



Universidad de las Ciencias Informáticas
Facultad 3

Migración de la capa de acceso a datos del subsistema
Estructura y Composición del Marco de Trabajo Sauxe a
Doctrine 2

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

Autor:

Yoandy Gil Pérez.

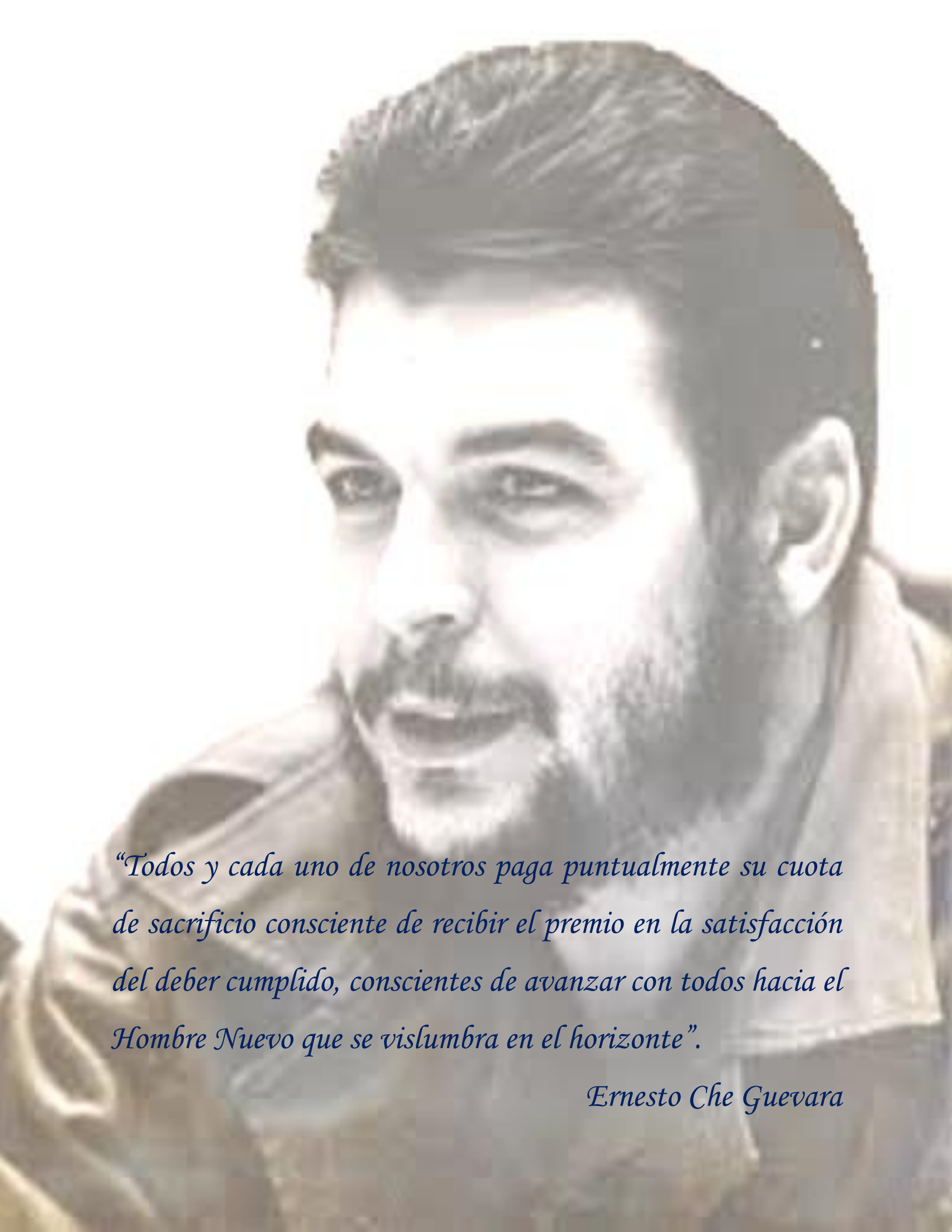
Tutores:

Ing. René R. Bauta Camejo.

Ing. Inoelkis Velazquez Osorio.

Ing. Héctor D. Peguero Alvarez.

La Habana
Junio de 2014



“Todos y cada uno de nosotros paga puntualmente su cuota de sacrificio consciente de recibir el premio en la satisfacción del deber cumplido, conscientes de avanzar con todos hacia el Hombre Nuevo que se vislumbra en el horizonte”.

Ernesto Che Guevara

DECLARACIÓN DE AUTORÍA

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

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

Yoandy Gil Pérez

Firma del Autor

Ing. René R. Bauta Camejo

Firma del Tutor

Ing. Inoelkis Velazquez Osorio

Firma del Tutor

Ing. Héctor D. Peguero Alvarez

Firma del Tutor

DATOS DE CONTACTO

Tutor: Ing. René Bauta Camejo

Edad: 29 años

Institución: Universidad de las Ciencias Informáticas (UCI)

Título: Ingeniero en Ciencias Informáticas

Categoría docente: Instructor

E-mail: rrbauta@uci.cu

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

Tutor: Ing. Inoelkis Velazquez Osorio

Edad: 24 años

Institución: Universidad de las Ciencias Informáticas (UCI)

Título: Ingeniero en Ciencias Informáticas

Categoría docente: Ninguna

E-mail: inoelkis@uci.cu

Ingeniero en Ciencias Informáticas, graduado en 2013 en la Universidad de las Ciencias Informáticas. Tres años de experiencia en el desarrollo de software. Recién Graduado en Adiestramiento.

Tutor: Ing. Héctor D. Peguero Alvarez

Edad: 25 años

Institución: Universidad de las Ciencias Informáticas (UCI)

Título: Ingeniero en Ciencias Informáticas

Categoría docente: Ninguna

E-mail: hdavid@uci.cu

Ingeniero en Ciencias Informáticas, graduado en 2013 en la Universidad de las Ciencias Informáticas. Tres años de experiencia en el desarrollo de software. Recién Graduado en Adiestramiento.

Agradecimientos:

...Agradezco a la Universidad de las Ciencias Informáticas por permitir que me formara como profesional en ella.

...A todos los profesores que contribuyeron a mi preparación y formación profesional.

...A todos mis amigos, los de ayer y los de hoy, que supieron ayudarme en los momentos difíciles cuando los necesité y brindarme su mano sin nada a cambio.

...A mis compañeros de la universidad que durante los 5 años de la carrera compartieron conmigo y de una manera u otra contribuyeron a cambiar mi persona.

...A mis tutores por la ayuda recibida y por guiarme todo este curso.

...A toda mi familia y vecinos, en especial a aquellos que siempre estuvieron pendientes de mí.

...A mi tío Nene y mi tía Niurka por tratarme en todo momento como a un hijo y por brindarme tanta atención. A ellos les estoy enormemente agradecido, y me siento orgulloso por saber que puedo contar con ellos.

...A mis dos hermanos por ayudarme tanto y por confiar siempre en mí.

...A mis padres, a quienes les doy agradecimiento especial, por su enorme preocupación, por fomentar valores en mí, por estar al tanto de cada paso de mi vida, por su impagable sacrificio, por su confianza hacia mi persona y por el cariño que me han dado toda la vida.

...A todo aquel que supo en momentos difíciles brindarme su mano, y a los que no también.

...A todos, ¡Gracias!

Dedicatoria:

...A toda mi familia que estuvo al tanto y preocupada por mí durante mi carrera universitaria.

...A mis amigos, aquellos que han estado a mi lado y me han ayudado todos estos años.

...A mis hermanos Eduardo y Daniel por confiar en mí todo este tiempo y estar al tanto de mis pasos.

...A mi tío Nene por hacer que me sintiera como un hijo para él y dedicarme siempre atención desmedida sin importar nada.

...A mi mamá Lourdes, la persona más buena de este mundo, para ella es esta tesis y todo lo que haga en lo adelante.

...A mi papá Eduardo, mi ejemplo y para hacerle sentir el mismo orgullo que yo siento por él.

...A todos en general, dedico esta tesis con el más grande afecto.

Resumen

Debido al auge que ha tomado en los últimos años el desarrollo de software, se hace imprescindible el uso de nuevas herramientas que faciliten el trabajo de los desarrolladores. Un ejemplo de la afirmación anterior, es la utilización de marcos de trabajo que son un conjunto estandarizado de conceptos, criterios y buenas prácticas para enfocar un tipo de problema que sirva como referencia para resolver nuevos problemas similares.

En el Departamento de Tecnologías del Centro de Informatización de Entidades, es desarrollado el marco de trabajo Sauxe, el cual contiene un conjunto de componentes reutilizables dentro de los que se encuentra el subsistema Estructura y Composición. Este subsistema está compuesto por la integración de varios estilos arquitectónicos de los que destacan el estilo en capas y el Modelo-Vista-Controlador (MVC). Algunas de las capas definidas son: Capa de Presentación, Capa de Control o de Negocio y Capa de Acceso a Datos (CAD). La CAD actualmente está compuesta por el Mapeador de Objeto-Relacional (ORM) *Doctrine* en su versión 1.2.2, pero su uso presenta algunos problemas como la carencia de soporte, deficiencias en el manejo de los datos y problemas de rendimiento.

Por ello surge la necesidad de migrar la CAD del subsistema Estructura y Composición para alcanzar un mejor rendimiento y solucionar los problemas existentes en la primera versión del ORM *Doctrine*. En el presente documento se muestran los resultados del trabajo investigativo realizado para llevar a cabo la migración de la CAD del subsistema Estructura y Composición.

PALABRAS CLAVE:

Capa de Acceso a Datos (CAD), *Doctrine*, Marco de Trabajo, Rendimiento, Subsistema Estructura y Composición.

ÍNDICE

INTRODUCCIÓN	1
CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA	5
1.1. Introducción.....	5
1.2. Principales conceptos relacionados con los ORM	5
1.3. Mapeadores de Objetos Relacionales	6
1.3.1. <i>Hibernate</i>	6
1.3.2. <i>Propel</i>	8
1.3.3. <i>Django</i>	9
1.3.4. <i>SubSonic</i>	10
1.3.5. <i>Doctrine 1.2.2</i>	11
1.3.6. <i>Doctrine 2.0.6</i>	11
1.4. Tecnologías y herramientas para el desarrollo	13
1.4.1. <i>Subversion v1.6.6</i>	13
1.4.2. <i>RapidSVN v0.12.0</i>	13
1.4.3. <i>PostgreSQL v9.1</i>	13
1.4.4. <i>NetBeans v7.4</i>	14
1.4.5. <i>Apache 2.2</i>	14
1.4.6. <i>Sauze 2.2</i>	15
1.4.7. Marco de Trabajo Zend 1.11	15
1.4.8. PHP 5.3.10	15
1.5. Conclusiones parciales.....	16
CAPÍTULO 2: DESARROLLO DE LA MIGRACIÓN.....	18
2.1. Introducción.....	18
2.2. Contenido de la migración	18
2.3. Etapa Inicial.....	18
2.3.1. Arquitectura del MT Sauze.....	18
2.3.2. Estructura de carpetas del componente Estructura y Composición en la CAD.....	19
2.3.3. Integración de <i>Doctrine 2</i> con el MT Zend.....	20
2.4. Etapa de Implementación	20
2.4.1. Nueva estructura de carpetas del componente Estructura y Composición	20
2.4.2. Equivalencia entre la nueva y anterior estructura de carpetas	21
2.4.3. Redefinir Capa de Acceso a Datos	22
2.4.4. Flujo de datos de la Capa de Acceso a Datos.....	22

2.4.5. Mapeo de las tablas	23
2.4.6. Creación de las funcionalidades en las clases Repository	29
2.4.7. Funcionalidades en las clases <i>Model</i>	31
2.4.8. Redefinir Capa de Negocio	32
2.4.9. Actualizar estado de los elementos de configuración	33
2.5. Etapa de Pruebas.....	33
2.5.1. Pruebas realizadas	33
2.5.2. Errores más comunes	33
2.6. Conclusiones parciales.....	34
CAPÍTULO 3: VALIDACIÓN DE LA MIGRACIÓN	35
3.1. Introducción.....	35
3.2. Pruebas de rendimiento.....	35
3.2.1. Entorno de prueba	36
3.2.2. Realizar pruebas de rendimiento	37
3.3. Resultado de las pruebas de rendimiento.....	45
3.4. Resolver no conformidades	49
3.5. Conclusiones Parciales	50
CONCLUSIONES GENERALES	51
RECOMENDACIONES	52
BIBLIOGRAFÍA	53
ANEXOS	56

ÍNDICE DE FIGURAS

Figura 1. Estructura de carpetas del componente metadatos para <i>Doctrine 1</i>	20
Figura 2. Nueva estructura del componente metadatos para <i>Doctrine 2</i>	21
Figura 3. Comparación de las estructuras de carpetas.....	21
Figura 4. Secuencia desde el Controlador hasta el Repository (Flujo de la CAD).....	23
Figura 5. Mapeando un objeto PHP a una tabla.	24
Figura 6. Cómo gestiona <i>Doctrine</i> las entidades relacionadas.....	26
Figura 7. Elemento Grupo de Hilos.....	37
Figura 8. Selección del Servidor Proxy.	38
Figura 9. Valores para el Servidor Proxy HTTP.	39
Figura 10. Grabación de Navegación de la Web.....	39
Figura 11. Ejemplo de petición HTTP de la grabación.	40

Figura 12. Selección de Aserción de Respuestas.....	41
Figura 13. Datos de la Aserción de Respuesta.....	41
Figura 14. Selección de Informe Agregado.....	42
Figura 15. Selección de Ver Árbol de Resultados.....	43
Figura 16. Confección de un Plan de Pruebas.....	43
Figura 17. Informe Agregado.....	44
Figura 18. Ver Árbol de Resultados.....	45
Figura 19. Gráfico de Resultados del Tiempo Total.....	48
Figura 20. Gráfico de Resultados del Tiempo Promedio.....	49

ÍNDICE DE TABLAS

Tabla I. Comparación de rendimiento en las versiones de Doctrine (23).....	12
Tabla II. Resultados de las pruebas realizadas.....	33
Tabla III. Valores correspondientes a la Prueba 1 para 100 hilos.....	45
Tabla IV. Valores correspondientes a la Prueba 2 para 50 hilos.....	46
Tabla V. Valores correspondientes a la Prueba 3 para 25 hilos.....	46
Tabla VI. Valores correspondientes a la Prueba 4 para 100 hilos.....	46
Tabla VII. Valores correspondientes a la Prueba 5 para 50 hilos.....	47
Tabla VIII. Valores correspondientes a la Prueba 6 para 25 hilos.....	47
Tabla IX. Resultados generales.....	47

INTRODUCCIÓN

La sociedad actual se encuentra en un proceso de constante cambio y el país enfrenta el reto de automatizarla utilizando las nuevas Tecnologías de la Información y las Comunicaciones (TIC). Esta acción ha motivado y acelerado los procesos de cambio debido a que los equipos de trabajo crecen continuamente y los problemas son mayores; lo que conduce a desarrollar sistemas más complejos a causa de las exigencias de los clientes en cuanto a calidad y tiempo de respuesta.

En los últimos años el desarrollo de software ha tomado fuerza provocando el uso de nuevas herramientas que faciliten el trabajo de los desarrolladores. Un ejemplo de la afirmación anterior, es la utilización de Marcos de Trabajo (MT) que apoyen y agilicen el proceso de desarrollo. Los MT son un conjunto de prácticas y criterios que pretenden mejorar la rapidez, productividad y profesionalidad en cuanto a la elaboración de sistemas complejos. Además tienden a promover una estructura estándar para las aplicaciones, así como el uso de buenas prácticas de la programación (1).

Cuba apuesta a que el sustento de su economía esté basado, entre otros aspectos, por el aporte del campo de la informática por conceptos de producción de software. La Universidad de las Ciencias Informáticas (UCI), tiene un papel importante en el cumplimiento de este objetivo por la cantidad de tecnologías con la que cuenta y los logros alcanzados en sus años de existencia.

La UCI cuenta con diversos centros de producción, uno de ellos es el Centro de Informatización de Entidades (CEIGE). En CEIGE, el Departamento de Tecnología desarrolló el MT Sauxe sobre el cual se desarrollan algunas de las soluciones de este centro. Sauxe está sustentado por la integración de *ExtJS*, *Zend Framework* y *Doctrine*, además tiene un conjunto de componentes reutilizables que logran una mayor estandarización, flexibilidad, integración y agilidad en el proceso de desarrollo (2).

El subsistema Estructura y Composición (EyC) está compuesto por la integración de algunos de los estilos arquitectónicos, de ellos se destacan el estilo en capas y el MVC. Una de las capas definidas es la CAD, la cual permite gestionar los datos utilizados en los procesos de negocio y abstraerse de la forma en que estos se persisten o son obtenidos. La CAD maneja gran cantidad de información distribuida por lo que necesita el manejo de objetos y la persistencia de los datos, que es la capacidad que tiene un objeto de perdurar en el tiempo (3).

En la actualidad, una de las soluciones más completas para tratar el tema de la persistencia de los datos, es el uso de un Mapeador de Objetos-Relacional (ORM, *Objects Relational Mapper*) que permite guardar objetos en base de datos relacionales de manera lógica al utilizar la técnica de mapeo objeto-relacional¹. Estos se encargan de crear una capa de abstracción entre la aplicación y la base de datos, separando la lógica del negocio al mapear los objetos de la aplicación a datos que persisten en una base de datos (5).

Actualmente el subsistema EyC, para garantizar la persistencia de los datos utiliza la versión 1.2.2 del ORM *Doctrine*, el cual conforma la CAD del MT garantizando la comunicación con la base de datos. El uso de esta versión provoca que, al ejecutar consultas que necesiten manejar gran cantidad de información o hacer referencia a varias tablas para obtener información, se afecte el rendimiento del MT. Por ejemplo cada vez que se realiza una consulta al sistema gestor de base de datos, la aplicación tendrá que convertir la consulta del lenguaje propio del ORM al lenguaje del proveedor usado, enviarla, leer los registros, limpiarlos y convertirlos a objetos (6).

Pruebas realizadas sobre el Sistema de Planificación de Recursos Empresariales Xedro-ERP, que fue desarrollado empleando el MT Sauxe, demostraron un rendimiento reducido de la aplicación y tiempos de ejecución elevados al utilizar el marco de persistencia de datos de *Doctrine*. A partir de los resultados obtenidos se evidenció que el rendimiento era mayor cuando no se utilizaba esta versión del ORM (7). Estos resultados se pueden apreciar en el **Anexo 1**.

Además 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 le dejó de dar soporte, provocando que no se corrijan vulnerabilidades en el código que puedan atentar contra el rendimiento (8).

A partir de la problemática antes descrita se identificó como **problema a resolver**: ¿Cómo aumentar el rendimiento de la capa de acceso a datos del subsistema Estructura y Composición del marco de trabajo Sauxe?

El problema antes expuesto está enmarcado en el proceso para la persistencia de datos como **objeto de estudio**.

¹ El **mapeo objeto-relacional** es una técnica de programación para convertir datos entre el sistema utilizado en un lenguaje de programación orientado a objetos y la utilización de una base de datos relacional, utilizando un motor de persistencia.

Con el propósito de darle solución al problema formulado se establece como **objetivo general**: Migrar la capa de acceso a datos del subsistema Estructura y Composición para garantizar un aumento de su rendimiento.

El objetivo general se desglosa en los siguientes **objetivos específicos**:

- ▶ Construir el Marco Teórico de la investigación relacionada con los ORM existentes para identificar buenas prácticas y posibles puntos de reutilización.
- ▶ Realizar la migración de la capa de acceso a datos del subsistema Estructura y Composición del marco de trabajo Sauxe a *Doctrine 2* para aumentar su rendimiento.
- ▶ Validar mediante el uso de la herramienta JMeter el aumento del rendimiento de la capa de acceso a datos del subsistema Estructura y Composición del marco de trabajo Sauxe.

Del objeto de estudio analizado, se puede definir como **campo de acción**: la capa de acceso a datos del subsistema Estructura y Composición del marco de trabajo Sauxe.

La investigación parte de la siguiente **idea a defender**: si se migra la capa de acceso a datos del subsistema Estructura y Composición, se aumentará su rendimiento.

Métodos teóricos:

Los métodos teóricos permiten estudiar las características del objeto de investigación que no son observables directamente (9). De ellos, se utilizan en este trabajo los siguientes:

- ▶ **Analítico – Sintético**: Este método permite distinguir los elementos relacionados con los ORM y proceder a revisar ordenadamente cada uno de ellos por separado, procesar toda la información y poder sintetizar, diferenciarlos y llegar a conclusiones.
- ▶ **Inductivo – Deductivo**: La inducción y la deducción son partes del conocimiento dialéctico de la realidad, que al aplicar este método permiten, a partir del estudio de los ORM más utilizados actualmente, arribar e inferir sus características peculiares y beneficios que brindan su uso.

Métodos empíricos:

Los métodos empíricos describen y explican las características del objeto y representan un nivel de la investigación cuyo contenido procede de la experiencia (9). De ellos se utilizan en este trabajo los siguientes:

- ▶ **Observación**: Permite centrar la atención de la situación existente en el MT Sauxe, en el subsistema EyC, y en los inconvenientes de la utilización de la versión 1.2.2 de *Doctrine*.

- **Medición:** Posibilita obtener información acerca del rendimiento de la aplicación del MT Sauxe al realizarse sobre este, diferentes llamadas a funciones entre varios componentes, y hacer referencia a numerosas tablas para obtener datos.

El presente trabajo se encuentra estructurado en **Introducción**, tres capítulos, **Conclusiones** y **Recomendaciones**. A continuación se describe el objetivo de cada uno de los capítulos.

En el **Capítulo 1:** “Fundamentación Teórica”, se realiza un análisis de las principales tendencias mundiales de la utilización de los ORM, para identificar buenas prácticas y puntos de reutilización, y se aborda sobre algunas características de las herramientas que se utilizan en la migración de la CAD del subsistema EyC.

En el **Capítulo 2:** “Desarrollo de la Migración”, se realiza una descripción de la migración de la CAD del subsistema EyC destacando los elementos más importantes y realizando los cambios necesarios para su funcionamiento con la nueva versión del ORM *Doctrine*.

En el **Capítulo 3:** “Validación de la Migración”, se describen los resultados obtenidos en la validación de la propuesta de solución, mediante la aplicación de pruebas de rendimiento con la utilización de la herramienta JMeter.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

1.1. Introducción

En este capítulo se refleja la base teórica que sustenta la investigación, enmarcada en varios conceptos asociados al proceso de persistencia de datos. Se realiza un estudio del estado del arte de los ORM donde se describieron las características y definiciones más importantes. Además se analizan las tecnologías y herramientas informáticas utilizadas en la migración de la CAD del MT Sauxe.

1.2. Principales conceptos relacionados con los ORM

Persistencia: Es la capacidad que tiene un objeto de perdurar o permanecer fuera del proceso que lo creó. Su estado puede ser almacenado en disco y recuperado en un futuro (3).

Un **ORM** es una técnica que persiste de forma transparente, los objetos de la aplicación a las tablas de una base de datos relacional. Se comporta como una base de datos virtual, ocultando la arquitectura de base de datos subyacente del usuario. Los ORM proporcionan una funcionalidad completa para llevar a cabo las operaciones de mantenimiento y consultas orientadas a objetos. Además admiten el mapeo de metadatos y ayudan en la gestión de transacciones de la aplicación (5).

ORM es una técnica que utiliza Programación Orientada a Objeto (POO) y los mapea directamente a la base de datos relacional. Es un concepto genérico que puede ser aplicado a cualquier lenguaje así como cualquier base de datos relacional. En cada lenguaje existen implementaciones propias que se usan independientes de su implementación (11).

Un **marco de trabajo de persistencia** es un marco de trabajo que se sitúa entre la capa de negocio y la capa de acceso a los datos, abstrayéndose uno del otro (12).

Marco de trabajo de persistencia: Conjunto cohesivo de clases que colaboran para prestar servicios a la parte fundamental e invariable de un subsistema lógico. Contiene clases concretas y, especialmente, abstractas que definen las interfaces que conformarán las interacciones en que participan (5).

“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” (13).

En la presente investigación se asume que un marco de trabajo de persistencia 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.

1.3. Mapeadores de Objetos Relacionales

La utilización de un ORM presenta grandes ventajas a los desarrolladores, algunas de estas son:

- ▶ Persistencia transparente: Los objetos no conocen absolutamente nada acerca de la base de datos donde son persistidos.
- ▶ Soporte de polimorfismo: Se pueden cargar jerarquías de objetos de forma polimórfica.
- ▶ Soporte completo de asociaciones: Los ORM soportan el mapeo de todos los tipos de relaciones (unidireccionales y bidireccionales) que pueden existir en un modelo de objetos del dominio.
- ▶ Soporte de *cached*: En el contexto de una transacción permite disminuir la cantidad de veces que se encuesta a la base de datos, guardando en memoria los objetos que son accedidos varias veces.
- ▶ Soporte de múltiples dialectos SQL: Son independientes del tipo de base de datos utilizada, simplemente cambian la configuración correspondiente.
- ▶ La principal ventaja de estos sistemas es que reducen la cantidad de código necesario para lograr lo que se conoce como una persistencia de objetos. Proporcionan grandes ventajas a los proyectos, al permitir tener acceso a los datos de una manera sencilla y rápida (14).

Las herramientas de mapeo objeto-relacional se encargan no solo de manejar e integrar las capacidades de los lenguajes de POO con las base de datos relacionales, sino que además buscan reducir el código necesario para llevar a cabo las operaciones de persistencia y recuperación de objetos. Con el propósito de conocer posibles puntos de reutilización y buenas prácticas, se hace un estudio de los siguientes ORM.

1.3.1. *Hibernate*

Es una herramienta ORM que facilita el mapeo de los atributos de una base de datos relacional con el modelo de datos de la aplicación. Este mapeo se hace mediante archivos XML o

anotaciones en las entidades que permiten establecer estas relaciones. También ofrece la posibilidad de utilizar un lenguaje de consulta de datos **HQL** (del inglés *Hibernate Query Language*) (15).

Hibernate es un conjunto de proyectos relacionados, que permiten a los desarrolladores utilizar estilo POJO², que no son más que modelos de dominio en sus aplicaciones de manera que extienden mucho más allá del mapeo objeto relacional. Es software libre, de código abierto, distribuido bajo los términos de la licencia GNU/LGPL³ (15).

Rendimiento:

Hibernate mejora el rendimiento al proporcionar instalaciones de almacenamiento en caché que ayudan a una recuperación más rápida de los datos de la base de datos (12). Trata con la reflexión de Java y el aumento de clases en tiempo de ejecución utilizando una biblioteca de generación de código Java muy poderosa y de alto rendimiento llamada CGLIB. Este se utiliza para extender clases Java e implementar interfaces Java en tiempo de ejecución (15).

Compatibilidad con el gestor de base de datos PostgreSQL:

Este soporta *Oracle*, *MySQL*, *PostgreSQL*, *Front Base* y *MSSQL*.

Uso del Lenguaje PHP:

Hibernate utiliza como lenguaje de programación Java.

Compatibilidad con el MT Sauxe

La compatibilidad con el MT Sauxe se ve limitada debido a que ambos utilizan lenguajes diferentes.

Comunidad que respalde su soporte:

Hibernate cuenta con una amplia comunidad y con cientos de colaboradores a lo largo de todo el mundo por lo que su soporte está garantizado.

Buenas prácticas:

- ▶ Por lo general, no es aconsejable crear una conexión cada vez que se interactúe con la base de datos. Para evitar esta situación, las aplicaciones Java, suelen utilizar una piscina

² **POJO:** del inglés (*Plain Old Java Object*), utilizado por programadores que implementan con Java para enfatizar el uso de clases simples y que no dependen de un MT en especial.

³ **GNU/LGPL:** Licencia Pública General Reducida de GNU, o más conocida por su nombre en inglés *GNU Lesser General Public License*.

(del inglés pool) de conexiones. Cada subproceso de la aplicación que realiza solicitudes sobre la base de datos, solicita una conexión a la piscina, devolviéndola cuando todas las operaciones SQL han sido ejecutadas.

- ▶ Cuando ocurra una excepción, se debe deshacer la operación y cerrar la sesión.
- ▶ Mapear cada clase en su propio fichero, evitando usar un solo documento para mapear todas las clases.
- ▶ No utilizar con frecuencia la recuperación temprana. Usar proxies, que son objetos que permiten la carga diferida, y colecciones perezosas para la mayoría de asociaciones a clases que probablemente no se encuentren en la caché de segundo nivel.

1.3.2. Propel

Propel es una herramienta de mapeo objeto-relacional de código abierto escrito en PHP y es además una parte integral del MT *Symfony* y su ORM por defecto. Permite el acceso a la base de datos utilizando un conjunto de objetos, proporcionando una API⁴ simple para almacenar y recuperar datos. Este le brinda al desarrollador de aplicaciones web las herramientas para trabajar con bases de datos, de igual manera que se trabaja con otras clases y objetos en PHP. Le da a la base de datos una API bien definida utilizando los estándares PHP5, carga automática e iteradores (16). La función primaria de *Propel* es proveer un mapa entre las clases de PHP y tablas de bases de datos.

Rendimiento:

Lo más importante es que *Propel* utiliza *PDO* (del inglés *PHP Data Objects*) en lugar de *Creole* como *DBAL* (Capa de Abstracción de Base de Datos), ofreciendo una significativa mejora del rendimiento (17).

Compatibilidad con el gestor de base de datos PostgreSQL:

Este soporta *MySQL*, *PostgreSQL*, *SQLite*, *MSSQL* y *Oracle* (16).

Uso del Lenguaje PHP:

Utiliza PHP5 o una versión superior.

Compatibilidad con el MT Sauxe:

Puede adaptarse perfectamente al MT Sauxe debido a que está basado en PHP como lenguaje de programación y soporta el gestor de base de datos PostgreSQL.

⁴ **API:** Interfaz de programación de aplicaciones (IPA) o *API* (del inglés *Application Programming Interface*).

Comunidad que respalde su soporte:

Propel posee una amplia comunidad y con un elevado prestigio mundialmente por sus potencialidades, aunque se trabaja en el aumento de su documentación que hasta el año 2010 era escasa (18).

Buenas prácticas:

- ▶ A cada URL5 se le debe crear la ruta asociada. Y esto es obligatorio si se quitan las reglas de enrutamiento por defecto.
- ▶ En caso de que se elimine un método generado se debe eliminar la plantilla generada asociada a ese método.

1.3.3. Django

Django es un entorno de desarrollo web escrito en *Python*⁶ que fomenta el desarrollo rápido y el diseño limpio y pragmático. *Django* es un MT de código abierto que permite construir aplicaciones web más rápido y con menos código. Está compuesto por un mapeo objeto-relacional para convertir los modelos de datos, un apoyo en configuración de URL, un sistema completo de plantillas, sistemas para realizar procesos de solicitud y administración de base de datos relacional, lo que permite un óptimo desarrollo de aplicaciones (19).

Rendimiento:

Django es muy potente, debido a que permite obtener provecho de las ventajas del lenguaje para el cual está diseñado: *Python*.

Compatibilidad con el gestor de base de datos PostgreSQL:

A pesar de que *Django* es compatible con el gestor de base de datos PostgreSQL se recomienda que para su uso se instale el paquete *psycopy2*; el uso de PostgreSQL en Django logra un delicado equilibrio entre costo, características, velocidad y estabilidad (19).

Uso del Lenguaje PHP:

Django está creado sobre el lenguaje *Python*.

Compatibilidad con el MT Sauxe:

⁵ **URL:** es un localizador uniforme de recursos, denominado URL (sigla en inglés de *Uniform Resource Locator*).

⁶ **Python:** es un lenguaje de programación interpretado cuya filosofía hace hincapié en una sintaxis muy limpia y que favorezca un código legible.

La compatibilidad de *Django* con el MT Sauxe se ve limitada debido a que los dos utilizan lenguajes diferentes.

Comunidad que respalde su soporte:

Django cuenta con una comunidad en crecimiento, se ha extendido a varios idiomas como son: español, inglés y portugués.

Buenas prácticas:

- ▶ Sigue el principio DRY (conocido también como Una vez y sólo una).
- ▶ Utiliza una modificación de la arquitectura MVC, esta forma de trabajar permite que sea pragmático.

1.3.4. SubSonic

Subsonic es un marco de persistencia que trabaja en la CAD usando *Linq*⁷ 3.0, es *Open Source* y está creado para la tecnología ASP.Net. Provee un lenguaje de consultas que permite la extracción de datos de la base de datos de manera muy simple sin menospreciar la potencia. Además de lo mencionado, ofrece en la estructura de clases generadas una separación clara que posibilita implementar el patrón MVC en la aplicación (20).

Rendimiento:

El rendimiento en *Subsonic* no está garantizado del todo debido a que se deja en manos del usuario la elección de donde se va a guardar el resultado en caché, ya sea a nivel de aplicación o a nivel del propio *Subsonic* (21).

Compatibilidad con el gestor de base de datos PostgreSQL:

Subsonic está creado específicamente para el gestor de base de datos PostgreSQL por lo que su compatibilidad está garantizada (22).

Uso del Lenguaje PHP:

Al estar creado sobre y para tecnología ASP.Net el lenguaje es Java.

Compatibilidad con el MT Sauxe:

La compatibilidad de *Subsonic* con el MT Sauxe se ve limitada debido a que los dos utilizan lenguajes diferentes, *Subsonic* usa Java y Sauxe PHP.

⁷ **Linq:** Language Integrated Query (LINQ) es un proyecto de Microsoft que agrega consultas nativas semejantes a las de SQL a los lenguajes .Net.

Comunidad que respalde su soporte:

Subsonic cuenta con una comunidad poco abundante y su reputación está limitada sobretodo porque es bastante joven.

Buenas prácticas:

- ▶ Los objetos de negocio deben comunicarse con los objetos de acceso a datos a través de interfaces, es decir, siempre depende de abstracciones.
- ▶ Los objetos de acceso a datos deben implementar interfaces definidas por el "cliente" en la capa de lógica de negocio.

1.3.5. Doctrine 1.2.2

Doctrine es un ORM escrito en PHP que proporciona una capa de persistencia para objetos PHP. Es una capa de abstracción que se sitúa justo encima de un sistema de gestión de base de datos. Además es la versión actual que utiliza el MT Sauxe para tratar la persistencia de los datos.

1.3.6. Doctrine 2.0.6

Doctrine 2 es la versión superior de ORM 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 (23).

Doctrine 2 implementa diseño Mapa de Datos (del inglés *Data Mapper*) proponiendo que los objetos de negocio sean POCO⁸ (POJO en Java), o sea, que no tengan nada de código que los acople a una tecnología de persistencia, al no ser 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.

Rendimiento (23)

- ▶ El uso de PHP 5.3 brinda un gran rendimiento.
- ▶ Mejor hidratación y algoritmo optimizado.
- ▶ Nuevas implementaciones de consulta y el resultado cache.
- ▶ El código más explícito y menos mágico, como resultados: el código mejor y más rápido.

El rendimiento en *Doctrine 2* es muy superior a *Doctrine 1*. En la **Tabla I** se muestra un estudio realizado a un sistema haciendo uso de dos versiones diferentes de *Doctrine* (*Doctrine 1.2.2* y

⁸ **POCO**: Componentes Portables (en inglés *Portable Components*).

Doctrine 2.2.0), la comparación se hizo teniendo en cuenta el tiempo de ejecución a la hora de insertar 5000 y 10000 registros respectivamente en una base de datos.

Tabla I. Comparación de rendimiento en las versiones de Doctrine (23).

Versión	Registros	Tiempo	Registros	Tiempo
<i>Doctrine 1</i>	5000	4,3 seg.	10000	7 seg.
<i>Doctrine 2</i>	5000	1,4 seg.	10000	3,5 seg.

Compatibilidad con el gestor de base de datos PostgreSQL:

Doctrine 2 soporta múltiples gestores de base de datos (*MySQL*, *PostgreSQL*, *SQLite*, *MSSQL* y *Oracle*).

Uso del Lenguaje PHP:

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

Compatibilidad con el MT Sauxe:

La compatibilidad con el MT Sauxe está garantizada debido a que este utiliza en su CAD la versión previa de este marco de persistencia de datos.

Comunidad que respalde su soporte:

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

Buenas prácticas (23):

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

Resumen del estudio de los ORM:

El estudio de los ORM de persistencia de datos permitió conocer que el más usado para PHP es *Doctrine* (13). Demostró las potencialidades que representa el uso de la versión *Doctrine 2* debido a que esta brinda un alto rendimiento y en ella se encuentran reflejadas las potencialidades que implementan los demás ORM. Además se conoce que en trabajos anteriores se logró la integración de esta versión con el MT Sauxe garantizando su funcionamiento, actualmente resta por hacer la migración de todos los subsistemas del MT

Sauxe para que utilicen el lenguaje de consulta que implementa *Doctrine 2* y garantizar un aumento de su rendimiento.

1.4. Tecnologías y herramientas para el desarrollo

El uso de tecnologías apropiadas y herramientas de software libre pueden influir en el éxito de las aplicaciones web debido a que se usan como forma de ahorro y maximización de beneficios por su fácil utilización y sostenibilidad.

1.4.1. Subversion v1.6.6

Subversion es un sistema de control de versiones libre (código abierto) que maneja ficheros y directorios a través del tiempo. Contiene un árbol de ficheros en un repositorio central y el repositorio es 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. En este sentido, se piensa en un sistema de control de versiones como una especie de "máquina del tiempo" (24).

Utiliza el concepto de revisión para guardar los cambios producidos en el repositorio. Entre dos revisiones sólo guarda el conjunto de modificaciones, optimizando así al máximo el uso de espacio en disco. Permite al usuario crear, copiar y borrar carpetas con la misma flexibilidad con la que lo haría si estuviese en su disco duro local. Dada su flexibilidad, es necesaria la aplicación de buenas prácticas para llevar a cabo una correcta gestión de las versiones del software generado (25).

1.4.2. RapidSVN v0.12.0

RapidSVN es una interfaz gráfica de usuario multiplataforma para el sistema de revisión de *Subversion* escrito en C++. Con *Subversion* se quiere construir un cliente visual superior que utilice las mejores características de los clientes de otras arquitecturas de control de revisiones. Si bien es bastante fácil para los nuevos usuarios de *Subversion* trabajar con él, también, debe ser lo suficientemente potente como para hacer que los usuarios experimentados sean aún más productivos (26).

1.4.3. PostgreSQL v9.1

PostgreSQL es un sistema de gestión de bases de datos objeto-relacional, distribuido bajo licencia BSD (del inglés *Berkeley Software Distribution*) para bases de datos y con su código fuente disponible libremente. Es el sistema de gestión de bases de datos de código abierto más potente del mercado y en sus últimas versiones ha logrado incorporar las potencialidades de

otros sistemas gestores de bases de datos incluso comerciales. *PostgreSQL* utiliza un modelo cliente-servidor y es multiprocesos en vez de multihilos para lograr la estabilidad del sistema. Un fallo en uno de los procesos no afectará el resto y el sistema continuará funcionando (27).

Durante su desarrollo las características que más se evidencian son: estabilidad, potencia, robustez, facilidad de administración e implementación de estándares. Este presenta un rendimiento elevado con grandes cantidades de datos y una alta concurrencia de usuarios accediendo a la vez al sistema (27).

Soporta distintos tipos de datos: tipo fecha, monetarios, elementos gráficos, datos sobre la red, cadenas de bits y permite la creación de tipos propios. Incluye herencia entre tablas, posee múltiples métodos de autenticación, contiene documentación completa y es multiplataforma (27).

1.4.4. NetBeans v7.4

NetBeans IDE es un entorno de desarrollo integrado (IDE, del inglés *Integrated Development Environment*), modular, de base estándar (normalizado), escrito en el lenguaje de programación Java. El proyecto *NetBeans* consiste en un IDE de código abierto y una plataforma de aplicación, las cuales pueden ser usadas como una estructura de soporte general (framework) para compilar cualquier tipo de aplicación (28).

La plataforma de *NetBeans* es principalmente un MT 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 SDK⁹, *NetBeans* IDE (28).

1.4.5. Apache 2.2

Apache es un servidor web gratuito y potente, que ofrece un servicio estable y sencillo de mantener y configurar. Es válido destacar las siguientes características: es multiplataforma (aunque idealmente está preparado para funcionar bajo Linux), muy sencillo de configurar, es *open-source* (código abierto), muy útil para proveedores de servicios de internet que requieran miles de sitios pequeños con páginas estáticas, amplias librerías de *PHP* y *Perl* a disposición de

⁹ **SDK**: Kit de desarrollo de software o SDK (siglas en inglés de *Software Development Kit*).

los programadores, posee diversos módulos que permiten incorporarle nuevas funcionalidades y es capaz de utilizar lenguajes como *PHP*, *TCL*, *Python*, entre otros (29).

Apache es un software libre y el servidor web más popular. Algunos estudios realizados demuestran que más del 70% de los sitios web en internet están manejados por *Apache*, haciéndolo más extensamente usado que todos los otros servidores web juntos. Es un servidor web flexible, rápido y eficiente, continuamente actualizado y adaptado a los nuevos protocolos HTTP (29).

1.4.6. Sauxe 2.2

Contiene un conjunto de componentes reutilizables que proveen la estructura genérica y el comportamiento para una familia de abstracciones, logrando una mayor agilidad en el proceso de desarrollo y las aplicaciones de gestión. Sauxe está basado en el MT Zend, además utiliza en la CAD el Lenguaje de Consulta de *Doctrine* (DQL). 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 amigable (4).

1.4.7. Marco de Trabajo Zend 1.11

Zend es un MT 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*”) (30).

Aunque se pueden utilizar de forma individual, los componentes de la biblioteca estándar de Zend conforman un potente y extensible MT de aplicaciones web al combinarse. Este ofrece un gran rendimiento y una robusta implementación del MVC, una abstracción de base de datos fácil de usar, y un componente de formularios que implementa la prestación de formularios HTML¹⁰, validación y filtrado para que los desarrolladores puedan consolidar todas las operaciones usando de manera sencilla la interfaz orientada a objetos (30).

1.4.8. PHP 5.3.10

PHP cuyas siglas responden a un acrónimo recursivo (PHP: *Hypertext Preprocessor*) es un lenguaje de programación interpretado, multiplataforma, originalmente diseñado para la

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

creación de páginas web dinámicas con acceso a información almacenada en una base de datos. Sigue un estilo clásico debido a que es un lenguaje de programación con variables, sentencias condicionales, bucles y funciones. Está más cercano a JavaScript o al lenguaje C, pero a diferencia de Java o JavaScript que se ejecutan en el navegador, PHP se ejecuta en el servidor (31).

El código fuente escrito en PHP es invisible al navegador y al cliente porque es el servidor el que se encarga de ejecutar el código y enviar su resultado HTML al navegador. Esto hace que la programación en PHP sea segura y confiable. Es libre, por lo que se presenta como una alternativa de fácil acceso para todos y permite aplicar técnicas de programación orientada a objetos (33).

Tiene manejo de excepciones (desde PHP5). Capacidad de conexión con la mayoría de los motores de base de datos que se utilizan en la actualidad, destaca su conectividad con MySQL y PostgreSQL (33).

Resumen del estudio de las herramientas:

Las herramientas antes mencionadas son las definidas en el documento “*Vista del Entorno de Desarrollo Tecnológico del proyecto Sauxe_v2.2*” establecido por el Departamento de Tecnologías donde se desarrolló el MT Sauxe. Se hace un estudio de estas para destacar algunas de sus potencialidades y conocerlas para su utilización durante la implementación de la migración de la CAD del subsistema EyC.

1.5. Conclusiones parciales

En este capítulo se han expresado los conceptos principales que se necesitan para comprender los elementos que maneja la investigación y llevar a cabo la migración, como la persistencia de los datos y los ORM.

El análisis realizado sobre algunos ORM utilizados para la persistencia de datos, demostró que estos exhiben características comunes y que en *Doctrine 2* se encuentran implementadas las principales potencialidades de los otros. Teniendo en cuenta lo antes mencionado, se demuestra que las buenas prácticas se pueden reutilizar en la versión 2 de *Doctrine* al brindar este un alto rendimiento.

El estudio de las herramientas y tecnologías, definidas por el Departamento de Tecnologías para el desarrollo de sus aplicaciones, contribuyó a sentar las bases de la investigación y de esta forma poder dar solución a la problemática existente y cumplir con el objetivo planteado.

CAPÍTULO 2: DESARROLLO DE LA MIGRACIÓN

2.1. Introducción

En este capítulo se describe la implementación realizada en la migración de la CAD del subsistema EyC a *Doctrine 2*. Se destacan sus elementos más importantes y se especifican las nuevas configuraciones y estructura de la CAD. Además se hace una descripción detallada de las principales acciones que se deben cumplir para realizar la migración a *Doctrine 2* definidas en trabajos anteriores. La migración se realiza siguiendo un procedimiento establecido con anterioridad el cual define tres etapas: Inicial, Implementación y Pruebas. De estas solo se describen en este capítulo la fase Inicial y la de Implementación.

2.2. Contenido de la migración

La migración del subsistema EyC del MT Sauxe contiene las tareas específicas propuestas en el trabajo “Migración de la capa de acceso a datos del marco de trabajo Sauxe”. Este procedimiento propone una guía para la realización de la migración de los componentes de Sauxe y de las soluciones que utilizan este MT a *Doctrine 2*. Cuenta además con la especificación de las etapas del proceso logrando una relación entre estas y mostrando una secuencia lógica y organizada para su correcta ejecución. A continuación se describen las etapas definidas en el procedimiento y su implementación en el subsistema EyC.

2.3. Etapa Inicial

En esta etapa se procede a realizar un análisis de la arquitectura de la aplicación para evaluar el contenido de cada una de las capas que contiene. Por otra parte se garantiza la integración del ORM *Doctrine 2* con el MT Zend al utilizar la versión 2.2 del MT Sauxe.

2.3.1. Arquitectura del MT Sauxe

La arquitectura del MT Sauxe está basada en capas y aunque en una de ellas implementa el patrón MVC básicamente está compuesto por cinco niveles o capas:

1. Capa de Presentación: En esta capa se emplean las facilidades que brinda el MT ExtJS, biblioteca de JavaScript para la construcción de interfaces a la vista de los usuarios. ExtJS centra su desarrollo en tres componentes fundamentales *JS-File*, *CSS-File* y *Client-page* y *Server-Page* (4).

2. Capa de Control o Negocio: En esta capa se emplea el patrón de MVC. De forma vertical al modelo, estarán los aspectos fundamentales del sistema así como el encargado del tratamiento de la seguridad a nivel de aplicación Web (4).

3. Capa de Acceso a Dato: En esta capa estará presente el ORM *Doctrine* para la comunicación con el servidor de datos mediante el protocolo PDO¹¹, también estará un Persistidor de Configuración que es el encargado de comunicarse vía XML con los Ficheros de Configuración del sistema denominado *FastResponse* (4).

4. Capa de Dato: En esta capa estará ubicado como servidor de base de datos PostgreSQL y un conjunto de Ficheros de Configuración de la arquitectura tecnológica (4).

5. Capa de Servicio: En esta última capa se encuentran todos los subsistemas que prestan y consumen servicios entre sí.

Esta gran estructura descrita, se comunica con una serie de servicios Web mediante protocolos SOAP¹², e IoC¹³, que interactúan con el sistema proporcionándole un conjunto de funcionalidades tanto de seguridad como de negocio (4).

El contenido de la migración está centrado en la CAD, debido a que en ella está presente el ORM *Doctrine* el cual se actualizará a la versión 2, con el objetivo de aumentar el rendimiento del subsistema EyC; aunque se debe garantizar su comunicación con las demás capas.

2.3.2. Estructura de carpetas del componente Estructura y Composición en la CAD

Sauxe está compuesto por un conjunto de componentes que presentan la misma estructura de carpetas en la CAD. En la **Figura 1** se muestra el subsistema EyC (*metadatos*) con la estructura de carpetas que garantiza el correcto funcionamiento del MT con la versión 1.2.2 del ORM *Doctrine*.

Es importante realizarle un profundo análisis debido a que sufre modificaciones durante la migración para satisfacer las exigencias de *Doctrine 2*. Además, los cambios que se realicen deben garantizar la comunicación con las otras capas y de esta manera no se afecte ni el negocio ni la interacción con el MT.

¹¹ **PDO:** acrónimo del inglés PHP *Data Objects*, es una extensión que provee una capa de abstracción de acceso a datos para PHP5.

¹² **SOAP:** siglas de *Simple Object Access Protocol* es un protocolo estándar que define cómo dos objetos en diferentes procesos pueden comunicarse por medio de intercambio de datos XML.

¹³ **IoC:** (siglas de *Inversion of Control*) es el mecanismo de integración de los componentes de Sauxe.

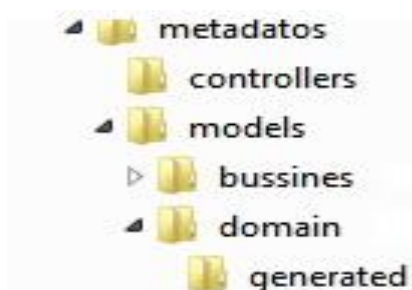


Figura 1. Estructura de carpetas del componente metadatos para *Doctrine 1*.

2.3.3. Integración de *Doctrine 2* con el MT Zend

La actividad de integración es la base para realizar la implementación de la migración y para ello se definen las bibliotecas que se utilizan y las clases que se modifican creando las funcionalidades necesarias. Esta tarea fue objetivo de trabajos anteriores en el Departamento de Tecnologías, logrando que el MT Zend funcione de manera correcta con *Doctrine 2* al utilizar Sauxe en su versión 2.2 o superior (6).

2.4. Etapa de Implementación

En la presente etapa es necesario realizar algunas modificaciones debido a que el funcionamiento en la nueva versión es diferente a la anterior. Una de las tareas es rediseñar la arquitectura base del sistema, donde se debe establecer una nueva estructura de carpetas para el componente EyC; lo que debe garantizar la ubicación correcta de los nuevos ficheros y funcionalidades a utilizar con *Doctrine 2*. Otra de las tareas es redefinir la CAD de este subsistema, donde se mapean las tablas y se definen las relaciones de la base de datos para la versión 2. También se vuelven a implementar todos los métodos y funcionalidades de esta capa utilizando el DQL de la nueva versión.

2.4.1. Nueva estructura de carpetas del componente Estructura y Composición

Doctrine 2 propone la utilización de una nueva estructura de carpetas para las clases del modelo. Contiene nuevos directorios (*Entity*, *Proxy* y *Repository*) con el fin de organizar los ficheros generados por el mapeo, ya sea por su tipo o su función. En la siguiente figura se muestra cómo queda conformada esta nueva forma:

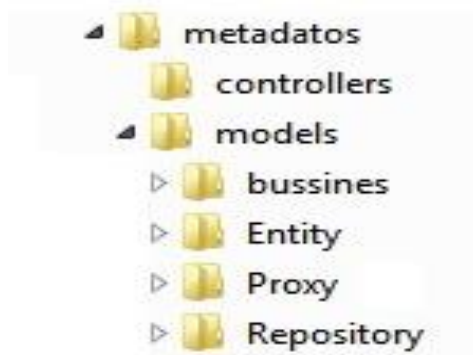


Figura 2. Nueva estructura del componente metadatos para Doctrine 2.

2.4.2. Equivalencia entre la nueva y anterior estructura de carpetas

La estructura anterior y la nueva, como se puede evidenciar en la **Figura** , son diferentes aunque se puede establecer cierta relación entre ellas, puesto que en algunos casos sólo basta con cambiar de ubicación los ficheros. La siguiente figura visualiza la nueva ubicación de los ficheros del modelo del componente “metadatos” y la equivalencia que existe entre las versiones 1.x y las 2.x:

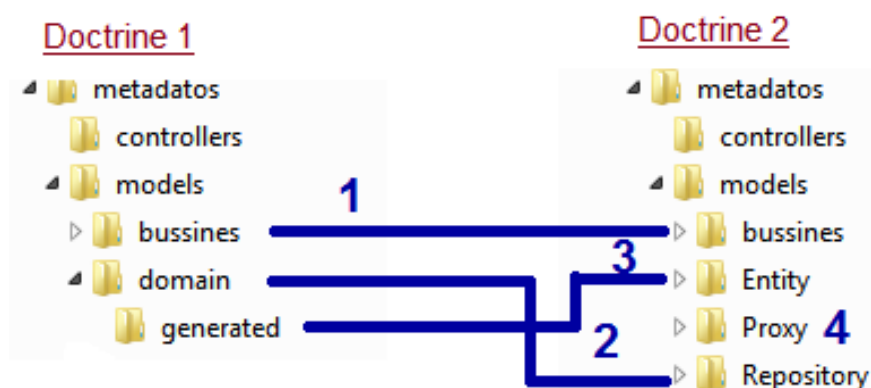


Figura 3. Comparación de las estructuras de carpetas.

1- Los ficheros localizados en la carpeta **bussines** que son los encargados de manejar el negocio de la aplicación se mantienen igual dentro de la carpeta **models**.

2- Las funcionalidades con las consultas que se encuentran en los ficheros localizados en la carpeta **domain** la deben realizar los ficheros ubicados dentro de la carpeta **Repository** y además se deben modificar de acuerdo a la nueva estructura de *Doctrine 2*, utilizando el DQL de esta versión.

En la carpeta **Repository** se encuentran los ficheros *Repository* generados como resultado del mapeo. Por cada tabla en la base de datos se genera un fichero de este tipo, su nomenclatura está compuesta por el nombre del fichero de mapeo de la tabla y la palabra *Repository* al final. Estos ficheros son clases que extienden de la clase **Doctrine\ORM\EntityRepository**.

3- Los ficheros que representan las tablas mapeadas ubicados en **generated** son generados con su nueva estructura en el directorio **Entity** de la carpeta **models**.

4- En la carpeta **Proxy** se ubican los ficheros generados como resultado del mapeo que contienen las clases *Proxy* usadas por *Doctrine 2*. Estas clases son objetos que se ponen en el lugar del objeto real, y se utilizan para realizar varias funciones, 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.

2.4.3. Redefinir Capa de Acceso a Datos

Con la utilización de la nueva estructura de carpetas, necesitada para el trabajo con *Doctrine 2*, se procede a volver a implementar todos los métodos y consultas del subsistema EyC. La primera acción a realizar en este paso, es mapear todas las tablas adaptándolas a la nueva estructura de la versión 2 de *Doctrine* y la migración manual de cada una de las clases ya mapeadas. Esta tarea requiere de un tiempo considerable para su realización, más allá de su complejidad, por el monto de objetos que se deben mapear y por la cantidad de funcionalidades que se implementan nuevamente.

Se mapearon un total de 47 clases, donde por cada una de ellas se generan tres ficheros el *Model*, el *Repository* y el *Entity*, lo que influyó en el cambio de la estructura de carpetas de los componentes de Sauxe. Se modificaron todos los ficheros *Model* del componente, garantizando que se pueda acceder al *Repository* de cada uno, en el cual se encuentran implementadas las consultas. Como resultado de haber vuelto a implementar las funcionalidades se cambiaron más de 360 consultas en la CAD del subsistema EyC.

2.4.4. Flujo de datos de la Capa de Acceso a Datos

Con la nueva versión del ORM *Doctrine*, la CAD adquiere otro funcionamiento interno, donde cambia por completo su flujo de datos. Este transita desde las clases controladoras, accediendo a los *models*, hasta los ficheros *Repository*. En este archivo se establece una conexión a la

base de datos, se realizan las peticiones y se envían los objetos necesarios o que fueron solicitados.

Una característica peculiar de *Doctrine 2* es que en lugar de que cada clase *Model* realice las peticiones directamente a la base de datos, en su lugar, utiliza una instancia del objeto **EntityManager**. Con este se puede acceder a los *Repository* donde se implementan todas las consultas a la base de datos.

En la **Figura** se puede observar el flujo desde que se invoca el modelo hasta que se retorna un resultado, de modo que este sería el funcionamiento de la CAD. Se puede evidenciar que los ficheros *Entity* son resultado del mapeo de las tablas de la base de datos y en los ficheros *Model* se encuentran implementados los métodos necesarios para manejar el negocio del subsistema EyC. Estos ficheros deben ser modificados para adaptarlos a la nueva estructura de carpetas y el nuevo funcionamiento de la CAD. Por su parte, en los ficheros *Repository* se deben implementar todas las consultas, utilizando el DQL de *Doctrine 2*, que se realizan al sistema gestor de base de datos.

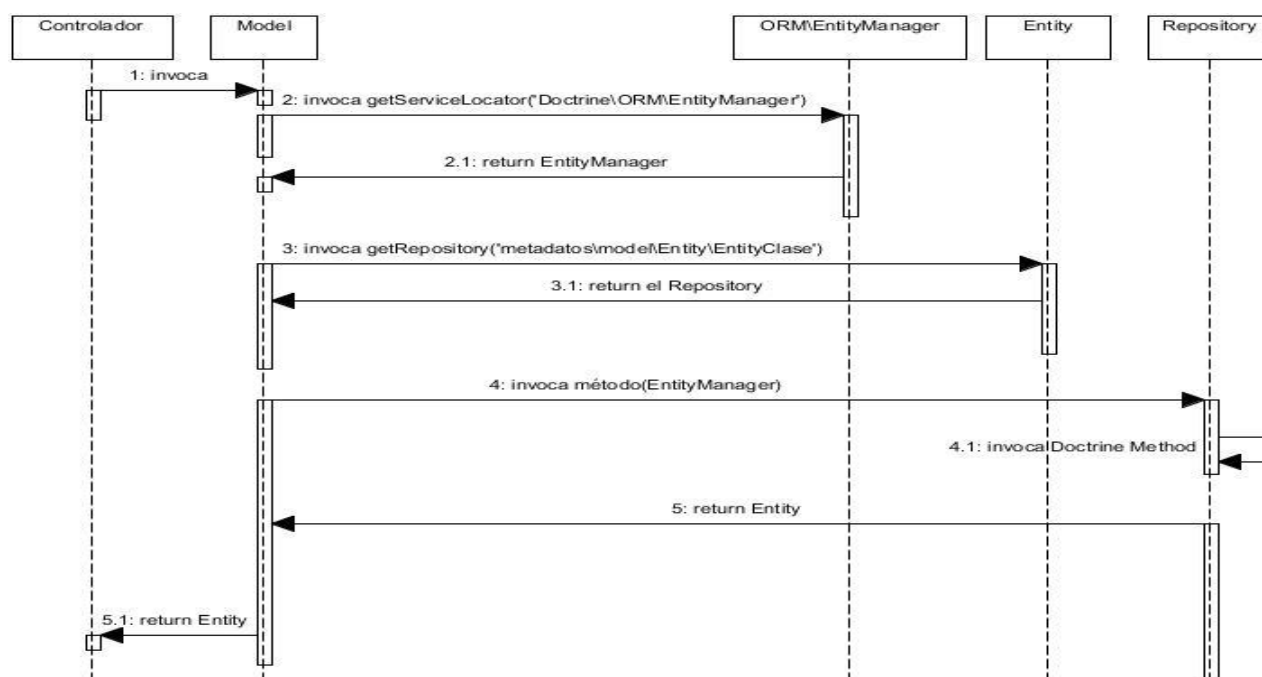


Figura 4. Secuencia desde el Controlador hasta el Repository (Flujo de la CAD).

2.4.5. Mapeo de las tablas

Básicamente las formas de mapeo de *Doctrine 1* y *Doctrine 2* son diferentes, aunque ambas definen el mapeo entre un tipo de dato PHP y un tipo de dato SQL. En el caso de *Doctrine 1*

siempre se generan entidades que extienden de una clase base haciendo uso de funciones predeterminadas para la definición de tablas y columnas. Mientras que *Doctrine 2* proporciona diferentes maneras de especificar el mapeo objeto-relacional (23):

- ▶ Por anotaciones en el código fuente (docblock¹⁴).
- ▶ Desde un archivo con formato YAML.
- ▶ Desde un archivo con formato XML.

Las anotaciones *docblock* de *Doctrine 2* cuentan con apoyo para espacios de nombre (*namespaces*) y anotaciones anidadas. Específicamente para la realización de la migración se emplea la anotación *docblock* porque suministra asignaciones de metadatos objeto-relacional. Es preciso señalar, que en un mismo entorno no se pueden mezclar diferentes formas de definir los metadatos. Si se utiliza, por ejemplo YAML en una entidad, no se puede utilizar anotaciones PHP en otras entidades.

Doctrine permite guardar y obtener objetos enteros a partir de la información de la base de datos. Para esto mapea una clase PHP a una tabla de la base de datos y después mapea las propiedades de la clase PHP a las columnas de esta tabla; en la siguiente figura se puede apreciar esta acción:

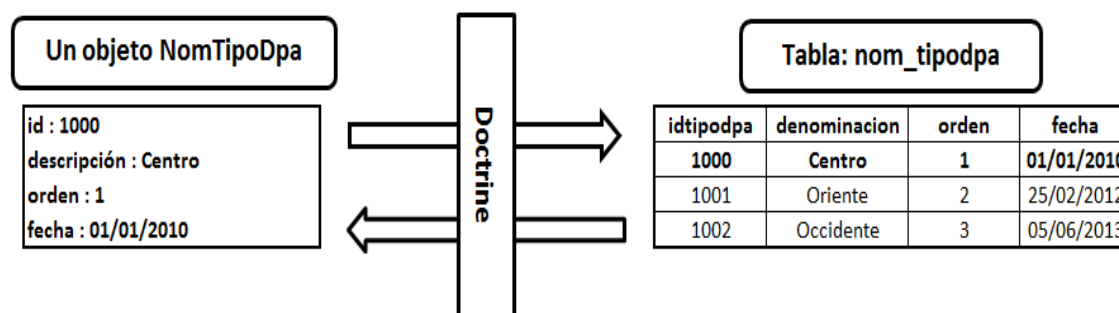


Figura 5. Mapeando un objeto PHP a una tabla.

Mapear la persistencia de una clase y definir una tabla

Para *Doctrine 1.x* se genera una clase que extiende de una base llamada *Doctrine_Record* y una funcionalidad llamada *setTableDefinition*:

```
abstract class BaseDatEstructura extends Doctrine_Record {  
    public function setTableDefinition () {  
        $this->setTableName ('mod_estructuracomp.dat_estructura');  
        ...  
    }  
}
```

¹⁴ **Docblock**: es un tipo especial de comentario en el código que puede proporcionar información detallada acerca de un elemento.

```
    }  
} ...
```

Por su parte *Doctrine 2.x*, con el fin de marcar una clase para ser persistida en la base de datos, lo hace con la anotación `@Entity`, por defecto la entidad será persistida con el mismo nombre de la clase, pero se puede modificar utilizando la anotación `@Table`:

```
<?php
```

```
namespace metadatos\models\Entity;
```

```
/**
```

```
 * DatEstructura
```

```
 * @Table (name = "mod_estructuracomp.dat_estructura")
```

```
 * @Entity (Repository = "metadatos\models\Repository\DatEstructuraRepository")
```

```
 */
```

```
 ...
```

Con esto ahora las instancias de *DatEstructura* serán persistidas en una tabla llamada *dat_estructura* en el esquema *mod_estructuracomp* en la base de datos.

Mapear la persistencia de las propiedades

Las propiedades, que son representadas en forma de columnas; *Doctrine 1* lo hace con la función `hasColumn ()` y se definen cada uno de los parámetros, mientras que *Doctrine 2* lo hace de manera similar pero utilizando las anotaciones y empleando los tipos de mapeo:

- ▶ **type**: opcional, por defecto "string", establece el tipo de mapeo a usar para la columna.
- ▶ **name**: opcional, por defecto el nombre de la propiedad, establece el nombre de la columna en la base de datos.
- ▶ **length**: opcional, por defecto 255, la longitud de la columna en la base de datos, solo si es un campo string.
- ▶ **unique**: opcional, por defecto FALSE, establece si la columna es una clave única.
- ▶ **nullable**: opcional, por defecto FALSE, establece si la columna puede ser nula.
- ▶ **precision**: opcional, por defecto 0, establece la precisión para una columna decimal, solo si es un campo decimal.
- ▶ **scale**: opcional, por defecto 0, establece la escala para una columna decimal, solo si es un campo decimal.

Doctrine 1

```
$this->hasColumn ('idestructura', 'decimal', null, array ('notnull' => true, 'primary' => true, 'sequence' => 'seq_datestructura'));
```

Doctrine 2

```
/**
 * @var float
 * @Id
 * @Column (name="idestructura", type="decimal", nullable=false)
 * @GeneratedValue (strategy="SEQUENCE")
 * @SequenceGenerator (sequenceName=
 "mod_estructuracomp.dat_estructura_idestructura_seq", allocationSize=1, initialValue=1)
 */
private $idestructura;
...
```

Cada clase *Entity* necesita una clave primaria que identifique unívocamente a la entidad. Se designa la propiedad que se usará como identificador con la anotación **@Id** y si es un valor generado automáticamente se utiliza la anotación **@GeneratedValue** y como estrategia **SEQUENCE** a través de **@SequenceGenerator** (23).

Relaciones entre tablas

La representación de las relaciones entre tablas es una de las cuestiones esenciales para realizar la representación de entidades de mapeo. La siguiente figura presenta cómo *Doctrine* gestiona las entidades relacionadas:

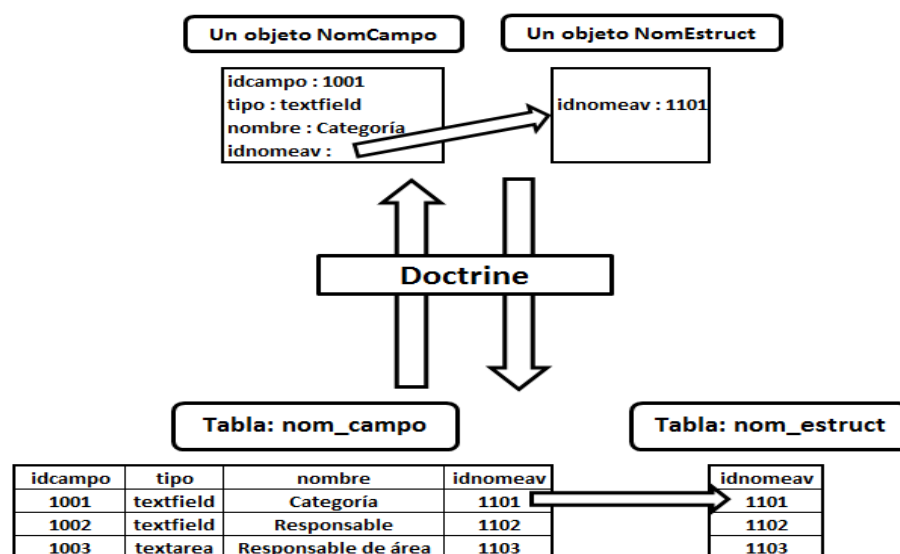


Figura 6. Cómo gestiona *Doctrine* las entidades relacionadas.

En *Doctrine 1* se definen a través de la funcionalidad **setUp ()** las relaciones existentes entre las tablas como por ejemplo en la clase **BaseNomGradomilit extends Doctrine_Record** se tiene:

```
parent :: setup ();  
$this->hasOne (“NomGsubcateg”, array (‘local’ => ‘idsubcateg’, ‘foreign’ => ‘idsubcateg’));  
...
```

Donde se especifica que se tiene una instancia de **NomGsubcateg** para muchos **NomGradomilit**, es decir, un **NomGradomilit** tiene una única instancia del objeto mapeado por la entidad **NomGsubcateg**.

Doctrine 2

Para que exista una relación donde se obtenga una única instancia de un objeto se utiliza: **@ManyToOne** y **@JoinColumn**, ejemplo:

```
/**  
 * @var \NomGsubcateg  
 * @ManyToOne (targetEntity = “NomGsubcateg”)  
 * @JoinColumn (name = “idgsubcateg”, referencedColumnName = “idgsubcateg”)  
 */  
private $idgsubcateg;  
...
```

La asociación de **uno a muchos** es la relación más común cuando se trabaja con modelos. Para que exista una relación de uno a muchos es importante definir dos anotaciones: **@OneToMany** y **@JoinColumn**, por ejemplo se tiene en la clase **NomCampoestruct**.

```
/**  
 * @var \NomTipo  
 * @OneToMany (targetEntity = “NomTipo”, mappedBy = “tipo”)  
 * @JoinColumn (name = “tipo”, referencedColumnName = “tipo”)  
 */  
private $tipo;  
...
```

Esto especifica que el objeto **NomCampoestruct** puede tener muchas instancias del objeto **NomTipo**. En algunos casos las relaciones son bidireccionales y se debe entonces utilizar en una clase la anotación **@OneToMany** y en la correspondiente a la relación emplear **@ManyToOne**.

Doctrine 1

En el caso en que la relación es de Mucho-Mucho *Doctrine* 1 utiliza la siguiente manera:

```
parent :: setup ();  
$this->hasMany (“NomFilaestruct”, array (‘local’ => ‘idnomeav’, ‘foreign’ => ‘idnomeav’));  
...
```

Aquí se evidencia, como el fragmento de código pertenece a la clase **NomNomencladoreavestruc**, donde se tienen varias instancias del objeto **NomFilaestruct** para muchos objetos **NomNomencladoreavestruc**.

Doctrine 2

En el caso particular de *Doctrine* 2 no se genera un fichero por la tabla en base de datos que guarda las llaves primarias de las tablas que se unen, en este caso se hace referencia a la entidad con la que se tiene la relación y con la anotación **@JoinTable** se le dice cuál tabla es la relacionada, esto se evidencia a continuación:

```
/**  
 * @var \NomNomencladoreavestruc  
 * @ManyToMany (targetEntity = “NomNomencladoreavestruc”, inversedBy = “idnomeav”)  
 * @JoinTable (name = “mod_estructuracomp.nom_filaestruct”,  
 *     joinColumns = (  
 *         @JoinColumn (name = “idnomeav”, referencedColumnName = “idnomeav”)),  
 *     inverseJoinColumns = (  
 *         @JoinColumn (name = “idnomeav”, referencedColumnName = “idnomeav”))  
 * )  
 */  
private $idnomeav;  
...
```

En todos los ejemplos anteriores a la hora de tratar las relaciones, es buena práctica inicializar los contenedores de objetos como un arreglo de colecciones en el constructor. Esto se realiza con el objeto de *Doctrine* **ArrayCollection ()** para que este no se quede indefinido, por ejemplo:

```
/**  
 * Constructor  
 */  
public function _construct () {  
$this-> idnomeav = new \Doctrine\Common\Collections\ArrayCollection ();
```

```
} ...
```

Con lo antes descrito se evidencia que las anotaciones *docblock* se pueden ver como comentarios que se emplean en clases PHP, que para el caso de *Doctrine 2* este las utiliza para garantizar el mapeo de las entidades. Además, se pueden representar todas las formas de mapeos que se utilizaban con la versión anterior realizando las mismas funciones.

2.4.6. Creación de las funcionalidades en las clases *Repository*

Dentro de la CAD, como se mencionó anteriormente, todas las consultas y funcionalidades serán implementadas en los ficheros *Repository* utilizando el DQL de *Doctrine 2*. En la versión anterior, la implementación de estas se realizaba en el directorio *domain* y utilizaban el DQL de *Doctrine 1*. Esto se realizaba haciendo uso de varias funciones definidas por el propio lenguaje de consultas de *Doctrine 1*, las cuales realizaban muchas peticiones al navegador afectando así el rendimiento de la aplicación. La manera en que quedan implementadas se representa de la siguiente manera:

```
public function buscarGradomtar ( $limit, $start, $denom ) {  
    $query = Doctrine_Query :: create ($conn);  
    $denom = strtolower ($denom);  
    $result = $q->select ('nm.*,a.*,s.*,ct.*')  
        ->from ('NomGradomilit nm')  
        ->leftJoin ('nm.NomGradomilit a')  
        -> leftJoin ('nm.NomGradomilit s')  
        -> innerJoin ('nm.NomGsubcateg ct')  
        -> where ("LOWER(nm.dengradomilit) like '%$denom%'")  
        -> setHydrationMode ( Doctrine :: HYDRATE_ARRAY)  
        -> limit ($limit)  
        -> offset ($start)  
        -> orderBy ('orden')  
        -> execute ();  
return $result;  
...  
}
```

La implementación de la funcionalidad mostrada anteriormente, después de volver a implementarla quedaría de la siguiente manera en *Doctrine 2*:

```
public function buscarGradomtar ( $limit, $start, $denom, $_em ) {
```

```
$denom = strtolower ($denom);  
$dql = "SELECT nm, a, s, ct FROM metadatos\models\Entity\NomGradomilit nm  
LEFT JOIN nm.antecesor a LEFT JOIN nm.sucesor s INNER JOIN nm.idgsubcateg ct  
WHERE LOWER (nm.dengradomilit) LIKE '%?1%' ORDER BY nm.orden";  
$query = $_em -> createQuery ($dql);  
$query = setParameter (1, $denom);  
$query = setMaxResults ($limit);  
$query = setFirstResult ($start);  
$result = $query -> getArrayResult ();  
return $result;  
}  
...
```

Es necesario destacar que todas las funcionalidades que se implementen en los ficheros *Repository* deben recibir por parámetro una instancia de la entidad *EntityManager* (*\$_em*) además de los necesarios. Esta instancia es la encargada de crear las consultas a través del método ***createQuery*** (*\$dql*), que recibe por parámetro un “*string*” el cual será convertido a SQL y procesado por el sistema gestor de base de datos. Además, se puede evidenciar como se reduce el número de peticiones al navegador y el manejo de la consulta en texto plano lo cual contribuye al aumento del rendimiento de la aplicación.

Algunas variaciones que sufren las funcionalidades (23):

- ▶ Manejo de los parámetros: se realiza utilizando parámetros posicionales, los cuales se especifican con números, por ejemplo: “?1”, “?2” y así sucesivamente. Estos son asignados con la función ***setParameter*** (1, *\$var*) o con ***setParameters*** (array (*\$parameters*)).
- ▶ Utilización de funciones agregadas: se emplean al igual que en SQL y son compatibles la mayoría, debido a que el DQL de *Doctrine 2* ofrece una gama de expresiones adicionales que provienen del SQL.
- ▶ Formato del resultado: El formato del resultado de determinada consulta está influenciado por el *modo de hidratación*. Este especifica de manera particular en que se transforma un conjunto de resultados SQL. Cada modo tiene su propio método de hidratación dedicado en la clase *Query*:
 - ✓ ***getResult*** (): Recupera una colección de objetos.
 - ✓ ***getSingleResult*** (): Recupera un único objeto.

- ✓ **getArrayResult ()**: Recupera una matriz anidada de objetos.
- ✓ **getScalarResult ()**: Recupera un resultado plano del conjunto de valores escalares que puede contener datos duplicados.
- ✓ **getSingleScalarResult ()**: Recupera un valor escalar único a partir del resultado devuelto.

► Primer elemento y máximos resultados: Mientras en *Doctrine 1* se utilizan las funciones **offset** (*\$start*) y **limit** (*\$limit*) para indicar estos valores, en la versión 2 se utiliza **setMaxResults** (*\$limit*) y **setFirstResult** (*\$start*).

2.4.7. Funcionalidades en las clases *Model*

Las clases *Model* se encargan de manejar el negocio de la CAD y se mantienen en el mismo directorio en la nueva estructura de carpetas. Estas son nombradas con la formación del nombre de la tabla y la palabra *Model* al final, extienden todas de la clase **ZendExt_Model** y su estructura es la siguiente:

```
class CampoModel extends ZendExt_Model {  
    public function CampoModel () {  
        parent :: ZendExt_Model ();  
        $this -> instance = new metadatos\models\Entity\NomCampoestruct ();  
    } ...  
} ...
```

En estos ficheros se realizan las llamadas a las funcionalidades de las clases *Repository* antes mencionadas. Para ello se crea una instancia de la clase **TransactionManager** y con la utilización del método **getConnection** ('metadatos') se obtiene una instancia del *EntityManager*. Esta instancia es la que permite acceder al *Repository* de determinada entidad haciendo uso del *namespace* de la clase y a través de la funcionalidad **getRepository** ('\$namespace') se solicita la funcionalidad deseada. A continuación se evidencia cómo se accede desde las clases *Model* a las funcionalidades de los *Repository*:

```
public function buscarGradomtar ($limit = 1000, $start = 0, $denom = false) {  
    $mg = ZendExt_Aspect_TransactionManager :: getInstance ();  
    $_em = $mg -> getConnection ('metadatos');  
    $result = $_em -> getRepository ('metadatos\models\Entity\NomGradomilit')  
        -> buscarGradomtar ($limit, $start, $denom, $_em);  
    return $result;  
} ...
```

En el caso anterior se utiliza el *namespace* “*metadatos\models\Entity\NomGradomilit*”, es decir se está utilizando el *Repository* de la *Entity* *NomGradomilit*. Además se puede apreciar el uso del parámetro *\$_em* y la llamada a la funcionalidad ***buscarGradomtar*** (*\$limit*, *\$start*, *\$denom*, *\$_em*).

2.4.8. Redefinir Capa de Negocio

Se redefine la capa de negocio donde se realizan cambios necesarios para la migración, aunque no fue en su totalidad. Se adaptaron las funcionalidades a la nueva estructura de la CAD después de haber vuelto a implementarlas. Se analizaron todos los servicios del subsistema EyC y las llamadas a las funcionalidades con el objetivo de eliminar el empleo de malas prácticas y establecer una mejor organización en el MT Sauxe. Algunas de las malas prácticas que se empleaban son las siguientes:

- ▶ Llamadas directamente a las funcionalidades sin la utilización de las clases *Model*.
- ▶ Implementación de consultas y funcionalidades dentro de los ficheros *Model* encargados del negocio.
- ▶ Acceso desde subsistemas directamente a las funcionalidades de otro, sin el empleo de los servicios que se encargan precisamente de establecer una comunicación entre varios subsistemas.

Con esta nueva estructuración, todo el negocio de EyC se debe encontrar dentro de las clases *Model* y como consecuencia de utilizar esta estructura es necesario redefinir la forma en que se utilizan las funcionalidades. Para ello se debe cambiar la manera de realizar las llamadas a las funcionalidades, que como se mencionó anteriormente, se encontraban en el directorio *domain* y para la nueva versión desaparece este directorio.

Por consiguiente, se realizan las llamadas de las funcionalidades estáticas o no, que se encuentran en las clases *Model* que son las que se encargan de realizar el acceso a las funcionalidades de los *Repository*. Como parte de utilizar una buena práctica se crea una instancia de la clase a la que se hace referencia y después se realiza la llamada a la funcionalidad en uso. A continuación un ejemplo:

```
function mostrarTipoSubordinacionAction () {  
    $subordinacionModel = new NomSubordinacion ($denominacion);  
    $total = $subordinacionModel -> contarNomSubordinacion ($denominacion);  
    $tablas = $subordinacionModel -> obtenerNomSubordinacion ($limit, $start, $denom);  
    ... } ...
```

2.4.9. Actualizar estado de los elementos de configuración

Anteriormente se mencionaron los cambios realizados a los ficheros de configuración. Después de realizadas las acciones hasta este punto, es necesario realizar la actualización en el repositorio de la aplicación. Además, se deben aplicar los cambios a los elementos de configuración, realizar una revisión de las actualizaciones e integrarlos. También se publican los elementos actualizados para su posterior uso en el MT Sauxe.

2.5. Etapa de Pruebas

En esta etapa se realizan pruebas a todas las funcionalidades del subsistema EyC para garantizar que sean operativas y que los cambios realizados sobre estas no afectan su comportamiento. Por otra parte, se garantiza que la aplicación cumple con las características planteadas por los clientes al someter la aplicación a una revisión técnica.

2.5.1. Pruebas realizadas

Utilizando los casos de prueba del subsistema EyC definidos en el Expediente de Proyecto del MT Sauxe, se realizaron las pruebas pertinentes con el objetivo de verificar que los cambios realizados no afecten el funcionamiento de este subsistema. Para realizar esta tarea, se creó una comisión en el Departamento de Tecnologías encargada de someter la aplicación migrada a una revisión técnica. Realizaron las pruebas varios especialistas del proyecto los cuales interactuaron con la aplicación por la red y emitieron un informe con los errores encontrados.

Tabla II. Resultados de las pruebas realizadas.

	Primera Iteración	Segunda Iteración
Errores encontrados	10	0

La presente revisión determinó que la herramienta está estable y lista para su uso después de realizarse dos iteraciones. La comisión emitió el acta de liberación evidenciando que la aplicación cumple con las expectativas del cliente definidas de igual manera en el Expediente de Proyecto de Sauxe.

2.5.2. Errores más comunes

En las pruebas realizadas los errores más comunes son la manera de devolver los resultados y la forma en que se capturan las excepciones, las cuales son diferentes en las dos versiones. Esto afecta el manejo de los datos recibidos en las vistas a la hora de mostrarlos; mientras que el manejo de las excepciones contribuye a que no se gestionen bien los mensajes de notificación al realizar operaciones que no se puedan ejecutar.

2.6. Conclusiones parciales

En este capítulo se realizó un estudio de la arquitectura del subsistema EyC y de su estructura de carpetas, lo que permitió entender el conjunto de modificaciones realizadas sobre la nueva CAD para interactuar con el ORM *Doctrine 2*.

Con la migración de la CAD del subsistema EyC, en la cual se estableció una nueva estructura de carpetas y se redefinió tanto la CAD como la Capa de Negocio, se eliminó el uso de malas prácticas y se logró una mejor organización en el código y en los ficheros generados del mapeo.

Las pruebas realizadas por la comisión del Departamento de Tecnologías sobre cada una de las funcionalidades del subsistema EyC, permitieron demostrar que cumplen con las expectativas de los clientes.

Con el cumplimiento de las tareas definidas para la realización del proceso de migración a *Doctrine 2*, se logró la implementación de la migración de la CAD del subsistema EyC, lo que permitirá un aumento en el rendimiento de la aplicación.

CAPÍTULO 3: VALIDACIÓN DE LA MIGRACIÓN

3.1. Introducción

En este capítulo se valida la migración realizada mediante el uso de la herramienta JMeter que permite evidenciar el rendimiento de la aplicación. Se utiliza un entorno de trabajo en igualdad de condiciones para realizar una comparación del sistema tanto con la versión anterior como con la nueva. Se realizan pruebas de rendimiento que demuestren si el sistema cumple con diferentes criterios y permitan conocer las estadísticas de los procesos que se ejecutan durante su utilización.

3.2. Pruebas de rendimiento

Como parte del cumplimiento del procedimiento de la migración se realizan pruebas de rendimiento, las cuales se centran en determinar la velocidad con la que el sistema, bajo pruebas, realiza una tarea en condiciones particulares del escenario de pruebas.

Las pruebas de rendimiento tienen que diseñarse para asegurar que el sistema pueda procesar su carga esperada. Esto normalmente implica planificar una serie de pruebas en las que el sistema se hace inaceptable. Estas pruebas implican estresar el sistema realizando demandas que están fuera de los límites del diseño del software (34).

El término “*pruebas de rendimiento*” se refiere al proceso de pruebas para determinar el rendimiento de un producto de software (36). Este hace referencia al estudio del comportamiento de un sistema (aplicación software, componente o una plataforma hardware) cuando se ve sometido a una carga (indicador sobre el sistema) que actúa de manera concurrente (acciones funcionales realizadas en un período de tiempo, procesos ejecutados de manera simultánea o usuarios validados en la aplicación en un momento determinado). Si el sistema no es sometido a carga su comportamiento no se puede conocer, pues las pruebas funcionales no incluyen niveles altos de concurrencia (37).

Existen varios tipos de pruebas de rendimientos de los cuales los más mencionados son:

- ▶ **Pruebas de carga:** Son la forma más simple de las pruebas de rendimiento y se llevan a cabo para comprender el comportamiento del sistema bajo una carga específica esperada. Estas pruebas le dan a los tiempos de respuesta de todas las transacciones importantes criterios de negocio. Intentarán validar que se alcanzan los objetivos de prestaciones a los que se verá sometido el sistema en un entorno productivo (37).

- ▶ **Pruebas de capacidad:** Buscan dar una estimación de hasta dónde se puede llegar cargando el sistema antes de que sea inutilizable. Su objetivo además de encontrar los límites de funcionamiento del sistema, es detectar el cuello de botella o elemento limitante para poder actuar en caso de ampliación de un servicio (37).
- ▶ **Pruebas de estrés:** Se utilizan normalmente para comprender los límites superiores de la capacidad dentro del sistema. Se aplican para determinar la robustez del sistema en términos de carga extrema y ayudan a los administradores de la aplicación para determinar si el sistema funcionará suficientemente si la corriente de carga va muy por encima del máximo esperado (37).

El empleo de este tipo de pruebas sirve para diferentes propósitos tales como demostrar que el sistema cumple los criterios de rendimiento, comparar dos sistemas para encontrar cuál de ellos funciona mejor o medir qué partes del sistema o de carga de trabajo provocan que el conjunto rinda mal. Para su diagnóstico se utilizan herramientas que pueden ser monitorizaciones que midan qué partes de un dispositivo o software contribuyen más al mal rendimiento o para establecer niveles que mantenga un tiempo de respuesta aceptable (38).

La herramienta a utilizar en este caso es JMeter, que es la propuesta en el procedimiento utilizado en esta investigación. La aplicación de escritorio Apache JMeter es un software de código abierto, diseñado para probar el comportamiento funcional y medir el rendimiento. Es una herramienta que puede ser utilizada para probar el rendimiento tanto de recursos dinámicos como estáticos (archivos, lenguajes dinámicos Web, bases de datos, consultas, servidores y más). Se puede utilizar para realizar un análisis gráfico de rendimiento o para probar el comportamiento del servidor bajo cargas concurrentes (39).

3.2.1. Entorno de prueba

El sistema se prepara para realizarle las pruebas de rendimiento para ello se utiliza una PC con las siguientes características:

Hardware: Intel Core i3-2120 a 3.30 GHzx4, 4 GB de RAM, sistema operativo Ubuntu 12.04, tipo de sistema 32-bit en una partición de 50 GB. **Software:** Gestor de Base de datos: PostgreSQL v9.1, Servidor de Aplicaciones: Apache 2.2, Máquina Virtual de Java: openjdk-7, Navegador Web: Mozilla Firefox. **LAN:** 100 Mbps de velocidad.

Se instala un MT que utiliza la versión 1.2.6 del ORM *Doctrine* y otro que funciona con las librerías de *Doctrine 2*. Por otra parte, se confeccionan los casos de pruebas de carga a

realizarle a las funcionalidades y que se emplean en el plan de prueba de la herramienta JMeter (Anexo 5. , Anexo 6 y Anexo 7).

3.2.2. Realizar pruebas de rendimiento

Como se menciona anteriormente, para la realización de este tipo de pruebas se selecciona la herramienta Apache JMeter específicamente para realizar pruebas de Carga y Estrés a la aplicación. El costo de la realización de estas pruebas suele ser en muchos casos elevado y por lo general la demanda de personal para la simulación de usuarios en diferentes escenarios es también elevada. La concurrencia y la demanda de los recursos del sistema normalmente son simuladas con un número de usuarios menor que la cantidad que realmente se necesitan para comprobar estos requerimientos.

Utilizando una herramienta para la automatización de este proceso se logra disminuir los costos en recursos y en tiempo destinados a su realización. Utilizar JMeter en aplicaciones web para la comprobación de los recursos del sistema, supone una mayor efectividad en el proceso y en la fiabilidad de los resultados. JMeter como herramienta de prueba dispone de varios componentes que facilitan la elaboración de los escenarios de prueba con la ventaja de simular para cada uno de esos escenarios miles de usuarios (40).

Con la previa elaboración de los Casos de Pruebas de Carga se procede a la realización de las pruebas con la herramienta JMeter utilizando para este trabajo la versión 2.3.1. Lo primero que se crea es un plan de pruebas sobre la base de una lista ordenada de peticiones HTTP. Utilizando el elemento **Grupo de Hilos** que se muestra en la **Figura** , se inicia la cabecera de la prueba en la que se define la cantidad de usuarios que se modelan definidos en los Casos de Pruebas.

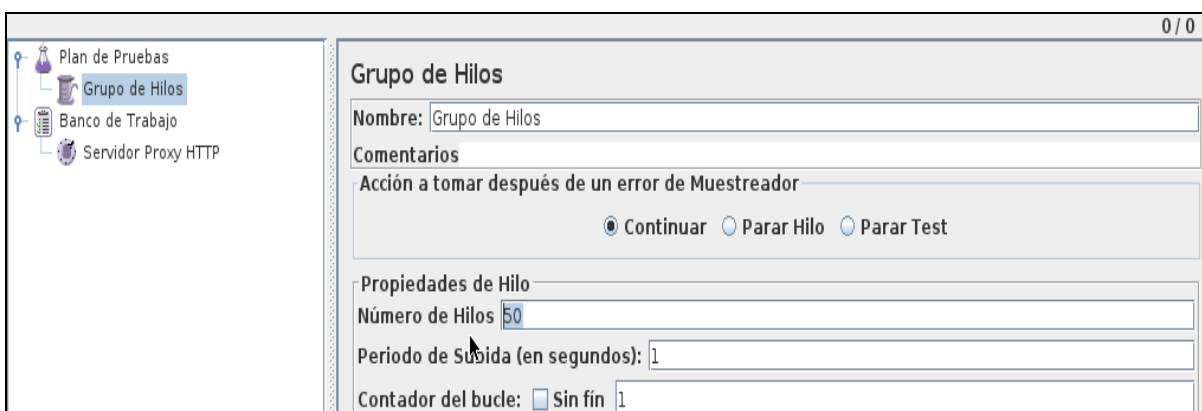


Figura 7. Elemento Grupo de Hilos.

Las propiedades de los hilos que se utilizan con este componente son:

- ✓ **Número de hilos:** número de usuarios a simular (100, 50 y 25 para cada caso de prueba utilizado).
- ✓ **Período de subida:** tiempo que se toma el JMeter en lanzar todos los hilos (especifica el período intermedio en segundos en los cuales va a ir iniciándose cada uno de los usuarios, para este caso se especifica un segundo).
- ✓ **Contador del bucle:** número de veces a realizar la prueba (para esta prueba se especifica que se simule una sola vez).

Para obtener una muestra de los elementos de interacción del sistema, donde se entran datos y se envían a la base de datos, se graban los escenarios a través del elemento **Servidor Proxy HTTP**, el cual se muestra en la siguiente figura:



Figura 8. Selección del Servidor Proxy.

Este elemento permite al JMeter grabar todos los pasos realizados en la aplicación de manera que se generan los escenarios de prueba automáticamente. Aquí se especifica un puerto para el servidor y la manera en la que se desean organizar los resultados de la grabación y se presiona el botón **Arrancar**.



Figura 9. Valores para el Servidor Proxy HTTP.

A continuación en el navegador, en las opciones de Internet, se le especifica la dirección y el puerto que se utiliza para la grabación de los escenarios. De esta manera se está indicando que el proxy que utilizará el navegador será el definido en el JMeter. Luego se ejecuta la aplicación y cada paso realizado en la Web será registrado en el JMeter de manera organizada.

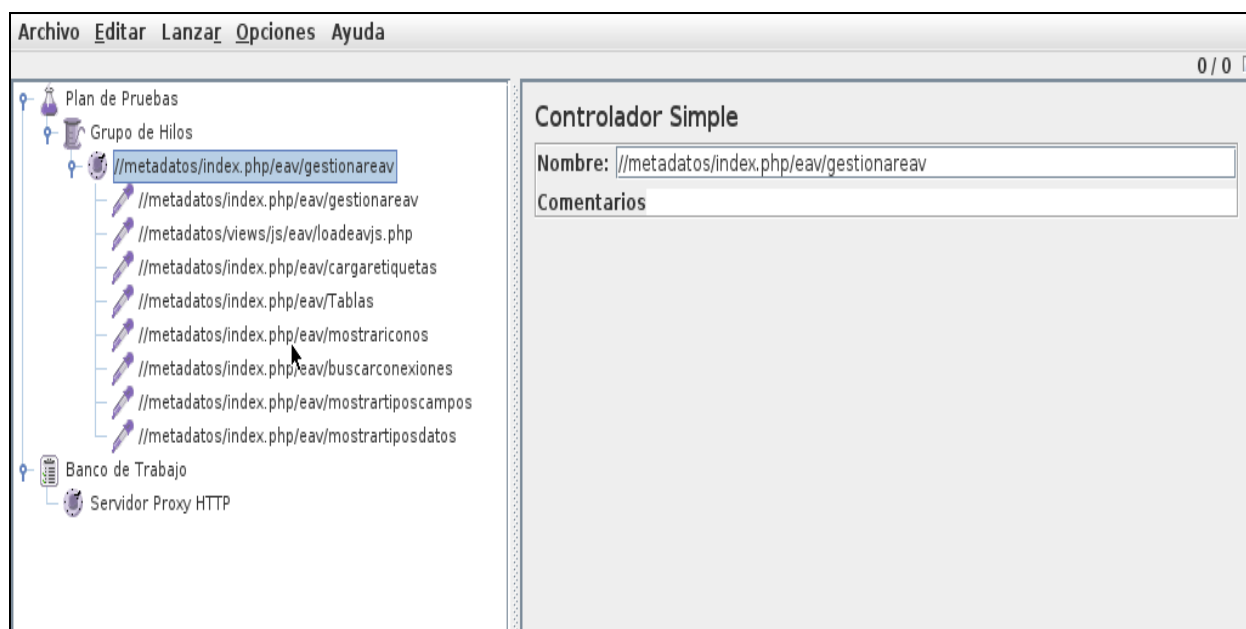


Figura 10. Grabación de Navegación de la Web.

Como se puede apreciar en la imagen anterior las peticiones http están grabadas dentro de controladores diferentes y cada una de ellas contiene toda su información; además los datos que han sido insertados en el sistema se muestran en los parámetros enviados con la petición.

Petición HTTP

Nombre: //metadatos/index.php/eav/modificarTablas

Comentarios

Servidor Web

Nombre de Servidor o IP: localhost

Petición HTTP

Protocolo: http Método: POST Content encoding: UTF-8

Path: //metadatos/index.php/eav/modificarTablas

Redirigir Automáticamente Seguir Redirecciones Utilizar KeepAlive Utilizar como M

Enviar Parámetros Con la Petición:

Nombre:	Valor
idtabla	9
nombre	Pruebas
fini	12/05/2014
concepto	on
idicono	
ffin	

Añadir Borrar

Enviar un archivo Con la Petición

Nombre de Archivo:

Nombre de Parámetro:

Tipo MIME:

Tareas Opcionales

Recuperar Todos los Recursos Empotrados de Archivos HTML Utilizar como M

Embedded URLs must match:

Figura 11. Ejemplo de petición HTTP de la grabación.

Por cada **Petición HTTP** se genera un **Gestor de Cabecera HTTP**, los cuales se eliminan porque interfieren en el éxito de las respuestas del servidor Web (40).

Una vez grabadas todas las peticiones http, es necesario agregar otros elementos en los cuales se especificarán los parámetros que se quieren comprobar en la prueba. Esto puede realizarse apoyándose en elementos como la **Aserción de Respuesta**.



Figura 12. Selección de Aserción de Respuestas.

En el caso de este elemento, se necesita conocer que la página a la cual se quiere acceder se ha cargado de manera satisfactoria por lo que se especifica el código 200 para una página cargada satisfactoriamente.

Figura 13. Datos de la Aserción de Respuesta.

Como las pruebas que se realizan son de carga y estrés se necesita un informe en el que se muestren resultados relacionados a los datos necesarios para comprobar el comportamiento de la aplicación. Para ello se utiliza el elemento **Informe Agregado**.

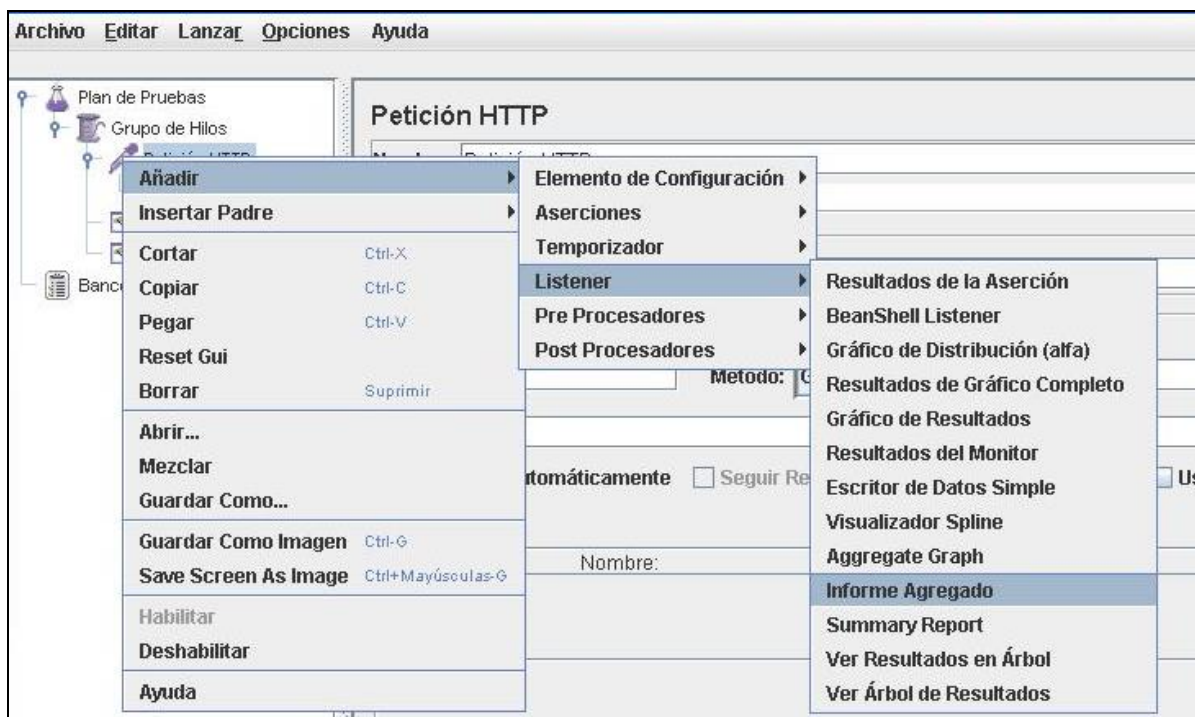


Figura 14. Selección de Informe Agregado.

En estos informes de cada una de las pruebas se recogen los siguientes indicadores:

- ▶ **# Muestras:** cantidad de muestras de peticiones utilizadas para la URL.
- ▶ **Media:** tiempo promedio en milisegundos transcurrido para un conjunto de resultados.
- ▶ **Mediana:** mediana aritmética (elemento de una serie ordenada de valores crecientes de manera que divide en dos partes iguales).
- ▶ **Min:** tiempo mínimo transcurrido para las muestras de peticiones de una determinada URL.
- ▶ **Max:** tiempo máximo transcurrido para las muestras de peticiones de una determinada URL.
- ▶ **% Error:** porcentaje de las peticiones con errores.

Por otra parte es recomendable utilizar el elemento **Ver Árbol de Resultados**, que muestra en detalle la información que ha sido cargada.

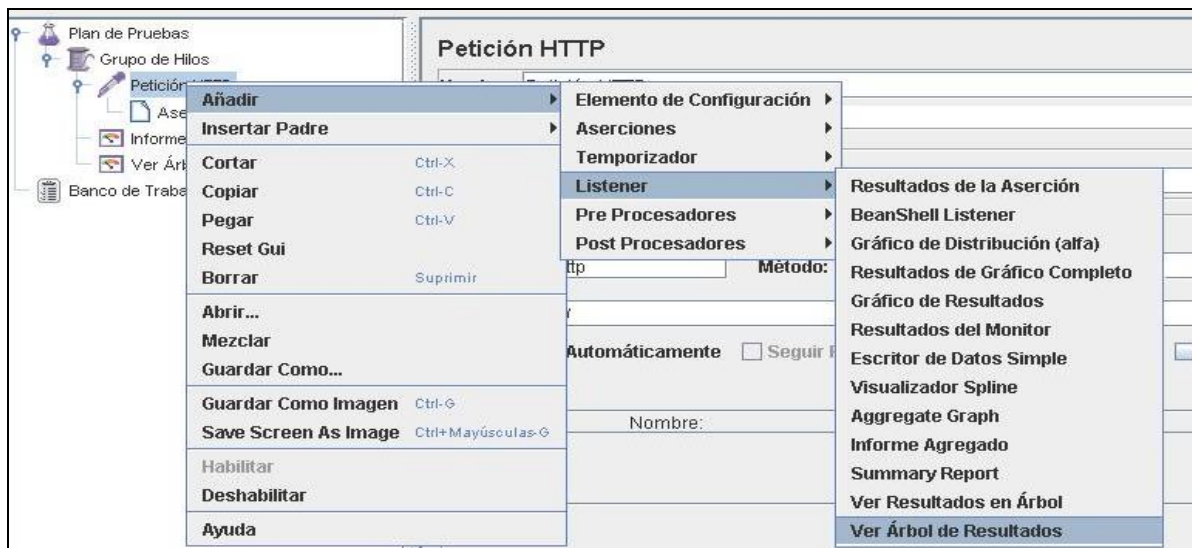


Figura 15. Selección de Ver Árbol de Resultados.

Con la grabación de las peticiones el **Plan de Pruebas** quedaría de la siguiente manera:

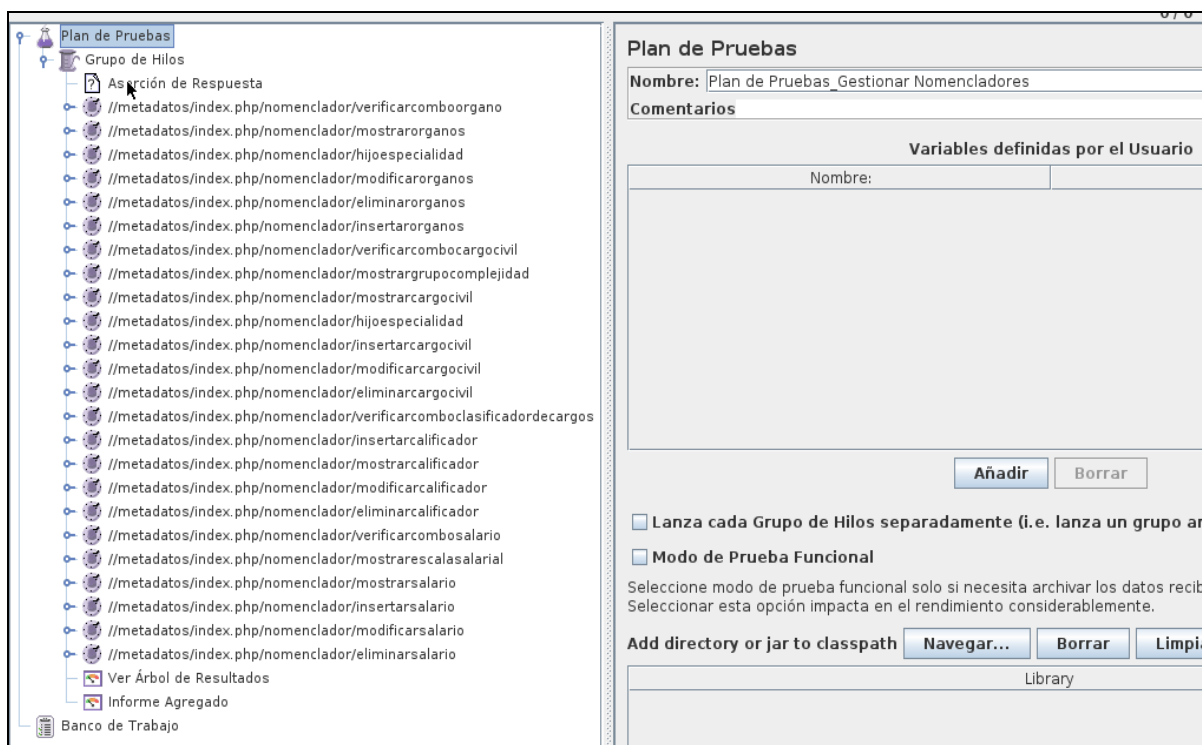


Figura 16. Confección de un Plan de Pruebas.

En la realización de las pruebas de carga y estrés al MT Sauxe se crearon 3 planes de prueba correspondientes a los 3 Casos de Prueba de Carga: “Gestionar Nomencladores” (**Anexo 2**), “Gestionar Estructuras” (**Anexo 3**) y “Definir Nivel Estructural” (**Anexo 4**).

El elemento **Informe Agregado** muestra los resultados correspondientes a cada Plan de Prueba, donde se evidencia el % de error para cada petición el no. de muestras y el rendimiento en segundos o milisegundos. A continuación un ejemplo de uno de ellos:

# Muestras	Media	Media...	Linea de ...	Mín	Máx	% Error
50	4150	4514	5642	166	15642	0.00%
250	2942	1299	6132	341	30560	0.00%
100	3833	1460	17422	386	24881	0.00%
100	2988	1829	6699	483	25597	0.00%
100	2509	1231	6244	105	25394	0.00%
150	3599	1425	15627	443	22822	0.00%
100	4763	2127	21332	818	28103	0.00%
50	3804	2136	6168	936	29318	0.00%
50	4011	1786	20438	799	33635	0.00%
150	2914	1191	2070	590	31806	0.00%
100	3540	1307	19944	499	28170	0.00%
50	3376	1220	20637	563	25301	0.00%
250	2778	1324	2555	111	26185	0.00%
200	3217	1309	12055	136	25213	0.00%
100	2658	1149	1697	236	26592	0.00%
50	3516	1244	15966	93	22339	0.00%
50	1175	1200	1735	110	2132	0.00%
150	2302	1055	3978	89	21646	0.00%
50	3366	1032	15921	81	24205	0.00%
50	2217	1169	1682	97	21898	0.00%
50	1387	999	1278	81	16663	0.00%
50	1748	902	1433	84	25145	0.00%
50	1760	1062	1424	87	19838	0.00%
50	2558	1084	12067	140	19700	0.00%
50	3008	1634	7692	366	21348	0.00%
50	2723	1008	11184	94	19815	0.00%
50	1245	663	1322	90	16511	0.00%
2500	2971	1283	6346	81	33635	0.00%

Figura 17. Informe Agregado.

Mientras que el componente “Ver Árbol de Resultados” permite visualizar cómo fueron manejadas cada una de las peticiones. Este elemento muestra el código de respuesta, el mensaje de respuesta, la cabecera de respuesta y otras propiedades. La siguiente imagen muestra un ejemplo de este componente.

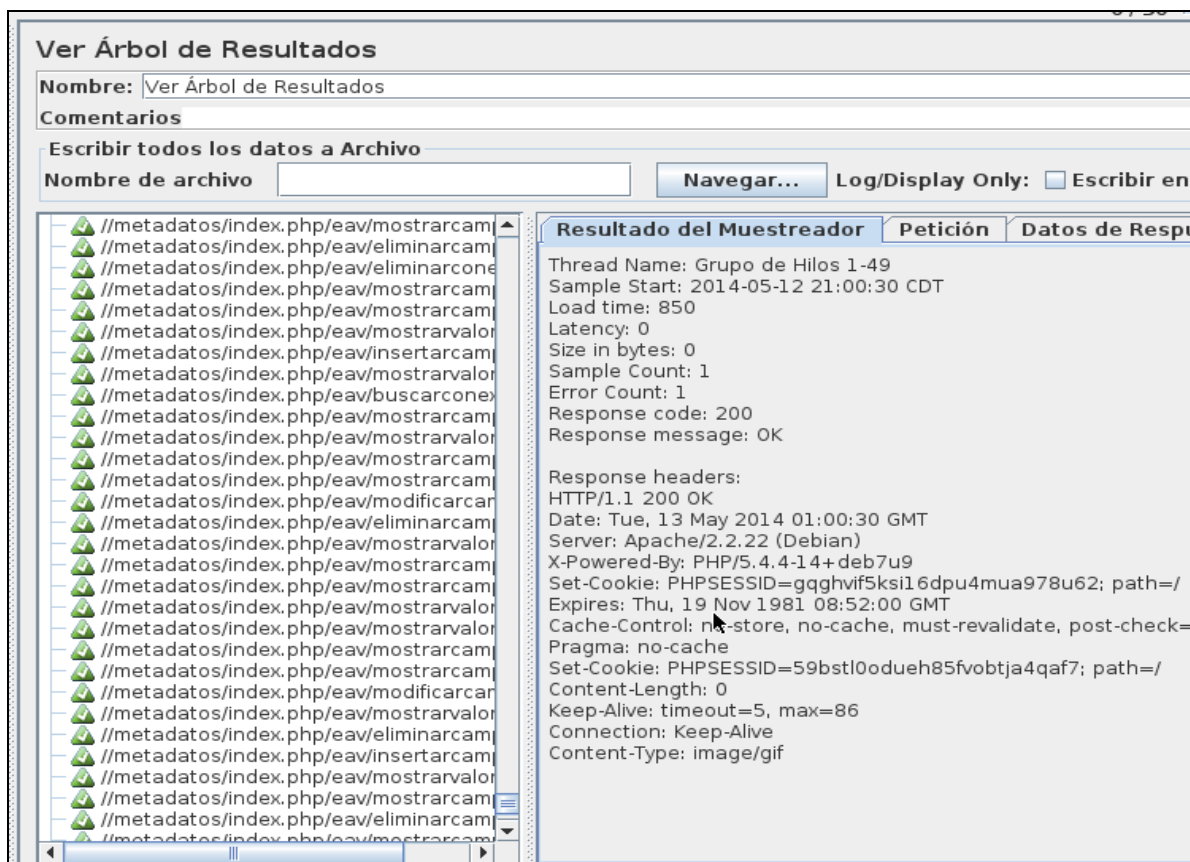


Figura 18. Ver Árbol de Resultados.

3.3. Resultado de las pruebas de rendimiento

Las pruebas realizadas consistieron en definir 3 estados de 100, 50 y 25 hilos para cada caso de prueba de carga, los cuales simulan 100, 50 y 25 accesos de usuarios respectivamente.

Prueba Nro. 1:

La primera prueba fue realizada el día 12/05/2014 a las 14:41 hs, se configuraron 100 hilos cada 1 seg. Los valores obtenidos por el componente **Informe Agregado**, correspondiente al caso de prueba “*Gestionar Estructuras*” con la versión *Doctrine 1*, se muestran en la siguiente tabla.

Tabla III. Valores correspondientes a la Prueba 1 para 100 hilos.

	# Muestra	Media	Mediana	Línea de 90%	Min	Max	% Error
TOTAL	4100	9615	1718	8805	92	136434	0.00%

Como se puede apreciar el tiempo promedio es 9,615 seg al realizar 4100 requerimientos al servidor. El tiempo total utilizado para los 100 hilos se puede calcular con la siguiente fórmula (41):

Tiempo Total = #Muestras * Media = 4100 * 9615 = 25891500 milisegundos.

El tiempo promedio total requerido por cada hilo, se puede calcular de la siguiente manera:
 $((\text{Tiempo Total}/1000)/60)/\text{cantidad de hilos} = ((25891500 /1000)/60)/100 = 4,31525$ minutos.

Prueba Nro. 2:

Esta prueba fue realizada el día 12/05/2014 a las 15:50 hs, se configuraron 50 hilos cada 1 seg. Los valores obtenidos por el componente **Informe Agregado**, correspondiente al caso de prueba “*Gestionar Nomencladores*” con la versión *Doctrine 1*, se muestran en la siguiente tabla.

Tabla IV. Valores correspondientes a la Prueba 2 para 50 hilos.

	# Muestra	Media	Mediana	Línea de 90%	Min	Max	% Error
TOTAL	2050	6097	1393	9924	90	72516	0.00%

El tiempo promedio es de 6,097 seg al realizar 2050 requerimientos al servidor. El tiempo total para los 50 hilos fue:

Tiempo Total = 2050 * 6097 = 12498850 milisegundos.

Tiempo promedio total = $((12498850/1000)/60)/50 = 4,1662$ minutos.

Prueba Nro. 3:

Esta prueba fue realizada el día 12/05/2014 a las 16:20 hs, se configuraron 25 hilos cada 1 seg. Los valores obtenidos por el componente **Informe Agregado**, correspondiente al caso de prueba “*Gestionar Nomencladores*” con la versión *Doctrine 1*, se muestran en la siguiente tabla.

Tabla V. Valores correspondientes a la Prueba 3 para 25 hilos.

	# Muestra	Media	Mediana	Línea de 90%	Min	Max	% Error
TOTAL	1025	3469	993	6097	115	39720	0.0%

El tiempo promedio es de 3,469 seg al realizar 1025 requerimientos al servidor. El tiempo total para los 25 hilos fue:

Tiempo Total = 1025 * 3469 = 3555725 milisegundos.

Tiempo promedio total = $((3555725/1000)/60)/25 = 2,3704$ minutos.

Prueba Nro. 4:

Esta prueba fue realizada el día 12/05/2014 a las 16:46 hs, se configuraron 100 hilos cada 1 seg. Los valores obtenidos por el componente **Informe Agregado**, correspondiente al caso de prueba “*Gestionar Nomencladores*” con la versión *Doctrine 2*, se muestran en la siguiente tabla.

Tabla VI. Valores correspondientes a la Prueba 4 para 100 hilos.

	# Muestra	Media	Mediana	Línea de 90%	Min	Max	% Error
TOTAL	4000	3389	3441	4309	121	10337	0,0%

El tiempo promedio es de 3,389 seg al realizar 4000 requerimientos al servidor. El tiempo total para los 100 hilos fue:

Tiempo Total = 4000 * 3389 = 13556000 milisegundos.

Tiempo promedio total = ((13556000/1000)/60)/100 = 2,2593 minutos.

Prueba Nro. 5:

Esta prueba fue realizada el día 12/05/2014 a las 17:14 hs, se configuraron 50 hilos cada 1 seg. Los valores obtenidos por el componente **Informe Agregado**, correspondiente al caso de prueba “Gestionar Nomencladores” con la versión *Doctrine 2*, se muestran en la siguiente tabla.

Tabla VII. Valores correspondientes a la Prueba 5 para 50 hilos.

	# Muestra	Media	Mediana	Línea de 90%	Min	Max	% Error
TOTAL	2000	1726	1787	2268	128	5982	0,0%

El tiempo promedio es de 1,726 seg al realizar 2000 requerimientos al servidor. El tiempo total para los 50 hilos fue:

Tiempo Total = 2000 * 1726 = 3452000 milisegundos.

Tiempo promedio total = ((3452000/1000)/60)/50 = 1,15066 minutos.

Prueba Nro. 6:

Esta prueba fue realizada el día 12/05/2014 a las 17:56 hs, se configuraron 25 hilos cada 1 seg. Los valores obtenidos por el componente **Informe Agregado**, correspondiente al caso de prueba “Gestionar Nomencladores” con la versión *Doctrine 2*, se muestran en la siguiente tabla.

Tabla VIII. Valores correspondientes a la Prueba 6 para 25 hilos.

	# Muestra	Media	Mediana	Línea de 90%	Min	Max	% Error
TOTAL	1000	910	910	1223	108	2313	0,0%

El tiempo promedio es de 910 ms al realizar 1000 requerimientos al servidor. El tiempo total para los 25 hilos fue:

Tiempo Total = 1000 * 910 = 910000 milisegundos.

Tiempo promedio total = ((910000/1000)/60)/25 = 0,6067 minutos.

Comparación general

Tabla IX. Resultados generales.

	100		50		25	
	Doctrine 1	Doctrine 2	Doctrine 1	Doctrine 2	Doctrine 1	Doctrine 2
# Muestras	4100	4000	2050	2000	1025	1000
Media	9615	3389	6097	1726	3469	910
Mediana	1718	3441	1393	1787	993	910
Línea de 90%	8805	4309	9924	2268	6097	1223
Min	92	121	90	128	115	108
Max	136434	10337	72516	5982	39720	2313
% Error	0.00	0.00	0.00	0.00	0.00	0.00
Tiempo Total (ms)	25891500	13556000	12498850	3452000	3555725	910000
Tiempo Promedio (min)	4,31525	2,2593	4,1662	1,15066	2,3704	0,6067

De la tabla anterior se puede concluir que el tiempo total utilizado para la cantidad de hilos definida, cambia notablemente en la versión 2.0.6 del ORM *Doctrine* respecto a la anterior. En el siguiente gráfico se visualizan los resultados obtenidos, que realizando una comparación se tiene como tiempo general de la prueba: 41946075 milisegundos para la versión 1.2.2 y 17918000 milisegundos para la 2.0.6. Estos resultados demuestran una disminución de 24028075 milisegundos en la ejecución del plan de pruebas en cuestión, lo que representa una reducción del 42,72% de este indicador.

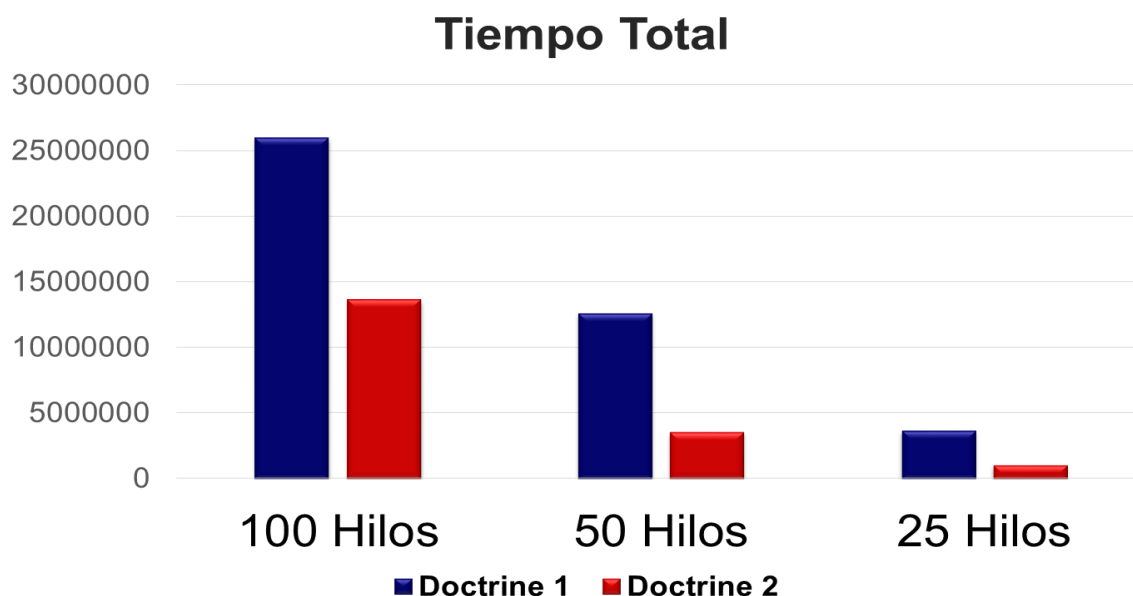


Figura 19. Gráfico de Resultados del Tiempo Total.

Por su parte, el tiempo promedio requerido para cada hilo se muestra en la **Figura 2**, donde se puede apreciar que hubo una mejora también de la versión 2 respecto a la 1. El tiempo

promedio general que se empleó para cada hilo fue de 10,85185 min para la versión 1.2.2 y 4,01666 min para la 2.0.6. Estos resultados arrojaron una reducción de 6,83519 min para la interacción con el sistema por las cantidades de usuarios simulados en cuanto a la utilización de ambas versiones, esto representa una disminución del 37,01% de este indicador.

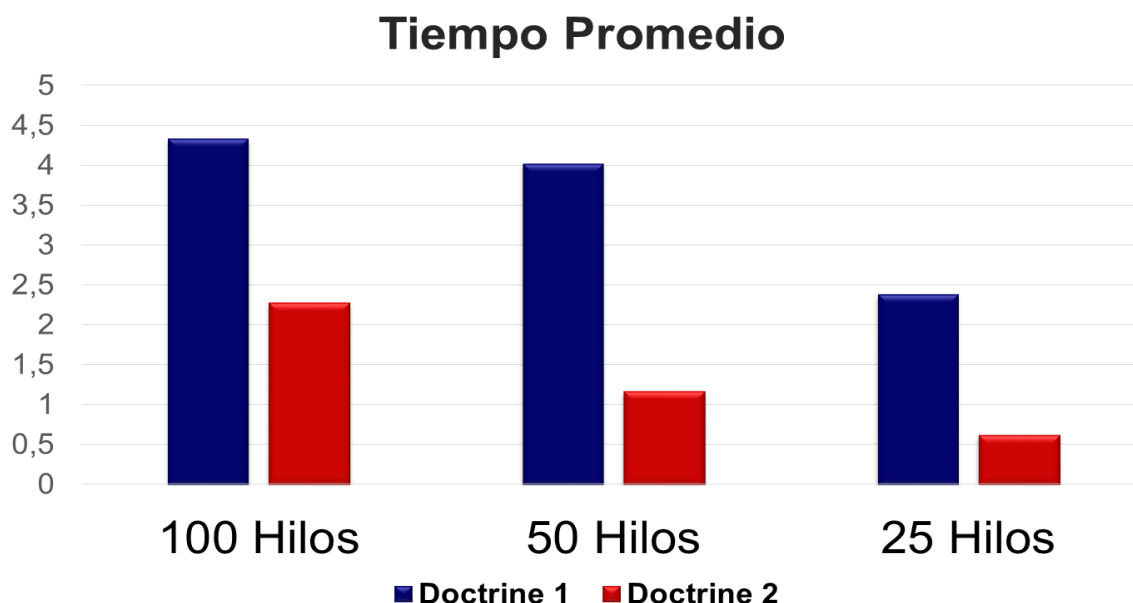


Figura 20. Gráfico de Resultados del Tiempo Promedio.

3.4. Resolver no conformidades

A partir de los resultados obtenidos con los elementos del JMeter: **Informe Agregado** y **Ver Árbol de Resultados**, se puede apreciar cuál petición no se ejecutó satisfactoriamente. En el caso del **Informe Agregado** este indica en la columna % Error si existió algún problema, mientras que **Ver Árbol de Resultados** indica el código de error y en color rojo la petición con errores.

Durante las pruebas realizadas, una acción a destacar es que no se reportaron incidencias de errores mientras la ejecución de las peticiones. Esto se garantiza debido a que durante la etapa de implementación se chequeó continuamente que las funcionalidades migradas tuvieran éxito. Por su parte, se apreciaron en las vistas de la aplicación la falta de información; debido a que los resultados de las consultas de la versión 2 de *Doctrine* no es igual a la anterior; para ello basta con cambiar la manera de solicitarle los datos al controlador en la vista.

3.5. Conclusiones Parciales

La explicación de los aspectos fundamentales de las pruebas de rendimiento que se le ejecutan a los sistemas, permitió comprender este proceso para aplicárselo al subsistema EyC.

Para la realización de las pruebas de rendimiento se utilizó la herramienta JMeter, cuyo análisis demostró las facilidades que brinda su empleo en este tipo de pruebas.

La evaluación realizada a la migración de la CAD del subsistema EyC, mediante las pruebas de rendimiento sobre las funcionalidades más críticas, arrojó como resultados la reducción del tiempo total y del tiempo promedio utilizado para una cantidad específica de usuarios en un 42,72% y un 37,01% respectivamente, lo que representa un aumento del rendimiento del subsistema EyC.

CONCLUSIONES GENERALES

El análisis realizado sobre algunos ORM utilizados para la persistencia de datos, demostró que estos exhiben características comunes y que en *Doctrine 2* se encuentran implementadas las principales potencialidades de los otros. Lo antes mencionado, demuestra que las buenas prácticas de los demás ORM se pueden reutilizar en la versión 2 de *Doctrine* al brindar este un alto rendimiento.

Con la actualización de la CAD del subsistema EyC, en la cual se estableció una nueva estructura de carpetas y se redefinió tanto la CAD como la Capa de Negocio, se eliminó el uso de malas prácticas y se logró una mejor organización en el código y en los ficheros generados del mapeo.

Las pruebas realizadas por la comisión del proyecto sobre cada una de las funcionalidades del subsistema EyC, permitieron demostrar que cumplen con las expectativas de los clientes.

La evaluación realizada a la migración de la CAD del subsistema EyC, mediante las pruebas de rendimiento sobre las funcionalidades más críticas, arrojó como resultados la reducción del tiempo total y del tiempo promedio utilizado para una cantidad específica de usuarios en un 42,72% y un 37,01% respectivamente, lo que representa un aumento del rendimiento del subsistema EyC. Se demuestra así que se le ha dado cumplimiento al objetivo general planteado y a cada uno de los objetivos específicos que se definieron.

RECOMENDACIONES

Se recomienda migrar las funcionalidades “*Recuperaciones*” y “*Subordinaciones*” del subsistema EyC del MT Sauxe a *Doctrine 2*. Además, implementar las funcionalidades de las librerías ZendExt utilizando el DQL de *Doctrine 2* para garantizar la interacción íntegra del MT Sauxe con el ORM *Doctrine 2*.

BIBLIOGRAFÍA

1. **Minetto, E.L.**, *Frameworks para Desarrollo en PHP*. 2007, Livraria Martin Fontes: Sao Paulo.
2. **Baryolo, O.G.**, SOLUCIÓN INFORMÁTICA DE AUTORIZACIÓN EN ENTORNOS MULTIENTIDAD Y MULTISISTEMA. 2001, Universidad de las Ciencias Informáticas: La Habana.
3. **Bugarin, J.Luis**. Slideshare. [En línea] 07 de Julio de 2013. [Citado el: 28 de Noviembre de 2013.] <http://www.slideshare.net/jlbugarin/persistencia-de-datoshibernatearquitecturasdesoftware#btnNext>.
4. **Pupo, Y.C.** *Libro de Ayuda del Marco de Trabajo Sauxe*. Libro de Ayuda del Marco de Trabajo Sauxe. La Habana : s.n., 2010.
5. **Madeja**. Marco de Desarrollo de la Junta de Andalucía. *Comparación de las tecnologías de acceso a datos*. [En línea] [Citado el: 29 de Noviembre de 2013.] <http://www.juntadeandalucia.es/servicios/madeja/contenido/recurso/180>.
6. **Osorio, I.V., Estrada, L.C., y Camejo, R.R.B.** *Procedimiento para actualización de la Capa de Acceso a Datos del Marco de Trabajo Sauxe de Doctrine 1.2.2 a Doctrine 2.0*. La Habana : s.n., 2013.
7. **Arquitectura, D.d.**, Pruebas de rendimiento sobre Cedrux. 2012: Universidad de las Ciencias Informáticas.
8. **McCafferty, B.** Best Practices with ASP.NET, 1.2nd Ed. 2008 [En línea] [Citado el: 11 de Noviembre de 2013]; Disponible en: <http://www.codeproject.com/Articles/13390/NHibernate-Best-Practices-with-ASP-NET-1-2nd-Ed#SUMMARY>.
9. **León, R.A.H y González, S.C.** *El proceso de investigación científica*. Ciudad de La Habana : Editorial Universitaria, 2011. ISBN 978-959-16-1307-3.
10. **Sampieri, R. H; Collado, C. F. y Lucio, P. B.** *Metodología de la Investigación*. Ciudad de México: Editorial Mc Graw Hill, 2006. Cuarta Edición. ISBN 970-10-5753-8.
11. **Leroy**. MyCyberAcademy. *Un poco sobre ORM y Frameworks*. [En línea] 14 de Agosto de 2013. [Citado el: 29 de Noviembre de 2013.] <http://mycyberacademy.com/articulo/Un-Poco-Sobre-ORM-y-Frameworks>.
12. **Rubén**. CODECRITICON. *Frameworks de persistencia*. [En línea] 18 de Noviembre de 2011. [Citado el: 23 de Noviembre de 2013.] <http://codecriticon.com/diferencias-frameworks-persistencia-i/>.

13. **Cadavid, A.N.** *Sistema de investigación de la Universidad Icesi*. 2008 [En línea] [Citado el: 25 de Noviembre de 2013]; Disponible en: <http://www.icesi.edu.co/investigaciones/proyecto.do?id=38>.
14. **Bisbé, R.L.** *¿Qué es un ORM y por qué nos interesa?*. [En línea] 26 de Noviembre de 2011. [Citado el: 06 de Diciembre de 2013.] <http://robertoluis.wordpress.com/2011/12/13/que-es-un-orm-y-por-que-nos-interesa/>.
15. **Community, H.** 2012 [En línea] [Citado el: 26 de Noviembre de 2013]; Disponible en: <http://docs.jboss.org/hibernate/orm/4.2/devguide/en-US/html/ch02.html#d5e618>.
16. **Propel, S.** *Propel, Smart, easy object persistence*. 2013 [En línea] [Citado el: 3 de Diciembre de 2013]; Disponible en: <http://propelorm.org/>.
17. **Vondrick, C.** *Symfony. ¿Cómo utilizar Propel?* [En línea] 2011; Disponible en: http://symfony.com/legacy/doc/cookbook/1_2/es/propel_13.
18. **Hasheado.** *Doctrine vs Propel*. 2010 [En línea] [Citado el: 09 de Febrero de 2014]; Disponible en: <http://www.hasheado.com/doctrine-vs-propel.html>.
19. **García, L.** *Scribd, Django ORM*. 2011 [En línea] 03 de Marzo de 2011 [Citado el: 27 de Noviembre de 2013]; Disponible en: <http://es.scribd.com/doc/63901225/Django-ORM>.
20. **Adrian Holovaty, J.K.-M.** *The Django Book*. 2012 [En línea] [Citado el: 4 de Diciembre de 2013]; Disponible en: <http://www.djangobook.com/en/2.0/chapter02.html>.
21. **SubSonic.** *SubSonic*. 2012 [En línea] [Citado el: 27 de Noviembre de 2013]; Disponible en: <http://subsonicproject.com/>.
22. **Subsonic.** *Subsonic*. 2012; [En línea] [Citado el: 27 de Noviembre de 2013]; Disponible en: <http://subsonicproject.com/docs/Comparisons/#Performance>.
23. **Documentation, D.O.** *Doctrine 2 ORM Documentation*. 2012 [En línea] 04 de Diciembre de 2012 [Citado 04 de Diciembre de 2013]; Documentación del ORM Doctrine 2; Disponible en: <http://docs.doctrine-project.org/en/latest/>.
24. **Ben Collins-Sussman, B.W.F.y.C.M.P.,** *Control de Versiones con Subversion*. 2013.
25. **Tecsisá.** *Buenas Prácticas de Gestión de Versiones con Subversion*. 2010 [En línea] 06 de Mayo de 2010 [Citado el: 09 de Febrero de 2014]; Disponible en: <http://blogs.tecsisa.com/articulos-tecnicos/buenas-practicas-de-gestion-de-versiones-con-subversion/>.
26. **RapidSVN.** *RapidSVN*. 2012 [En línea] [Citado el: 14 de Enero de 2014]; Disponible en: <http://www.rapidsvn.org/>.
27. **Martínez, R.,** *Portal de Postgres en Español*. 2013.
28. **NetBeans,** *NetBeans quick start*. 2013.

29. **Foundation, T.A.S.** *The Apache Software Foundation*. 2012 [Citado 17 de Enero de 2014]; Disponible en: <http://www.apache.org>
30. **Inc, Z.T.** *Manual Zend Framework Español*. 2005 01 de Diciembre de 2012 [Citado 2014 30 de Enero de 2014]; Disponible en: <http://manual.zfdes.com/es/introduction.overview.html>.
31. **México, U.N.A.d.** *Universidad Nacional Autónoma de México*. 2012 28 de enero 2013 [Citado 2014 31 de Enero de 2014]; Administración para la toma de decisiones; Disponible en: <http://fcasua.contad.unam.mx/apuntes/interiores/docs/2005/administracion/5/1553.pdf>.
32. **Community, H.** 2012 [En línea] 08 de Febrero de 2012 [Citado el: 09 de Febrero de 2014]; Disponible en: <http://docs.jboss.org/hibernate/orm/3.6/reference/es-ES/html/best-practices.html>.
33. **Php.** *Manual de PHP* [En línea] [Citado el: 09 de Febrero de 2014]; Disponible en: <http://www.php.net/manual/es/index.php>.
34. **Sommerville, I.** *Ingeniería del software*. Pearson Educación, 2005.
35. **Globe Testing.** *Pruebas de rendimiento*. [En línea] [Citado el: 28 de Abril de 2014]; Disponible en: <http://www.globetesting.com/pruebas-de-rendimiento/>.
36. **QA Técnico.** *Definiciones ISTQB*. [En línea] [Citado el: 28 de Abril de 2014]; Disponible en: <http://qatecnico.blogspot.com/p/definiciones-istqb.html>.
37. **QA Técnico.** *Pruebas de rendimiento: Tipos y objetivos*. [En línea] 03 de Marzo de 2012 [Citado el: 28 de Abril de 2014]; Disponible en: <http://qatecnico.blogspot.com/2012/03/pruebas-de-rendimiento-tipos-y.html>.
38. **Madeja.** Marco de Desarrollo de la Junta de Andalucía. *Buenas prácticas en el diseño de pruebas de rendimiento*. [En línea] [Citado el: 04 de Mayo de 2014.] <http://www.juntadeandalucia.es/servicios/madeja/contenido/libro-pautas/75>.
39. **JMeter, A.** *Apache JMeter*. [En línea] 2013 [Citado el: 12 de Abril de 2013]; Disponible en: <http://jmeter.apache.org/>.
40. **Almenares, L.S.** *Cómo Realizar Pruebas de Carga y Estrés en JMeter*. La Habana: s.n., 2008.
41. **Díaz, J., Banchoff, C. M. T., Rodríguez, A. S., y Soria, V.** *Usando Jmeter para pruebas de rendimiento*. En XIV Congreso Argentino de Ciencias de la Computación. La Plata : s.n., 2008.

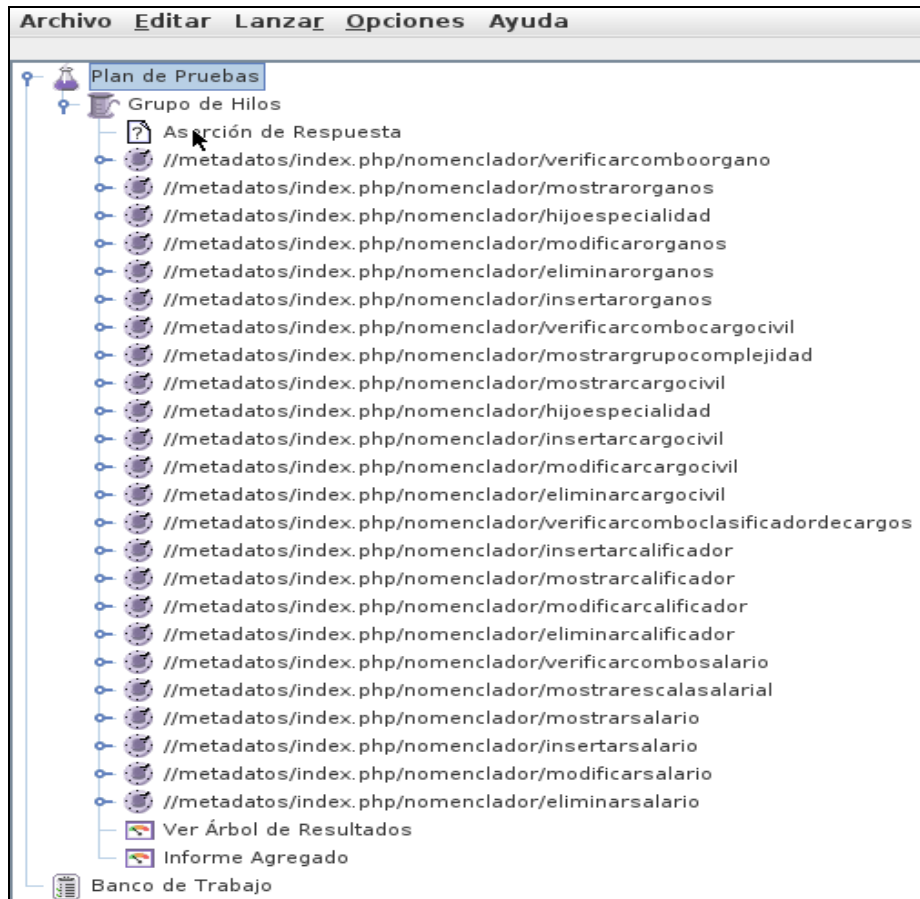
ANEXOS

Anexo 1. Resultado de pruebas al sistema Xedro-ERP

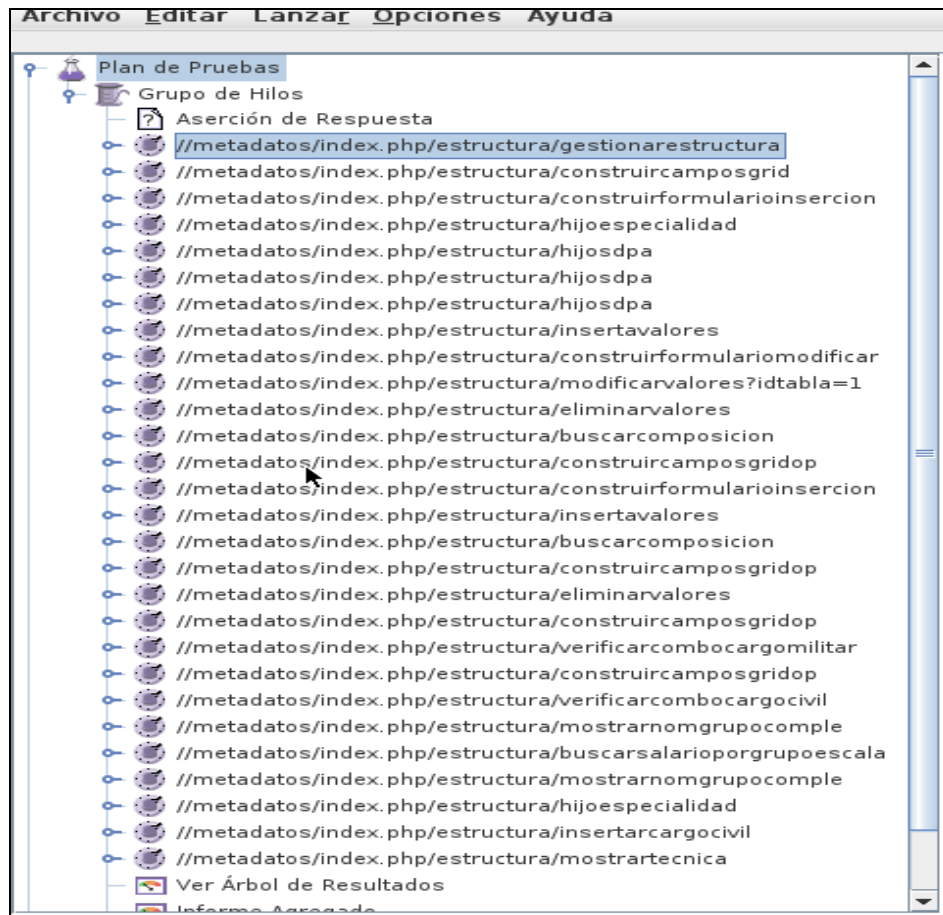
Descripción de la Funcionalidad	Tiempo de ejecución (seg)
Creando DatPlan e insertando.	0.353950977325
Insertando los ejercicios del plan adicionado.	0.105323076248
Insertando la 1 versión del plan.	0.0349161624908
Buscando etapas de la configuración (servicio).	0.334882974625
Insertando las etapas de ese plan de acuerdo a la configuración elegida. Insertando los modelos de una etapa.	20.6633300781
Buscando modelos de la configuración.	0.15479683876
Insertando modelos al plan.	20.6963710785

Bloque de código	Tiempo de respuesta
Con Doctrine	3.975062847
Con función de BD	0.178375959396

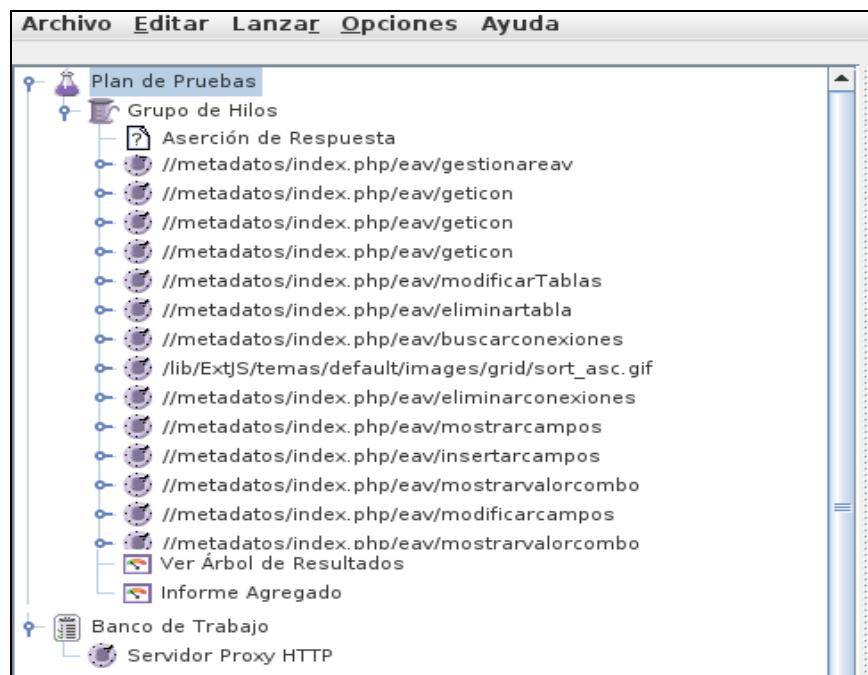
Anexo 2. Confección del Plan de Pruebas para el Caso de Prueba de Carga “Gestionar Nomenclador”.



Anexo 3. Confección del Plan de Pruebas para el Caso de Prueba de Carga “Gestionar Estructuras”.



Anexo 4. Confección del Plan de Pruebas para el Caso de Prueba de Carga “Definir Nivel Estructural”.



Anexo 5. Casos de Prueba de Carga para “Gestionar Nomenclador”

ID del escenario	Escenario de la sección	Carga de Trabajo	Descripción
EC1-Acceder a la interfaz “Gestionar nomencladores”.	EC1.1- Acceder a la interfaz “Gestionar nomencladores”. http://localhost:5800/portal/index.php/portal/	100, 50 y 25	Es la interfaz para gestionar los nomencladores del módulo Estructura y Composición.
EC2-Nomenclador Órgano.	EC2.1- Mostrar órganos. Para ello ejecutar clic en el indicador “Órgano”.	100, 50 y 25	Esta interfaz muestra información relacionada con los órganos registrados en la aplicación.
	EC2.2- Buscar órganos. Para ello llenar el campo Denominación: y ejecutar clic en el botón “Buscar”	100, 50 y 25	Esta interfaz muestra información relacionada con los órganos registrados en la aplicación.
	EC2.3- Insertar un órgano. Para ello ejecutar clic en el botón Adicionar llenar los campos y ejecutar clic en	100, 50 y 25	Esta interfaz muestra los campos necesarios para insertar un órgano en la base de datos desde la aplicación.

	Aceptar.		
	EC2.4- Modificar un órgano. Para ello seleccionar un elemento ejecutar clic en el botón Modificar, cambiar los campos y ejecutar clic en Aceptar.	100, 50 y 25	Esta interfaz muestra los campos necesarios para modificar un órgano en la base de datos desde la aplicación.
	EC2.5 Eliminar órgano. Para ello seleccionar un elemento y ejecutar el botón Eliminar y después Aceptar.	100, 50 y 25	Se muestra una notificación sobre el órgano que se desea eliminar.
EC3- Nomenclador Cargo civil.	EC3.1- Mostrar cargos civiles. Para ello ejecutar clic en el indicador "Cargo civil".	100, 50 y 25	Esta interfaz muestra información relacionada con los cargos civiles registrados en la aplicación.
	EC3.2- Buscar cargo civil. Para ello llenar el campo Denominación: y ejecutar clic en el botón "Buscar"	100, 50 y 25	Esta interfaz muestra información relacionada con los cargos civiles registrados en la aplicación.
	EC3.3- Insertar un cargo civil. Para ello ejecutar clic en el botón Adicionar llenar los campos y ejecutar clic en Aceptar.	100, 50 y 25	Esta interfaz muestra los campos necesarios para insertar un cargo civil en la base de datos desde la aplicación.
	EC3.4- Modificar un cargo civil. Para ello seleccionar un elemento ejecutar clic en el botón Modificar, cambiar los campos y ejecutar clic en Aceptar.	100, 50 y 25	Esta interfaz muestra los campos necesarios para modificar un cargo civil en la base de datos desde la aplicación.
	EC3.5 Eliminar cargo civil Para ello seleccionar una fila y ejecutar el botón Eliminar y	100, 50 y 25	Se muestra una notificación sobre el cargo civil que se desea eliminar.

	después Aceptar.		
EC4- Nomenclador Calificador de cargo.	EC4.1- Mostrar calificadores de cargos. Para ello ejecutar clic en el indicador “Calificador de cargos”.	100, 50 y 25	Esta interfaz muestra información relacionada con los calificadores de cargos registrados en la aplicación.
	EC4.2- Buscar calificador de cargo. Para ello llenar el campo Denominación: y ejecutar clic en el botón “Buscar”	100, 50 y 25	Esta interfaz muestra información relacionada con los calificadores de cargos registrados en la aplicación.
	EC4.3- Insertar un cargo civil. Para ello ejecutar clic en el botón Adicionar llenar los campos y ejecutar clic en Aceptar.	100, 50 y 25	Esta interfaz muestra los campos necesarios para insertar un calificador de cargo en la base de datos desde la aplicación.
	EC4.4- Modificar un calificador de cargo. Para ello seleccionar un elemento ejecutar clic en el botón Modificar, cambiar los campos y ejecutar clic en Aceptar.	100, 50 y 25	Esta interfaz muestra los campos necesarios para modificar un calificador de cargo en la base de datos desde la aplicación.
	EC4.5 Eliminar calificador de cargo Para ello seleccionar una fila y ejecutar el botón Eliminar y después Aceptar.	100, 50 y 25	Se muestra una notificación sobre el calificador de cargo que se desea eliminar.
EC5- Nomenclador Escala salarial.	EC5.1- Mostrar escalas salariales. Para ello ejecutar clic en el indicador “Calificador de cargos”.	100, 50 y 25	Esta interfaz muestra información relacionada con las escalas salariales registradas en la aplicación.

	EC5.2- Buscar escala salarial. Para ello llenar el campo Denominación: y ejecutar clic en el botón “Buscar”	100, 50 y 25	Esta interfaz muestra información relacionada con las escalas salariales registradas en la aplicación.
	EC5.3- Insertar una escala salarial. Para ello ejecutar clic en el botón Adicionar llenar los campos y ejecutar clic en Aceptar.	100, 50 y 25	Esta interfaz muestra los campos necesarios para insertar una escala salarial en la base de datos desde la aplicación.
	EC5.4- Modificar una escala salarial. Para ello seleccionar un elemento ejecutar clic en el botón Modificar, cambiar los campos y ejecutar clic en Aceptar.	100, 50 y 25	Esta interfaz muestra los campos necesarios para modificar una escala salarial en la base de datos desde la aplicación.
	EC5.5 Eliminar escala salarial Para ello seleccionar una fila y ejecutar el botón Eliminar y después Aceptar.	100, 50 y 25	Se muestra una notificación sobre la escala salarial que se desea eliminar.

Anexo 6. Casos de Prueba de Carga para “Gestionar Nivel Estructural”

ID del escenario	Escenario de la sección	Carga de Trabajo	Descripción
EC1 – Acceder a la interfaz “Definir nivel estructural”.	EC1.1- Acceder a la interfaz “Definir nivel estructural”. http://localhost:5800/portal/index.php/portal/	100, 50 y 25	Es la interfaz para gestionar las estructuras y sus relaciones así como los campos estructurales que presentan.
EC2- Estructuras	EC2.1- Insertar estructura. Para ello ejecutar clic en el botón “Adicionar”.	100, 50 y 25	Esta interfaz muestra los campos necesarios para insertar una estructura.
	EC2.2- Modificar estructura. Para ello seleccionar un elemento ejecutar clic en el	100, 50 y 25	Esta interfaz muestra los campos necesarios para modificar una estructura en la base de datos

	botón Modificar, cambiar los campos y ejecutar clic en Aceptar.		desde la aplicación.
	EC2.3 Eliminar estructura. Para ello seleccionar una fila y ejecutar el botón Eliminar y después Aceptar. Nota: Sólo se eliminarán las estructuras que se tengan permisos.	100, 50 y 25	Se muestra una notificación sobre la estructura que se desea eliminar.
EC3-Relaciones	EC3.1- Insertar una relación. Para ello seleccionar una estructura y ejecutar el botón "Adicionar".	100, 50 y 25	Se muestra una notificación y se muestra la relación adicionada.
	EC3.2- Eliminar una relación Para ello seleccionar una relación y ejecutar el botón "Eliminar" y clic en Aceptar.	100, 50 y 25	Se muestra una notificación y se elimina la relación seleccionada.
EC4- Gestionar campos	EC4.1- Mostrar campos. Para ello seleccionar una estructura y ejecutar el indicador "Gestionar campos".	100, 50 y 25	Esta interfaz muestra la información de los campos que contiene una determinada estructura.
	EC4.2- Insertar un campo estructural. Para ello ejecutar clic en el botón Adicionar llenar los campos y ejecutar clic en Aceptar.	100, 50 y 25	Esta interfaz muestra los campos necesarios para insertar un campo estructural a la estructura seleccionada en la base de datos desde la aplicación.
	EC4.3- Modificar un campo estructural. Para ello seleccionar un elemento ejecutar clic en el botón Modificar, cambiar los campos y ejecutar clic en	100, 50 y 25	Esta interfaz muestra los campos necesarios para modificar un campo estructural en la base de datos desde la aplicación.

	Aceptar.		
	EC4.4 Eliminar campo estructural. Para ello seleccionar una fila y ejecutar el botón Eliminar y Aceptar.	100, 50 y 25	Se muestra una notificación sobre el campo que se desea eliminar.

Anexo 7. Casos de Prueba de Carga para “Gestionar Estructuras”.

ID del escenario	Escenario de la sección	Carga de Trabajo	Descripción
EC1- Acceder a la interfaz “Gestionar estructuras”.	EC1.1- Acceder a la interfaz “Gestionar estructuras”. http://localhost:5800/portal/index.php/portal/	100, 50 y 25	Es la interfaz que muestra las estructuras su composición y garantiza su gestión.
EC2- Estructuras	EC2.1- Insertar un tipo de estructura. Para ello seleccionar una estructura y ejecutar clic derecho seleccionar en la opción Adicionar: El tipo de estructura	100, 50 y 25	Esta interfaz muestra los campos necesarios para insertar un tipo de estructura.
	EC2.2- Modificar estructura. Para ello seleccionar un elemento ejecutar clic derecho seleccionar la opción Modificar, cambiar los campos y ejecutar clic en Aceptar.	100, 50 y 25	Esta interfaz muestra los campos necesarios para modificar una estructura en la base de datos desde la aplicación.
	EC2.3 Eliminar estructura. Para ello seleccionar un elemento y ejecutar el botón Eliminar y Aceptar.	100, 50 y 25	Se muestra una alerta sobre el elemento que se desea eliminar.
EC3- Composición	EC3.1- Mostrar composición de una estructura. Para ello seleccionar una estructura.	100, 50 y 25	Esta interfaz muestra en forma de árbol la composición de la estructura seleccionada.

	<p>EC3.2- Insertar tipo de composición.</p> <p>Para ello seleccionar una composición ejecutar clic derecho y en la opción Adicionar seleccionar un tipo de composición.</p>	100, 50 y 25	Esta interfaz muestra los campos necesarios para insertar una composición a la estructura seleccionada en la base de datos desde la aplicación.
	<p>EC3.3- Eliminar composición.</p> <p>Para ello seleccionar una composición ejecutar clic derecho y seleccionar la opción Eliminar.</p>	100, 50 y 25	Se muestra una alerta sobre el elemento que se desea eliminar.
EC4- Adicionar cargo.	<p>EC4.1-Adicionar cargo militar.</p> <p>Para ello seleccionar una composición ejecutar clic derecho y seleccionar en la opción Adicionar cargo: "Militar".</p>	100, 50 y 25	Esta interfaz muestra los campos necesarios para adicionar a la composición seleccionada un cargo militar en la base de datos desde la aplicación.
	<p>EC4.2- Adicionar cargo civil.</p> <p>Para ello seleccionar una composición ejecutar clic derecho y seleccionar en la opción Adicionar cargo: "Civil".</p>	100, 50 y 25	Esta interfaz muestra los campos necesarios para adicionar a la composición seleccionada un cargo civil en la base de datos desde la aplicación.
EC5- Gestionar medio.	<p>EC 5.1- Mostrar los medios.</p> <p>Para ello seleccionar una composición ejecutar clic derecho y seleccionar la opción "Gestionar medios".</p>	100, 50 y 25	Esta interfaz muestra la información de los medios pertenecientes a la composición seleccionada.
	<p>EC5.2- Confeccionar técnica.</p> <p>Para ello ejecutar el botón "Adicionar" de la interfaz "Gestionar técnicas".</p>	100, 50 y 25	Esta interfaz muestra los campos necesarios para adicionarle una técnica a la composición seleccionada en la base de datos desde la aplicación.

	EC5.3- Modificar técnica. Para ello seleccionar una técnica y ejecutar el botón "Modificar".	100, 50 y 25	Esta interfaz muestra los campos necesarios para modificar una técnica a la composición seleccionada en la base de datos desde la aplicación.
	EC5.4 – Eliminar técnica. Para ello seleccionar una técnica y ejecutar el botón Eliminar y después Aceptar.	100, 50 y 25	Se muestra una notificación y se elimina la técnica seleccionada.