migraron un total de 202 ficheros de mapeo divididos en las 4 carpetas generadas por Doctrine 2.2.0, esta actividad fue la que más tiempo consumió, no por la complejidad, sino por el monto de ficheros que se realizaron de forma manual.

CAPÍTULO III: VALIDACIÓN DE LA SOLUCIÓN

3.1. Introducción

En el presente capítulo se describe la validación de la migración mediante la utilización de la herramienta JMeter Apache 2.3.1 realizando pruebas de rendimiento sobre el componente Nomenclador de cuentas del subsistema Contabilidad de Xedro-ERP. Entre las pruebas a realizar se encuentra la prueba de carga y/o estrés que permite conocer cuál será el rendimiento de la aplicación luego de recibir una gran cantidad de usuarios conectados simultáneamente.

Las prueba serán aplicadas específicamente al componente Nomenclador de cuentas del subsistema Contabilidad en la versión 1.2.2 y 2.2.0 de Doctrine. Los resultados obtenidos serán comparados para verificar el aumento del rendimiento del componente a partir de la migración a Doctrine 2.2.0 de la capa de acceso a datos.

3.2. Pruebas de rendimiento

Las pruebas de rendimiento se realizan para determinar lo rápido que puede ejecutar un sistema una o varias tareas en condiciones particulares de trabajo. También sirven para validar y verificar otros atributos de la calidad del sistema, tales como la escalabilidad, fiabilidad y uso de los recursos. Las pruebas de rendimiento son un subconjunto de la ingeniería de pruebas, una práctica informática que se esfuerza por mejorar el rendimiento, englobándose en el diseño y la arquitectura de un sistema, antes incluso del esfuerzo inicial de la codificación [38].

3.2.1. Pruebas de carga

Una prueba de carga se realiza generalmente para observar el comportamiento de una aplicación bajo una cantidad de peticiones esperadas. Esta carga puede ser el número esperado de usuarios concurrentes utilizando la aplicación y que realizan un número específico de transacciones durante el tiempo que dura la carga.

3.2.2. Pruebas de estrés

Las pruebas de estrés son utilizadas para someter a la aplicación al límite de su funcionamiento, mediante la ejecución de un número de usuarios muy superior al esperado. Esta prueba tiene como finalidad determinar la robustez de una aplicación cuando la carga es

extrema y ayuda a los administradores a determinar si la aplicación se comportará correctamente en dichas situaciones.

3.3. Características de la herramienta JMeter Apache 2.3.1

JMeter presenta una estructura en árbol que brinda mayor cantidad de variantes para recoger los resultados obtenidos y de esta forma permite hacer un análisis exhaustivo de las pruebas realizadas.

La licencia para usar JMeter tiene un costo de adquisición de cero pesos, la curva de aprendizaje de esta herramienta es muy corta y es fácil encontrar manuales completos, foros y gran contenido en línea [29].

La utilización del JMeter supone un ahorro considerable de tiempo para la realización de pruebas a un sistema. La mayor inversión de tiempo es utilizada en la fase de estudio de los casos de uso críticos en la aplicación web y la elaboración del plan de pruebas en la herramienta.

Una de las ventajas más significante es que permite almacenar los resultados de las pruebas y generar gráficos que representan los aspectos a probar. Otro aspecto a favor de esta herramienta es su desarrollo en el marco del software libre, política que apoya la universidad y que por sus características existe un por ciento superior de confianza en la utilización de la misma ya que se puede contar con todo su código [31].

3.4. Especificación de las pruebas

Para realizar las pruebas se definió que se realizarán en entornos de 100, 250 y 500 hilos simulando los usuarios conectados simultáneamente al sistema. Los resultados de las pruebas se recopilarán en el Informe Agregado generado por la herramienta JMeter.

Informe Agregado: Se crea una fila por cada petición con los resultados de la prueba realizada. Por cada una de estas filas se muestra la siguiente información:

- **Label:** El nombre de la muestra (conjunto de muestras).
- **# Muestras:** El número de muestras para cada URL.
- **Media:** El tiempo medio transcurrido para un conjunto de resultados.
- **Mín:** El mínimo tiempo transcurrido para las muestras de la URL dada.

- **Máx:** El máximo tiempo transcurrido para las muestras de la URL dada.
- **Error %:** Porcentaje de las peticiones con errores.
- **Rendimiento:** Rendimiento medido en base a peticiones por segundo/minuto/hora.
- **Kilobytes/sec:** Rendimiento medido en Kilobytes por segundo.

Las pruebas de carga se medirán a partir de los resultados obtenidos en el Informe Agregado, específicamente los valores que se encuentren en la columna del rendimiento medido en Kilobytes por segundo.

Para determinar el tiempo total en que los usuarios estuvieron accediendo al sistema se utilizó la siguiente fórmula: *Tiempo Total = #Muestras * Media*. El resultado obtenido se expresa en milisegundos.

Para las pruebas de estrés se simulará una conexión de 100, 250 y 500 usuarios realizando simultáneamente peticiones a la aplicación.

Para determinar la estabilidad de la aplicación se simularán 100, 250 y 500 usuarios accediendo concurrentemente al sistema y que el mismo responda correctamente, se comprobará en la columna Error % del Informe Agregado.

3.5. Análisis e Interpretación de los resultados

Medición a la funcionalidad Gestionar nomenclador por apertura simulando una muestra de 100 usuarios:

Doctrine 1.2.2

# Muestras	Media	Mediana	Linea de 90%	Mín	Máx	% Error	Rendimiento	Kb/sec
100	21171	15234	42228	170	102594	0,00%	57,9/min	,0
400	9876	5067	23518	256	85743	0,00%	2,6/sec	4,5
100	11541	7114	32587	449	70599	0,00%	42,4/min	,5
100	11720	7008	30435	502	59149	0,00%	42,5/min	,0
100	11222	6398	32536	374	68911	0,00%	42,4/min	,2
100	9100	6399	23189	349	40655	0,00%	43,0/min	,1
100	9585	6669	23996	323	37185	0,00%	47,5/min	,0
1000	11384	6607	29473	170	102594	0,00%	6,5/sec	5,2

Doctrine 2.2.0

# Muestras	Media	Mediana	Linea de 90%	Mín	Máx	% Error	Rendimiento	Kb/sec
100	5521	6276	9072	183	10012	0,00%	9,0/sec	161,0
100	1995	1971	3617	121	6452	0,00%	7,2/sec	131,5
100	2185	2357	3516	202	7309	0,00%	5,9/sec	108,0
100	2041	2213	3351	127	4030	0,00%	5,1/sec	92,7
300	1767	1832	3044	74	7644	0,00%	10,8/sec	196,8
100	2008	2226	3236	179	3860	0,00%	4,2/sec	77,2
100	2068	2072	3375	174	4047	0,00%	3,9/sec	71,6
100	1594	1636	2720	216	3424	0,00%	3,7/sec	67,9
1000	2271	2060	3723	74	10012	0,00%	35,1/sec	636,2

El rendimiento en Doctrine 2.2.0 muestra que para una simulación de 100 usuarios junto a un período de subida de un segundo el servidor es capaz de aceptar una media de 35.1 peticiones por segundo con una velocidad de 636.2KB/sec mejorando el rendimiento obtenido con Doctrine 1.2.2 que mostró una media de 6.2 peticiones por segundos con una velocidad de 5.2Kb/sec.

Medición a la funcionalidad Gestionar nomenclador por apertura simulando una muestra de 250 usuarios:

Doctrine 1.2.2

# Muestras	Media	Mediana	Linea de 90%	Mín	Máx	% Error	Rendimiento	Kb/sec
250	94293	55902	245192	152	481591	0,00%	31,1/min	,0
1000	83145	61551	188614	140	419046	0,00%	45,9/min	1,3
250	160476	148613	304165	483	592110	0,00%	18,0/min	,2
250	79522	44597	192577	377	609316	0,00%	15,5/min	,0
250	148697	135901	292842	400	613812	0,00%	13,0/min	,1
250	123935	107330	252258	473	552626	0,00%	12,4/min	,0
250	103381	81797	200983	343	459222	0,00%	13,2/min	,0
2500	104288	82640	231417	140	613812	0,00%	1,9/sec	1,5

CONCLUSIONES

Doctrine 2.2.0

# Muestras	Media	Mediana	Linea de 90%	Mín	Máx	% Error	Rendimiento	Kb/sec
250	21155	11499	48782	183	51189	0,00%	4,7/sec	83,9
250	3405	3718	4962	224	7669	0,00%	4,4/sec	80,7
250	3399	3598	4939	104	10068	0,00%	4,2/sec	76,8
250	3421	3569	4931	263	13341	0,00%	4,1/sec	73,8
750	3250	3439	5009	119	6016	0,00%	10,8/sec	195,3
250	3302	3469	4936	236	5776	0,00%	3,8/sec	69,4
250	3443	3408	4918	347	48650	0,00%	3,7/sec	68,1
250	3204	3425	5055	146	6182	0,00%	3,7/sec	67,5
2500	5108	3679	5589	104	51189	0,00%	35,2/sec	638,9

El rendimiento en Doctrine 2.2.0 muestra que para una simulación de 250 usuarios junto a un período de subida de un segundo el servidor es capaz de aceptar una media de 35.2 peticiones por segundo con una velocidad de 638.9KB/sec mejorando el rendimiento obtenido con Doctrine 1.2.2 que mostró una media de 1.9 peticiones por segundos con una velocidad de 1.5Kb/sec.

Medición a la funcionalidad Gestionar nomenclador por apertura simulando una muestra de 500 usuarios:

Doctrine 1.2.2

# Muestras	Media	Mediana	Linea de 90%	Mín	Máx	% Error	Rendimiento	Kb/sec
500	51338	45027	127178	400	127339	11,20%	3,8/sec	62,3
500	5813	3876	5083	124	127268	0,20%	3,8/sec	69,0
500	3830	4023	5108	170	44428	0,20%	3,8/sec	68,7
500	3776	3968	5058	575	6603	0,00%	3,8/sec	68,9
1500	3558	3872	5065	67	6460	0,00%	11,0/sec	200,3
500	3761	3929	5219	305	6681	0,00%	3,9/sec	70,0
500	3726	4007	5017	117	6100	0,00%	3,9/sec	71,3
500	3432	3767	4950	72	6195	0,00%	4,2/sec	75,6
5000	8635	4011	5591	67	127339	1,16%	35,1/sec	629,8

Doctrine 2.2.0

# Muestras	Media	Mediana	Linea de 90%	Mín	Máx	% Error	Rendimiento	Kb/sec
298	102422	66242	261317	149	643851	0,00%	27,7/min	,0
1286	277428	224883	618200	142	1701580	0,00%	22,6/min	,6
299	173151	155027	325209	492	1862996	0,00%	7,5/min	,1
311	122204	64729	308054	352	2663226	0,00%	5,7/min	,0
306	205912	161620	414144	394	1108485	0,00%	5,4/min	,0
306	245738	189012	485990	380	1530518	0,00%	5,4/min	,0
307	343915	292513	656959	108	1952675	0,00%	5,4/min	,0
3113	231564	174508	522697	108	2663226	0,00%	54,8/min	,7

El rendimiento en Doctrine 2.2.0 muestra que para una simulación de 500 usuarios junto a un período de subida de un segundo el servidor es capaz de aceptar una media de 35.1 peticiones por segundo con una velocidad de 629.8KB/sec, mejorando el rendimiento obtenido con Doctrine 1.2.2 que mostró una media de 59.8 peticiones por minuto con una velocidad de 7 Kb/sec.

3.5. Análisis general

	Doctrine 1.2.2			**Doctrine 2.2.0**		
Estrés	**100**	**250**	**500**	**100**	**250**	**500**
Carga	5.2	1.5	7	636.2	638.9	629.8
Flujo de datos	6.5	1.9	-	35.1	35.2	35.1
Estabilidad	Sí	Sí	No	Sí	Sí	Sí
Tiempo total	11384000	260720000	720858732	2271000	12770000	43175000

Tabla1. Comparación del rendimiento de Doctrine Gestionar nomenclador por apertura.

En la Tabla1 se muestran los valores obtenidos de las pruebas realizadas a la funcionalidad Gestionar nomenclador por apertura del componente Nomenclador de Cuentas del subsistema Contabilidad de Xedro-ERP.

CONCLUSIONES

A través de las pruebas de carga al sistema se pudo comprobar que el rendimiento medido en Kb/segundos mejoró notablemente con la versión 2.2.0 de Doctrine, lo que significa que luego de la migración la aplicación aceptará un mayor volumen de peticiones por segundo.

El rendimiento en cuanto al flujo de datos en Doctrine 2.2.0 evidencia que para una simulación de 250 usuarios junto a un período de subida de un segundo el servidor es capaz de aceptar una media de 35.2 peticiones mejorando el resultado obtenido con Doctrine 1.2.2 que mostró una media de 1.9 peticiones.

En cuanto a la estabilidad del sistema se pudo analizar que para más de 400 usuarios conectados simultáneamente a la aplicación esta comienza a presentar fallos cuando hacemos uso de Doctrine 1.2.2.

El tiempo total de respuesta obtenido disminuyó en 247950 segundos aproximadamente para escenarios de 250 usuarios, lo que evidenció que la migración a la versión 2.2.0 de Doctrine mejoró el rendimiento de la aplicación en un 70% en comparación con la versión 1.2.2.

Representación gráfica de los resultados

Para una mejor comprensión de estos resultados se representaron los valores más significantes en las gráficas que a continuación se muestran. La Figura 15 representa la comparación del tiempo de ejecución de la funcionalidad Gestionar nomenclador por apertura.

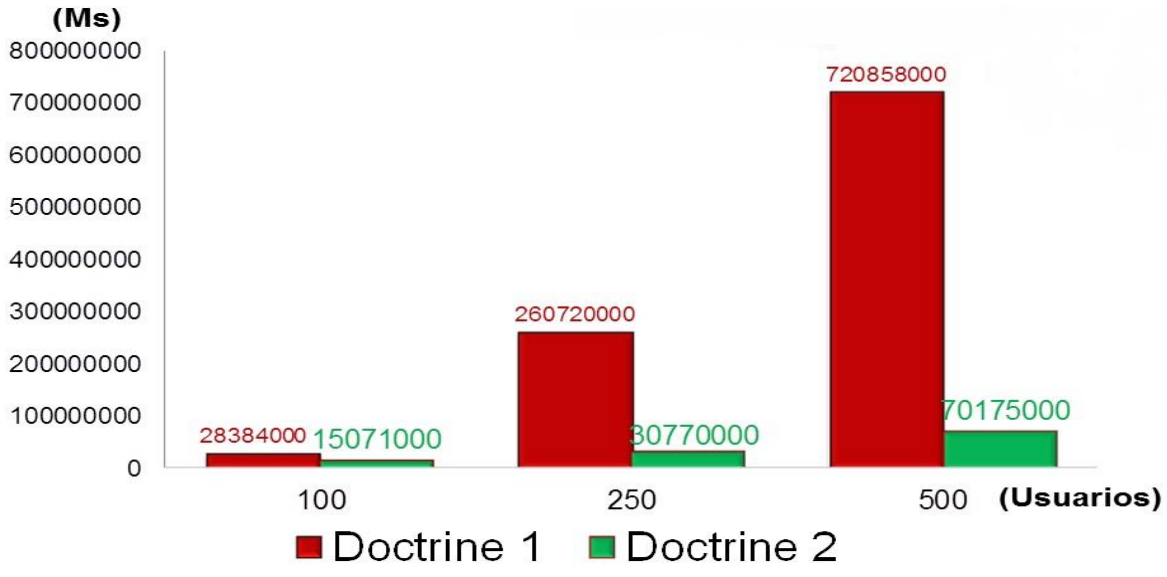


Figura14. Comparación del tiempo de respuesta en milisegundos.

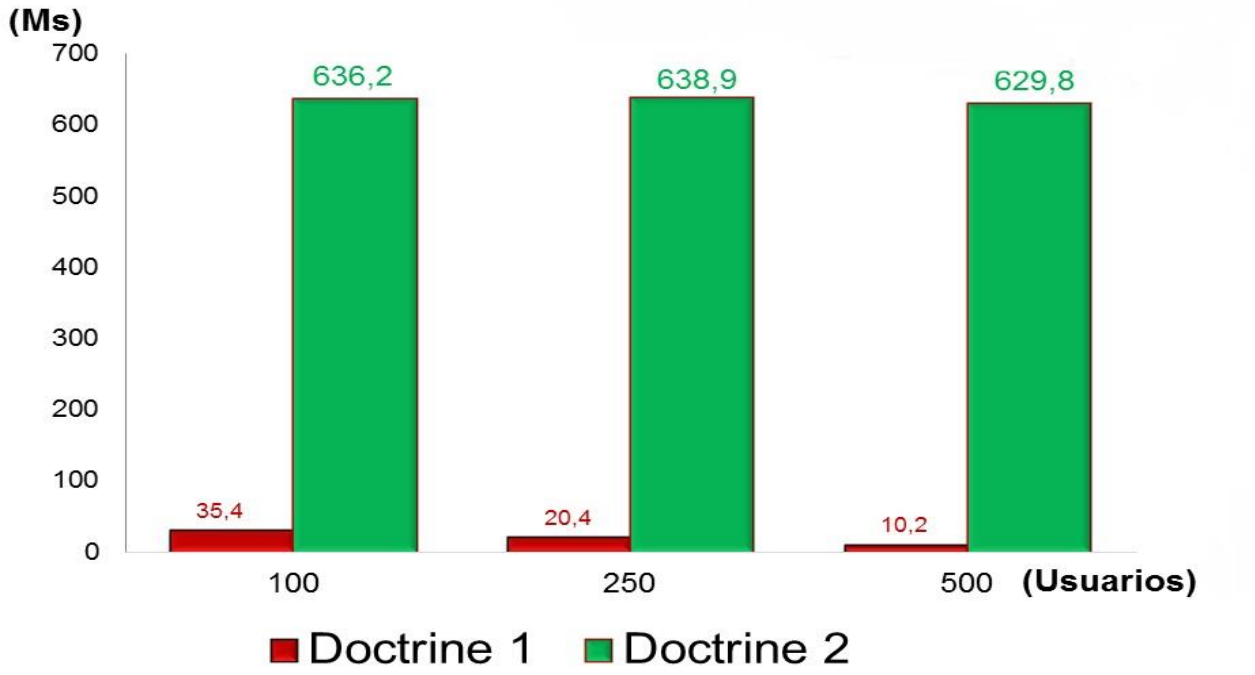


Figura15. Comparación de la carga en Kb/sec.

3.6. Conclusiones parciales

En este capítulo se logró aplicar una serie de pasos para validar la migración realizada, donde el eslabón principal lo constituyó la ejecución de pruebas de rendimiento utilizando como herramienta el JMeter Apache 2.3.1.

Luego del análisis e interpretación de los indicadores de las pruebas, se obtuvo que: como resultado de la migración a Doctrine 2.2.0 de la capa de acceso a datos del componente Nomenclador de cuentas del subsistema Contabilidad de Xedro-ERP el tiempo de respuesta disminuyó un 70% aproximadamente, lo que significa un aumento del rendimiento del mismo.

CONCLUSIONES

Con la realización del presente trabajo de diploma se puede afirmar que se logró alcanzar el objetivo general propuesto, luego de realizar todas las tareas trazadas al inicio de la investigación. El desempeño de los objetivos específicos permitió arribar a las siguientes conclusiones:

- ✓ El estudio realizado sobre el funcionamiento de Doctrine 1.2.2 en Xedro-ERP manifestó la necesidad de llevar a cabo una migración de este marco de trabajo para mejorar el tiempo de respuesta del sistema.
- ✓ El estudio y caracterización de las tecnologías y herramientas definidas para el desarrollo de la migración contribuyeron a su correcta utilización garantizando la disminución del tiempo de desarrollo.
- ✓ Las etapas para la migración contribuyeron a una mayor organización y ágil desarrollo, permitiendo establecer pautas para futuras migraciones de Xedro-ERP.
- ✓ La validación realizada confirmó el mejoramiento del tiempo de respuesta del componente Nomenclador de Cuentas, evidenciando un aumento del rendimiento del subsistema Contabilidad.

RECOMENDACIONES

Para mejorar el funcionamiento de Xedro-ERP se recomienda:

- ✓ Estudiar la factibilidad de realizar la migración del resto de los componentes del subsistema Contabilidad.
- ✓ Valorar la realización de una refactorización de código para mejorar el rendimiento del componente.
- ✓ Mantener una vigilancia tecnológica de nuevas versiones de Doctrine para la realización de futuras migraciones.

ANEXOS

BIBLIOGRAFÍA

1. Baryolo, O.G., *SOLUCIÓN INFORMÁTICA DE AUTORIZACIÓN EN ENTORNOS MULTIENTIDAD Y MULTISISTEMA*. 2001, Universidad de las Ciencias Informáticas: La Habana.
2. Arquitectura, D.d., *Pruebas de rendimiento sobre Cedrux*. 2012: Universidad de las Ciencias Informáticas.
3. Introducción a la evaluación de rendimiento. Departamento de Informática ETSII. Universidad de Valladolid. Disponible en: http://www.infor.uva.es/~miguelv/eesi/mat/01.2-Introduc.pdf.
4. McCafferty, B. Best Practices with ASP.NET, 1.2nd Ed. 2008 [cited 2008 11 Jun 2008]; Available from: http://www.codeproject.com/Articles/13390/NHibernate-Best-Practices-with-ASP-NET-1-2nd-Ed#SUMMARY.
5. Cadavid, A.N. *Sistema de investigación de la Universidad Icesi*. 2008 [cited 2013 25 de Noviembre de 2013]; Available from: http://wwwisis.ufg.edu.sv/wwwisis/documentos/TE/005.74-G934s/005.74-G934s-Capitulo%20II.pdf.
6. Manel Pérez, 2009. ¿Qué es Doctrine ORM? Disponible en: http://manelperez.com/programacion/que-es-doctrine-orm/.
7. Doctrine. Doctrine. Welcome to the Doctrine Project. Disponible en: http://www.doctrine-project.org/
8. Documentation, D.O. *Doctrine 2 ORM Documentation*. 2012 04 de Diciembre de 2012 [cited 2012 04 de Diciembre de 2013]; Documentación del ORM Doctrine 2]. Disponible en: http://www.doctrine-project.org/blog/doctrine-performance-revisited.html.
9. Ing.Yoandry Morejón Borbón,Ing.Darien García Tejo Ing.Oiner Gómez Baryolo, "ARQUITECTURA TECNOLÓGICA PARA EL DESARROLLO DE SOFTWARE," Cuidad de la Habana, 2008.
10. Inc, Z.T., Manual Zend Framework Español.

11. México, U.N.A.d. Universidad Nacional Autónoma de México. 2012 28 de enero 2013 [cited 2012 04-12-2012]; Administración para la toma de decisiones]. Disponible en: http://fcasua.contad.unam.mx/apuntes/interiores/docs/2005/administracion/5/1553.pdf.
12. Ben Collins-Sussman, B.W.F.y.C.M.P., Control de Versiones con Subversion. 2012.
13. RapidSVN. RapidSVN. 2012 04 de Diciembre de 2012; Available from: http://www.rapidsvn.org/.
14. Martínez, R., Portal de Postgres en Español. 2012.
15. Foundation, T.A.S. The Apache Software Foundation. 2012 [cited 2012 20 de Diciembre de 2012]. Disponible en: http://www.apache.org.
16. NetBeans, NetBeans quick start. 2012.
17. JMeter, A. Apache JMeter . 2013 [cited 2013 12 de Abril de 2013]. Disponible en: http://jmeter.apache.org/.
18. Pupo, Y.C. (2010) Libro de Ayuda del Marco de Trabajo Sauxe. Libro de Ayuda del Marco de Trabajo Sauxe.
19. Valdés, Damián Pérez. Maestros del Web. [En línea] 2007. [Citado el: 5 de Diciembre de 2013.] http://www.maestrosdelweb.com/editorial/%C2%BFque-es-javascript/.
20. Garcia, Yunesky del Rio. Implementación del Módulo Índice de Peligrosidad Pre-Delictiva del Proyecto Sistema de Gestión Fiscal. Cuidad de la Habana : s.n., 2010.
21. openformats.org. openformats.org. [En línea] 27 de Agosto de 2010. [Citado el: 10 de Noviembre de 2013.] http://www.openformats.org/es9.
22. Lapuente, María Jesús Lamarca. Hipertexto: El nuevo concepto de documento en la cultura de la imagen. [En línea] 2007. [Citado el: 08 de Diciembre de 2013.] http://www.hipertexto.info/documentos/html.htm
23. González, Carlos D. Curso PostgreSQL, SQL avanzado y PHP. [En línea]Noviembre de 2013. http://www.usabilidadweb.com.ar/postgre.php.
24. Aranzadi.es. Aranzadi.es. [En línea] [Citado el: 24 de Enero de 2012.] http://www.aranzadi.es/index.php/informacion-juridica/informacion-interes/glosario-de-terminos-sobre-internet-y-spam.

25. Ciberaula. Ciberaula. [En línea] 2010. [Citado el: 25 de Enero de 2012.] http://linux.ciberaula.com/articulo/linux_apache_intro/.

26. Cerda, Felipe. NetBeans 6.5 . [En línea] [Citado el: 9 de Febrero de 2012.] http://api.ning.com/files/PcRFEwSyAz6w2k4r-KsPqXzHa3EJ6JtOUDOCxjxcpwJJFWH8XaIgnfUFGtj2TURFJc3LgS7W9tCq-NsM68o3ZBHIMDWV12-W/netbeans65es_cl.pdf.

27. Eclipse.org. 2013 05 de Diciembre de 2013; Available from: http://www.eclipse.org/

28. Descargarte.net. software java. [En línea] 25 de 7 de 2012. [Citado el: 26 de noviembre de 2013.] http://www.descargarte.net/tag/software-java.

29. Palacios, Rodolfo Vázquez. Scribd. [En línea] 2 de Septiembre de 2010. [Citado el: 17 de Febrero de 2012.] http://es.scribd.com/doc/37957736/Mozilla-Firefox-Original.

30. Murray Turoff, L.H.A. *The Delphi Method. Techniques and Applications*. 2013 2002 [cited 2013 25 de Febrero de 2013]; Available from: http://is.njit.edu/pubs/delphibook/delphibook.pdf.

31. Eneko, A. *METODO DELPHI - Prospectiva*. 2001 [cited 2013 27 de Noviembre de 2013]; Available from: http://www.prospectiva.eu/zaharra/03_Delphi_ESTE.pdf.

32. Testing calidad de software. [En línea] [Citado el: 13 de Enero de 2011.]. Disponible en: http://www.ivanfl.com/testingcalidad/index.php/pruebas-de-carga/107-jmeter/99-1-primer-contacto.html

1