Universidad de las Ciencias Informáticas

Facultad 15

Migración del subsistema Gestión de Cuadros y Personal de Apoyo del proyecto Sistema de Gestión Fiscal a la versión 1.3 del framework Symfony.

Trabajo de Diploma para optar por el título de Ingeniero Informático.

Autor: Frank Orlando Menéndez Guerra

Tutor: Ing. Alain Hernández López

2010

Frase

Lo que puedes hacer, o has soñado que podrías hacer, debes comenzarlo. La osadía lleva en sí, genio, poder y magia.

GOETHE

Declaratoria de Autoría

Declaro que soy el único autor de este trabajo y autorizo al <nombre del área> de la Universidad de las Ciencias Informáticas a hacer uso de su del mismo en su beneficio.

Para que así conste firmo el presente a los __ días del mes de _____ del año _____.

Frank Orlando Menéndez Guerra	Ing. Alain Hernández López
_____	_____
Firma del Autor	**Firma del Tutor**

Datos de Contacto

Datos del Autor

Nombre: Frank Orlando Menéndez Guerra

Correo electrónico: fomenendez@estudiantes.uci.cu

Datos del Tutor

Nombre: Ing. Alain Hernández López

Correo electrónico: ahernandezlop@uci.cu

Agradecimientos

A mi Querida Madre Ana Guerra.

A mi Segunda Madre Mercedes.

A mis Hermanos: Aimee, Osain, Anoy, Moremis y Juan Miguel.

A mis Tías: Tía China, Diamil, Margarita, Marilyn, Nancy, Onelia y Grisel.

A mi Tío Arturo.

A mis Amigos: Henry, Luis, Yunetsy, Sujaila, Jessica, Mileydis , Raidel y Raidel, Osmany, Ileana, Ernesto y a la Familia de mis Amigos que también lo son para mí.

A mi gran Familia.

A mis Vecinos.

A los que estuvieron conmigo lejos y fueron por seis meses y por siempre mis amigos: Yudelmis, Elien(La flaca), Michael, Juana, Juan Carlos, Guiogui Ernesto, Doña Dorca, Katiuska y Yunior.

A todos los que faltaron por mencionar pero que están presentes.

Y a mí.

Dedicatoria

A mi Mamá y a Mima.

Resumen

En el Proyecto Sistema de Gestión Fiscal (en lo adelante SGF) se realizó la migración del *framework Symfony* sobre el cual se desarrolla la aplicación (del mismo nombre) con el objetivo principal de acortar el tiempo de desarrollo, incorporar nuevas funcionalidades y eliminar vulnerabilidades de seguridad. Para ello se decidió migrar uno de los subsistemas: ¨Gestión de Cuadros y Personal de Apoyo¨ (en lo adelante GCPA) que había sido el primero en ser desarrollado sobre la versión 1.0 del *framework* para garantizar la correcta integración de todos los Subsistemas de la aplicación sobre en la versión 1.3.

En este documento se recoge un estudio de las herramientas y pasos que se utilizaron para realizar la migración de GCPA. También contiene los resultados de las pruebas realizadas al subsistema.

Palabras Claves

Migración, *Symfony*, *framework*, subsistema.

Índice

Introducción

El Hombre a lo largo de la historia ha buscado la manera de desarrollar sus herramientas para realizar su trabajo de manera más cómoda, económico y funcional. Esta búsqueda lo ha llevado, a lo largo de la historia, a ir evolucionando y perfeccionando las herramientas con las que cuenta hasta llegar a las potentes computadoras de la actualidad, las que juegan un papel fundamental en el desarrollo del mundo actual. Cuba no se encuentra apartada de este fenómeno. Hace varios años el país ha destinado cuantiosas sumas de dinero y esfuerzo para realizar un proceso de informatización con el objetivo de usar las tecnologías con fines sociales.

La Fiscalía General de la República de Cuba (en lo adelante FGRC) se ha sumado a este proceso, para ello ha acordado con la UCI (Universidad de las Ciencias Informáticas) informatizar los principales procesos que intervienen en la Fiscalía. De este modo se crea el proyecto Sistema de Gestión Fiscal (SGF) que tiene como objetivo fundamental: informatizar todos los procesos que se desarrollan en la fiscalía general. Este proyecto se encuentra desarrollando una aplicación web que cuenta con ocho subsistemas:

- Procesos Penales (PP).
- Gestión de Cuadros y Personal de Apoyo (GCPA).
- Control de la Legalidad en Establecimientos Penitenciarios (CLEP).
- Dirección General de Control (DGC).
- Herramientas Comunes a todas las Áreas (HCTA).
- Protección a los Derecho Ciudadanos (PDC).
- Relaciones Internacionales (RI).
- Verificación Fiscal (VF).

Gestión de Cuadros y Personas de Apoyo es uno de los subsistemas principales desarrollados por el proyecto SGF teniendo como objetivos fundamentales:

- Garantizar la gestión de los cuadros (fiscales) y el personal de apoyo (estudiantes que cursan la carrera de derecho y que están haciendo las prácticas en la fiscalía). Implica la gestión de datos de

cuadros, gestión de la capacitación de cuadros, movimiento de cuadros dentro de la fiscalía. Además de la gestión de los datos del personal de apoyo que están vinculado a la fiscalía.

- Garantizar los niveles de seguridad y acceso tanto de los cuadros como del personal de apoyo a este subsistemas de la aplicación.

GCPA fue el primer subsistema desarrollado por el equipo de desarrollo del proyecto. Fue desarrollado sobre el *framework* para el desarrollo de aplicaciones web en PHP: "*Symfony*", en la versión 1.0. Todos los restantes subsistemas están siendo desarrollados en versiones superiores del *framework* debido al incremento de las nuevas funcionalidades que disminuyen el tiempo de desarrollo de las aplicaciones, el soporte técnico actualizado presente en las nuevas versiones (necesario en la etapa de desarrollo de la aplicación) y de las mejoras de seguridad para asegurar la integridad de los datos ante ataques de tipo *XSS* (*Cross-Site Scripting*) y *CSRF* (*Cross-Site Request Forgery* o falsificación de petición en sitios cruzados). El objetivo de ambos ataques es el mismo: aprovechar ciertas vulnerabilidades de la aplicación, el agresor puede colocar en la página cualquier código, el cual posteriormente puede servir para la ejecución de operaciones no planificadas por el creador del sitio web, por ejemplo, capturar archivos cookies sin que el usuario se percate. Estos cambios enriquecen la aplicación con la eliminación de vulnerabilidades antes no encontradas haciéndolas más seguras y fiables.

Además de ser necesario integrar GCPA a los demás subsistemas de la aplicación SGF.

Ante la anterior situación entonces se plantea la lo siguiente:

Problema de Investigación:

¿Cómo eliminar del subsistema GCPA (del proyecto SGF) los errores de seguridad, además de añadir nuevas funcionalidades para acortar el tiempo de desarrollo, adquirir mayor tiempo de soporte técnico actualizado y garantizar su integración con los demás subsistemas?

Objeto de Estudio

La migración de sistemas.

Campo de acción:

La migración de versiones en el *framework Symfony*.

Objetivo General:

- Migrar el subsistema GCPA del proyecto SGF a la versión 1.3 de *framework* Symfony.

Hipótesis

Con la migración del subsistema GCPA del proyecto SGF a la versión 1.3 de *Symfony* se garantizará la integración de todos los subsistemas con nuevos beneficios de seguridad, nuevas funcionalidades y mayor tiempo de soporte.

Tareas de Investigación:

- Construcción de un marco teórico sobre los diferentes métodos y herramientas de migración existentes en el mundo.
- Selección del método y herramienta de migración adecuada.
- Migración del subsistema GCPA a la versión 1.3 de *Symfony*.
- Realización de pruebas correspondientes al proceso de migración.

Resultados:

- Guía de migración para productos desarrollados con el *framework Symfony*.
- Código fuente del subsistema GCPA del proyecto SGF migrado a la versión 1.3 del *framework Symfony*.

Métodos Teóricos:

Historio-lógico: Se realiza un análisis de cómo han evolucionado las diferentes aplicaciones web realizadas en *Symfony* y que fueron actualizadas.

Métodos Empíricos:

La entrevista: Se realiza la entrevista a diferentes proyectos que han utilizado la migración para aportar información a la guía de migración.

Capítulo 1. Fundamentos Teóricos.

1.1. Introducción

Se denomina migración a todo desplazamiento de población que se produce desde un lugar de origen a otro destino y lleva consigo un cambio de la residencia habitual en el caso de las personas o del hábitat en el caso de las especies animales migratorias, pero también existe el término migración en el mundo de la informática, siendo en este caso el proceso consistente en hacer que los datos y las aplicaciones existentes funcionen en una computadora, software o sistema operativo distinto. En la actualidad este término se ha utilizado en muchas ocasiones entorno al auge de las migraciones a software libre y al hecho de que instituciones públicas a nivel mundial han realizado este proceso de migración exitosamente.

El presente capítulo contiene un estudio de los tipos de migración, del *framework Symfony* con un análisis de sus versiones y de las herramientas necesarias para realizar la migración de proyectos creados en él, así como la necesidad y propuesta de migración para el proyecto GCPA. Todo esto proporciona mayor entendimiento de este trabajo.

1.2. Estado del Arte

1.2.1. Migración

La **RAE** (*Real Academia Española*) define migración:

1. f. emigración.

2. f. Acción y efecto de pasar de un país a otro para establecerse en él. Se usa hablando de las migraciones históricas que hicieron las razas o los pueblos enteros.

3. f. Viaje periódico de las aves, peces u otros animales migratorios.

4. f. Desplazamiento geográfico de individuos o grupos, generalmente por causas económicas o sociales.

En el ámbito informático la migración se refiere al traslado de una aplicación de un ordenador a otro en condiciones de compatibilidad. Es también elevar una versión de un producto software a otra de más alto nivel, o bien el movimiento de una arquitectura a otra.

La migración de versiones: consiste en actualizar el software a una versión superior que reemplaza una versión instalada de un producto por una versión más reciente del mismo producto. Este proceso de actualización a una versión superior no altera normalmente los datos ni las preferencias del usuario existentes al reemplazar el software existente por la nueva versión.

Normalmente las personas instalan uno o varios programas y los dejan funcionando durante un período prolongado creyendo que son perfectos. Con el paso del tiempo se detectan fallos y problemas de seguridad en los mismos. Otras veces sencillamente se busca mejorar los programas con nuevas herramientas y opciones. Por ambos motivos surgen nuevas versiones de los mismos programas y llega el momento de actualizarse.

Al igual que con los sistemas operativos, es necesario tener, en la medida de lo posible, actualizados los programas, como mínimo los que se utilizan con mayor frecuencia. De ésta manera, siempre se podrá contar con las nuevas herramientas y utilidades que brindan estos programas.

Mantener los sistemas actualizados es muy importante debido al creciente desarrollo alcanzado por las redes telemáticas e Internet, los cuales han propiciado que los usuarios compartan rápidamente sus conocimientos retroalimentándose unos de otros continuamente.

En la actualidad, las empresas más importantes de software actualizan sus productos por Internet, mediante una búsqueda automática o manual (accediendo a la Web del producto), dependiendo del fabricante y siempre bajo el consentimiento del usuario. Por lo general, estas actualizaciones corrigen los errores que se detectan una vez lanzada la aplicación, en ningún caso se implantan mejoras de funcionalidad, como herramientas nuevas, etc. (Muñoz Tamayo, y otros, 2009)

Los proveedores de software le dan diferentes nombres y calificativos a las actualizaciones según las distintas versiones, los errores que son solucionados o las nuevas funcionalidades que presentan. Estos nombres pueden ser: actualizaciones importantes, de seguridad, de alta prioridad o recomendadas. (Muñoz Tamayo, y otros, 2009)

A continuación se listan las ventajas y desventajas de la migración de versiones.

Ventajas:

- Es posible que ocurra la incorporación de nuevas y últimas tecnologías.
- Eliminación de errores presentes en las versiones anteriores.
- Desarrollo de nuevas funcionalidades que enriquecen el manejo de las herramientas.
- El soporte técnico actualizado sobre la versión actualizada es brindado por los proveedores de la herramienta.

Desventajas:

- Algo muy importante que se tiene que tener en cuenta a la hora de actualizar es el precio que se tiene que pagar para mantener actualizado un sistema.
- No todas las versiones nuevas que aparecen son compatibles al 100% con sus versiones anteriores.
- En muchos casos se estrecha más el tiempo entre una versión y otra.

1.2.2. Symfony

Symfony es un *framework* que simplifica el desarrollo de una aplicación web mediante la automatización de algunos de los patrones utilizados para resolver las tareas comunes. Está basado en el clásico patrón de diseño web conocido como arquitectura MVC (Modelo-Vista-Controlador). Proporciona varias herramientas y clases encaminadas a reducir el tiempo de desarrollo de una aplicación web compleja. También, automatiza las tareas más comunes, permitiendo al desarrollador dedicarse por completo a los aspectos específicos de cada aplicación. (Potencier, y otros, 2009)

Symfony está desarrollado completamente con PHP 5. Ha sido probado en numerosos proyectos reales y se utiliza en sitios web de comercio electrónico de primer nivel. *Symfony* es compatible con la mayoría de gestores de bases de datos como *MySQL*, *PostgreSQL*, *Oracle* y *SQL Server* de *Microsoft*. Se puede ejecutar tanto en plataformas **nix* (Unix, Linux, etc.) como en plataformas Windows. (Potencier, y otros, 2009)

Desde su primera versión, publicada en octubre de 2005 bajo la licencia de software libre por su creador Fabien Potencier, muchos desarrolladores de todo el mundo se descargaron e instalaron el *framework*, comenzaron a leer la documentación y construyeron sus primeras aplicaciones con *Symfony*, aumentando poco a poco la popularidad del mismo. (Potencier, y otros, 2009)

En ese momento, los *frameworks* para el desarrollo de aplicaciones web estaban en pleno florecimiento, y era muy necesario disponer de un completo *framework* realizado con PHP. *Symfony* proporcionaba una solución irresistible a esa carencia, debido a la calidad de su código fuente y a la gran cantidad de documentación disponible, dos ventajas muy importantes sobre otros *frameworks* disponibles. Los colaboradores aparecieron en seguida proponiendo parches y mejoras, detectando los errores de la documentación y realizando otras tareas muy importantes. (Potencier, y otros, 2009)

El repositorio público de código fuente y el sistema de notificación de errores y mejoras mediante tickets permite varias formas de contribuir al proyecto y todos los voluntarios son bienvenidos (Potencier, y otros, 2009). Estos beneficios que brindan los creadores del *framework* atraen a los programadores con deseos que ven en Symfony una poderosa herramienta para la creación de aplicaciones web.

A través del foro de *Symfony*, las listas de correo y el IRC (canal de mensajería instantánea) se ofrecen otras alternativas válidas para el soporte del *framework*. De manera que los desarrolladores de *Symfony* alrededor del mundo están conectados en una comunidad que muestra muchas ganas de colaboración entre sus integrantes. Dentro de la misma se encuentran también los creadores del *framework* lo que nutre en gran medida las discusiones que se realizan en dicha comunidad.

Hoy día internet cuenta con cientos sitios creados con *Symfony*. Desde grandes redes sociales como: *Delicious*, *Dailymotion* o *Yahoo! Answers* hasta medianos y pequeños sitio. De forma que millones de usuarios utilizan aplicaciones construidos con *Symfony*. (Potencier, y otros, 2009)

1.2.3. Análisis de Versiones de Symfony

Es necesario realizar un análisis de las versiones existentes hasta la actualidad para poder guiar el proceso de migración.

El equipo de *Symfony* ha corregido los errores y problemas de seguridad del *framework*. El promedio de versiones muestra que se publica una versión de corrección de errores al mes (desde el año 2005 hasta

2009). Estas versiones no contienen nuevas características o funciones. Por lo tanto, siempre son compatibles con versiones anteriores, fáciles y seguras para actualizarse a versiones estables.

1.2.3.1. Versión 1.0

En octubre de 2005 se publica la primera versión de *Symfony 1.0.19*. Los primeros desarrolladores en probarlo detectaron rápidamente muchos errores y nuevas posibilidades de mejoras.

Comenzaron entonces los desarrolladores a brindar soporte a esta versión dándole soluciones a los diferentes errores que encontraban hasta llegar a la primera versión estable 1.0.20 que contiene los siguientes errores corregidos y mejoras adicionadas:

- Se soluciona el problema con la tarea para la construcción del esquema de la base de datos de la aplicación: **propel:schema-to-yml** y la conversión de llaves foráneas compuestas.
- Se soluciona el error con la posible corrupción de la caché cuando la carga de la aplicación es alta.
- Se corrige el comportamiento cuando se solicita una URL interna mediante el método: *sfRequest::getUri()* utilizado en Microsoft IIS.
- Agregados mecanismos de escape al contenido de las variables que se muestran en las excepciones del entorno de desarrollo.
- Mejorado el rendimiento del método de búsqueda: *search_in()* de la clase: *sfFinder* cuando no se buscan enlaces simbólicos.
- Incluida una mejora en el rendimiento.
- También se centraron en corregir un gran error de seguridad concerniente a los ataques *XSS* (*Cross Site Scripting*). Estos ataques son realizados por usuarios que aprovechan la confianza que un sitio web deposita en ellos y de esta forma colocan un código en un campo de texto no verificado con el objetivo de introducir cualquier contenido. Los datos colocados de este modo pueden servir posteriormente para extraer información confidencial del usuario, ejecutar determinadas operaciones con atributos de usuario registrado y acciones similares.

El problema de seguridad respecto a los ataques XSS tardó un año en solucionarse sin que nadie advirtiera sobre el grave peligro que suponía. Esto propició la necesaria modificación de la política de

seguridad del proyecto *Symfony* y el paso a la nueva versión 1.2. Desde ese momento cualquier usuario o programador que encuentre un problema de seguridad en *Symfony*, puede notificarlo a una dirección de correo electrónico, aportando toda la información posible sobre el error encontrado y cualquier archivo o parche que pueda ayudar a solucionarlo. Los mensajes recibidos en esa dirección pasan directamente a los programadores de *Symfony* que lo investigan para actuar lo antes posible.

Los desarrolladores de *Symfony* muestran gran interés por resolver los problemas que encuentran en el *framework* mediante la interacción con la comunidad asociada a ellos. Esto es algo positivo en la medida que permite la incorporación de iniciativas y soluciones que pueden provenir incluso de otros colaboradores. Permitiendo el avance del *framework* y su popularidad debido a su abierta política de colaboración.

Esta versión a pesar de cumplir muchas necesidades a la hora de crear una aplicación web no es menos cierto que una vez comenzada la interacción con los desarrolladores demostró que quedaban algunos elementos que precisaban estar presentes, como es el caso de los errores de seguridad. Demostró esta versión la capacidad de integración de *Symfony* con otras tecnologías y *frameworks*.

1.2.3.2. Versión 1.1

Esta versión tiene como principal novedad la eliminación del problema de seguridad presente en la rama de la versión 1.0.

Otras de las novedades presentes en esta versión son las siguientes:

- Una nueva arquitectura formada por varias clases relacionadas entre sí pero completamente desacopladas, lo que se conoce con el nombre de plataforma *Symfony*. Estas clases no tienen ninguna dependencia y la única condición para utilizarlas es registrar el cargador automático de clases de *Symfony*, es decir, llamar a las clases. La gran ventaja de la plataforma *Symfony* es que puedes utilizar algunas de las utilidades que incluye sin la obligación de utilizar toda la arquitectura MVC completa.
- Se integra un ORM (*Object-Relational Mapping*) alternativo a *Propel* que se llama *Doctrine*. *Propel* se incluye en el *framework* en forma de *plugin*, lo que facilita el cambio al otro ORM, para lo cual sólo es necesario instalar el *plugin*.

- Entre los errores corregidos estaba el correspondiente a cuando se activaba la barra de depuración para revisar consultas *SQL* muy grandes.
- Otro error importante estaba relacionado con la internacionalización la cual es uno de los puntos fuertes de *Symfony*, por lo que sólo se mejoró aquello que era incómodo para trabajar. Por ello se añadió utilidades para facilitar la gestión de las cadenas de texto que han de traducir en las interfaces de las aplicaciones.
- Se eliminan las vulnerabilidades de los ataques *XSS* en los formularios. Mientras que para los ataques *CSRF* (*Cross-Site Request Forgeries*) es creado un *plugin* llamado *pdCSRFPlugin*.

Los ataques *CSRF* se basan en el uso de elementos, existentes o introducidos legalmente, de la página para fines maliciosos. Este tipo de ataque en lugar de explotar la confianza del usuario explota la confianza que hace el sitio web a sus usuarios. Es uno de los ataques más populares y se basa en el uso del marcador HTML *<img>* que sirve para la visualización de gráficos. En vez del marcador con la URL del archivo gráfico el agresor pone un código JavaScript que es ejecutado en el navegador de la víctima. Esto permite llevar a cabo una infinidad de operaciones en el marco de la sesión del usuario quien frecuentemente no es consciente de que precisamente está siendo atacado.

El hecho de que esta versión cuente con un *plugin* para hacer frente a los ataques que comprometen la seguridad de una aplicación y que no estén implementados dentro del *framework* provoca inseguridad en los desarrolladores lo que constituye una desventaja para *Symfony 1.1*.

La nueva arquitectura presente en esta versión es el eslabón fundamental de *Symfony*. Con las clases desacopladas y relacionadas entre ellas permiten la integración de otras tecnologías y *framework*, así como la creación de un framework propio utilizando los componentes básicos de la plataforma *Symfony*.

1.2.3.3. Versión 1.2

Para esta rama las mejoras añadidas fueron importantes:

- *Propel* fue actualizado a la versión 1.3, por lo que significa una mejora apreciable en el rendimiento y *Doctrine* (versión 1.0.12) fue incluido por defecto en el *framework*, facilitando su uso en vez de *Propel*. (SensioLab, 2009)

- Los formularios de *Symfony 1.2* incluyen mejoras, añadidos y correcciones respecto a los de *Symfony 1.1*. (Eguiluz, 2009)
- Mejora de la cache duplicando su rendimiento. (Eguiluz, 2009)
- El sistema de enrutamiento es ahora diez veces más rápido. (Eguiluz, 2009)
- La versión estable 1.2.6 corrige un problema de seguridad de la parte de administración lo que impide la desconexión automática de los campos ocultos olvidados en los formularios. (SensioLab, 2009)

Esta es una versión de paso lo que significa que solo fueron mejorados algunos elementos para dar paso a la siguiente versión del *framework*.

1.2.3.4. Versión 1.3 y 1.4

Tanto *Symfony 1.3* como *Symfony 1.4* se han publicado aproximadamente al mismo tiempo a finales de 2009. Las dos versiones tienen exactamente las mismas características. La única diferencia entre ellas es cómo realizan la retro-compatibilidad con las versiones anteriores de *Symfony*. (Eguiluz, 2009)

Symfony 1.3 es la versión que se debe utilizar para actualizar un proyecto de una versión anterior de *Symfony* (1.0, 1.1, o 1.2). Esta versión dispone de una capa de retro-compatibilidad que hace que todas las características que se han declarado obsoletas sigan estando disponibles. Por tanto, la actualización suele ser más sencilla, fácil y segura.

En la versión 1.4 se han eliminado todas las características obsoletas de la versión 1.3, incluyendo la capa de retro-compatibilidad, por lo que se hace muy difícil realizar una migración a esta versión. La versión 1.4 es mucho más limpia y más rápida. Es recomendable crear nuevos proyecto con esta versión antes que migrar hacia ella. (Eguiluz, 2009)

Entre las nuevas características que tienen estas versiones están:

- *Symfony 1.3* es compatible con *PHP 5.2.4* o superior.
- La protección frente a los ataques XSS y CSRF ahora es obligatoria para las aplicaciones porque ya están activadas por defecto.

- *Symfony 1.3/1.4* incluyen un nuevo *framework* para enviar correos llamado *Mailer que* hace más sencillo realizar esta función.
- La validación de los campos de los formularios es más sencilla.
- La barra de depuración web puede ser configurable por los desarrolladores según sus necesidades.
- Se elimina el filtro común, que es el encargado de añadir automáticamente las hojas de estilos y los archivos JavaScript en las páginas. Por lo que es obligatorio incluir a manualmente los archivos CSS y JavaScript. Esto mejora el funcionamiento de la aplicación haciéndola más flexible y sencilla permitiendo tener un control preciso sobre dónde se añaden.

Con todas estas características *Symfony* llega al fin de la rama de versiones 1.x brindando las funcionalidades para crear aplicaciones seguras, flexibles y de manera rápida.

1.2.3.5. Versión 2.0

Para esta versión tiene las novedades como:

- Un contenedor de inyección de dependencias, un *framework* para plantillas y una nueva parte del controlador.
- En aplicaciones web reales, la diferencia de rendimiento entre *Symfony 2.0* y *1.2* no es muy superior, pero el núcleo de *Symfony 2.0* sí es mucho más rápido que el de *Symfony 1.2*.
- El elemento fundamental de la parte del controlador del nuevo *Symfony 2.0* será el “Gestor de Peticiones” o *Request Handler*.
- En *Symfony 2.0* es imposible actualizar los proyectos realizados con la versión anterior, a menos que se reescriban completamente, debido a su notable diferencia con las versiones anteriores.
- Las ideas clave de *Symfony 2.0* son: rapidez y flexibilidad.
- Presenta una nueva estructura de directorios para los proyectos de *Symfony*.
- Los archivos de configuración se pueden escribir en PHP, XML (*Lenguaje de marcas extensible* del inglés *Extensible Markup Language*), YAML (*"Otro lenguaje de marcado* más" del inglés *"Yet*

Another Markup Language") o INI (proviene de *"Windows Initialization file"*, es decir, *archivo de inicialización de Windows.*). Internamente *Symfony 2.0* utiliza XML, para permitir el autocompletado en los IDE y para poder validar los archivos.

- Desaparecen los *plugins* y se sustituyen por un elemento mucho más potente llamado *bundle*.
- La barra de depuración web ahora se muestra en la parte inferior de la página.
- *Symfony 2.0* será mucho más fácil de aprender que *Symfony* 1.
- El lanzamiento está previsto para finales de 2010.

El proyecto *Symfony* obligará a utilizar PHP 5.3 o superior, de forma que pueda aprovechar todas las nuevas características de PHP 5.3.

1.2.4. Método de migración

Existen métodos para cada una de las de migraciones de los proyectos desarrollados. Estos métodos están descritos en el sitio oficial del *framework*. En cada de ellas hay una serie de pasos que guían el proceso migratorio.

No existen pasos que actualicen una aplicación de la versión 1.0 a la 1.3 directamente, para ello es obligatorio migrar primero de la versión 1.0 a la versión 1.1, después de la versión 1.1 a la 1.2 y finalmente de 1.2 a 1.3.

Los métodos de migración son realizados a través de tareas que están incorporadas al *framework* y que son ejecutadas por mediación de una terminal o consola. Una vez ejecutada la tarea de migración *Symfony* se encarga de reemplaza algunos archivos como por ejemplo los controladores frontales. Muchos de sus elementos y clases son modificados o eliminados en la actualización mientras que otros deben ser modificados por el desarrollador.

Sin embargo, basado en la experiencia del proceso de migración de aplicaciones web del *framework Symfony*, los métodos y pasos de migración que se encuentran en el sitio oficial de *Symfony* están dirigidos a usuarios con bastante práctica en la construcción de estas aplicaciones.

1.3. Herramientas para la migración

1.3.1. CLI

La Interfaz de Línea de Comandos (por su acrónimo en inglés de *Command Line Interface*) es un método que permite a las personas dar instrucciones a algún programa informático por medio de una línea de texto simple. (Potencier, 2008)

Se puede emplear interactivamente escribiendo instrucciones en alguna especie de entrada de texto o pueden utilizarse de una forma mucho más automatizada, leyendo comandos desde un archivo de scripts. (Potencier, 2008)

La contraparte de *CLI* es la interfaz gráfica de usuario que ofrece una estética mejorada y una mayor simplificación a costa de un mayor consumo de recursos computacionales, y, en general, de una reducción de la funcionalidad alcanzable. Asimismo aparece el problema de una mayor vulnerabilidad por complejidad. (Potencier, 2008)

Las *CLI* son usadas por muchos programadores y administradores de sistemas como herramienta primaria de trabajo, especialmente en sistemas operativos basados en *Unix*; en entornos científicos y de ingeniería, y un subconjunto más pequeño de usuarios domésticos avanzados. (Potencier, 2008)

1.3.2. PEAR

PEAR (*PHP Extension and Application Repository*) es un *framework y sistema de distribución para componentes de código PHP reutilizables*. PEAR permite descargar, instalar, actualizar y desinstalar scripts de PHP. Si se utiliza un paquete de PEAR, no es necesario decidir donde guardar los scripts, cómo hacer que se puedan utilizar o cómo extender la línea de comandos (*CLI*). (Potencier, 2008)

PEAR es un proyecto creado por la comunidad de usuarios de PHP, está desarrollado con PHP y se incluye en las distribuciones estándar de PHP. (Potencier, 2008)

El propósito de PEAR es proveer:

- ✓ Una biblioteca de código abierto bien estructurada para usuarios de PHP.
- ✓ Mantener un sistema de distribución y mantenimiento de paquetes de código.
- ✓ Un sitio web, listas de correo y descargar los retrovisores de apoyo a la comunidad PHP/PEAR.

Consiste en una lista bastante grande de bibliotecas de código PHP que permiten hacer ciertas tareas de manera más rápida y eficiente reutilizando código escrito previamente por otras personas. Generalmente las bibliotecas contienen clases en archivos PHP que luego se incluyen y usan sin muchas complicaciones.

PEAR también proporciona funciones para manejar errores y fija el comportamiento en caso de error. Además, proporciona a los desarrolladores de paquetes una serie de funciones para hacer sus vidas más sencillas.

PEAR es el método más profesional para instalar librerías externas en PHP. *Symfony* aconseja el uso de PEAR para disponer de una instalación única y centralizada que pueda ser utilizada en varios proyectos. Los *plugins* de *Symfony* son paquetes de PEAR con una configuración especial. El propio *framework Symfony* también está disponible como paquete de PEAR. (Potencier, 2008)

1.3.3. Eclipse

Eclipse es un Entorno de Desarrollo Integrado (IDE por sus siglas en Ingles) Multiplataforma de código abierto escrito en Java. Esta plataforma, ha sido usada para desarrollar otros IDE, como el llamado *Java Development Toolkit* (JDT) y el compilador (ECJ) que se embarca como parte de Eclipse (y que son usados también para desarrollar el mismo Eclipse).

Eclipse emplea módulos (en inglés *plugins*) para proporcionar toda su funcionalidad al frente de la plataforma de cliente rico, lo que le permite extenderse usando varios lenguajes de programación como PHP, Python, C++ a diferencia de otros entornos monolíticos donde las funcionalidades están todas incluidas, las necesite el usuario o no. Este mecanismo de módulos es una plataforma ligera para componentes de software. Eclipse ha ganado muchos programadores que lo encuentran más rápido y estable que otros IDE. (Potencier, 2008)

El *plugin* PDT (*PHP Development Tools*) es usado para el desarrollo en PHP y es de ayuda también en lenguajes para web como HTML y CSS. Eclipse presenta muchas características entre las que se encuentran:

- ✓ Editor de texto.
- ✓ Resaltado de sintaxis.

- ✓ Compilación en tiempo real.
- ✓ Pruebas unitarias.
- ✓ Control de versiones con CVS.
- ✓ Asistentes para creación de proyectos, clases y test.
- ✓ Refactorización.

1.3.4. NetBeans

Es un IDE de código abierto pensado para escribir, compilar, depurar y ejecutar programas. Está escrito en Java pero puede servir para cualquier otro lenguaje de programación. Existe además un número importante de módulos para extender el IDE NetBeans. NetBeans es un producto libre y gratuito sin restricciones de uso al igual que Eclipse.

NetBeans permite crear aplicaciones Web con PHP 5, un potente *debugger* integrado y además viene con soporte para *Symfony* un gran *framework* MVC escrito en PHP. Al tener también soporte para AJAX, cada vez más desarrolladores de aplicaciones LAMP o WAMP, están utilizando NetBeans como IDE.

Sin embargo muchos programadores prefieren Eclipse porque es más fácil de usar gracias al diseño general del IDE, el cual permite mantener todas las herramientas necesarias al alcance inmediato.

1.4. Metodología de Desarrollo

1.4.1. Desarrollo Rápido de Aplicaciones (*Rapid Application Development* - RAD)

Durante mucho tiempo, la programación de aplicaciones web fue una tarea tediosa y muy lenta. Siguiendo los ciclos habituales de la ingeniería del software (como los propuestos por el *Proceso Racional Unificado* o *Rational Unified Process*) el desarrollo de una aplicación web no puede comenzar hasta que se han establecido por escrito una serie de requisitos, se han creado los diagramas UML (*Unified Modeling Language*) y se ha producido abundante documentación sobre el proyecto. Este modelo se veía favorecido por la baja velocidad de desarrollo, la falta de versatilidad de los lenguajes de programación (antes de ejecutar el programa se debe construir, compilar y reiniciar) y sobre todo por el hecho de que los clientes no estaban dispuestos a adaptarse a otras metodologías. (Potencier, 2008)

Hoy día, las empresas reaccionan más rápidamente y los clientes cambian de opinión constantemente durante el desarrollo de los proyectos. De este modo, los equipos de desarrollo deben adaptarse a esas necesidades y tienen que poder cambiar la estructura de una aplicación de forma rápida. Afortunadamente, el uso de lenguajes de script como PHP permite seguir otras estrategias de programación, como DRA (*Desarrollo Rápido de Aplicaciones*). (Potencier, 2008)

El Desarrollo Rápido de Aplicaciones es un modelo de proceso del desarrollo del software lineal secuencial que enfatiza un ciclo de desarrollo extremadamente corto. DRA es una adaptación a "Alta velocidad" con el que se logra el desarrollo rápido utilizando un enfoque de construcción basado en componentes. Si se comprenden bien los requisitos y se limita el ámbito del proyecto, el proceso DRA permite al equipo de desarrollo crear un "sistema completamente funcional" dentro de periodos cortos de tiempo. Se utiliza principalmente para aplicaciones de sistemas de información. (Potencier, 2008)

Una de las ideas centrales de esta metodología es que el desarrollo empieza lo antes posible para que el cliente pueda revisar un prototipo que funciona y pueda indicar el camino a seguir. A partir de ahí, la aplicación se desarrolla de forma iterativa, en la que cada nueva versión incorpora nuevas funcionalidades y se desarrolla en un breve espacio de tiempo. (Potencier, 2008)

Esta modalidad de desarrollo consiste de diferentes etapas que suceden de forma paralela y exigen la colaboración de los usuarios en todos los niveles. Por el contario, en la metodología de diseño tradicional, las etapas suceden de forma lineal y el usuario es excluido del proceso, lo que hace que esta modalidad sea más lenta y poco eficiente. DRA enfatiza el desarrollo de componentes de programas reutilizables. La reutilización es la piedra angular de las tecnologías de objetos, y se encuentra en el modelo de proceso de ensamblaje.

Symfony es la herramienta ideal para el DRA. De hecho, el *framework* ha sido desarrollado por la empresa *Sensio S.A* que aplica el DRA a sus propios proyectos. Por este motivo, aprender a utilizar *Symfony* no es como aprender un nuevo lenguaje de programación, sino que consiste en aprender a tomar las decisiones correctas para desarrollar las aplicaciones de forma más efectiva. (Potencier, 2008)

1.5. Propuesta de Migración

1.5.1. Necesidad de migración del subsistema GCPA a nuevas versiones

Con la versión 1.3 del *framework Symfony* los ocho subsistemas que serán desarrollados en el proyecto SGF podrán contar con:

- ✓ Corrección de la seguridad que le garantiza a la aplicación frenar los ataques *XSS* y *CSRF*. La eliminación de esta vulnerabilidad es fundamental para el desarrollo de la aplicación SGF y que esta versión 1.3 de *Symfony* lo deja atrás después de tener este agujero por largo tiempo.
- ✓ Nuevas funcionalidades que permiten su desarrollo con mayor facilidad.
- ✓ Mayor tiempo de soporte técnico que ayuda a gestionar errores de implementación durante la construcción de la aplicación.
- ✓ Una nueva arquitectura llamada ¨*Plataforma de Symfony*¨ formada por varias clases relacionadas entre sí pero completamente desacopladas.
- ✓ Integración con los restantes subsistemas de la aplicación haciéndolos compatibles.

Dado que GCPA fue el primer subsistema desarrollado por el equipo del proyecto SGF en la versión 1.0 del *framework Symfony* y que todos los restantes subsistemas serán implementados en la versión 1.3, se hace imprescindible su migración con el objetivo de realizar una integración a la aplicación de manera satisfactoria.

Es necesario destacar que solo se va a migrar GCPA mientras que los demás subsistemas se desarrollan en la versión 1.3. No se harán migraciones en los subsistemas restantes de la aplicación a otra versión.

Uno de los elementos más decisivos a la hora de elegir la propuesta de migración a la versión 1.3 de *Symfony* es la corrección de los problemas de seguridad presentes en las versiones anteriores. Esta vulnerabilidad es una preocupación de suma importancia para la Fiscalía a la hora de preservar sus datos.

Este es el momento más idóneo para realizar la migración del proyecto, debido a que solo se ha concluido uno de los ocho subsistemas a desarrollar. Comenzar el proceso de migración en una etapa superior del proyecto conllevaría a un mayor tiempo y esfuerzo invertido por los desarrolladores del proyecto SGF. Por

tanto se tomó la decisión de comenzar a desarrollar los demás subsistemas en la versión 1.3 de *Symfony* e integrar GCPA (a la aplicación) una vez se concluyera la migración del mismo.

1.6. Conclusiones

Al concluir el presente capítulo, se logra tener un mayor entendimiento de las razones por las que se realiza este trabajo. Esto se logra a través del cumplimiento de la tarea de investigación propuesta anteriormente concerniente a la realización del estudio de las diferentes herramientas y métodos de migración de las aplicaciones web desarrolladas en *Symfony*. Con lo que se concluye con la siguiente selección de herramientas y métodos para la proseguir a realizar la migración del Subsistema GCPA:

- *CLI* como herramienta para ejecutar las tareas de migración.
- *PEAR c*omo *frameworks* para la ejecución de los script PHP debido a que esta propuesto por *Symfony*.
- Es seleccionado *Eclipse* como entorno de desarrollo con el *plugin PDT* para integrar el leguaje *PHP* debido a que en el proyecto Sistema de Gestión Fiscal existe experiencia de trabajo con dicha herramienta.
- *DRA* como metodología de desarrollo ya que es la metodología que propone *Symfony* para el desarrollo de aplicaciones en su *framework*. Permitiendo, en conjunto con las nuevas funcionalidades de la versión 1.3 de *Symfony*, acortar el tiempo de desarrollo de una aplicación.

Capítulo 2. Características del Sistema

2.1. Introducción

En este capítulo describe las principales características del Subsistema GCPA, así como la estructura que lo componen y sus objetivos principales. Se describe también los cambios introducidos por el *framework Symfony* al código de la aplicación en cada una de las migraciones.

2.2. Glosario de términos

El glosario de términos ayuda a los usuarios, clientes, desarrolladores y otros interesados a utilizar un vocabulario común. La terminología común permite compartir el conocimiento con los otros por lo que es muy importante para lograr un acuerdo entre los desarrolladores, en lo referente a la definición de los diferentes conceptos y nociones, y para disminuir en general, el riesgo de confusiones. En el glosario se definirán los términos comunes utilizados en el proceso de migración.

A continuación se presenta el glosario de términos.

Helpers: es una función PHP definida por *Symfony* y que está pensada para ser utilizada en las plantillas. Los *helpers* generan código HTML y normalmente resultan más eficientes que escribir a mano ese mismo código HTML. Permiten encapsular los efectos JavaScript compatibles con todos los navegadores en una única línea de código.

Prototype: es una librería de JavaScript muy completa que amplía las posibilidades del lenguaje de programación, añade todas esas funciones que faltaban y con las que los programadores soñaban y ofrece nuevos mecanismos para la manipulación de los elementos DOM.

PDO: (*PHP Data Objects*) como capa de abstracción de bases de datos.

Plugins: es una extensión encapsulada para un proyecto *Symfony*. Los *plugins* permiten no solamente reutilizar código propio, sino que permiten aprovechar los desarrollos realizados por otros programadores y permiten añadir al núcleo de *Symfony* extensiones realizadas por otros desarrolladores. Permiten agrupar todo el código diseminado por diferentes archivos y reutilizar este código en otros proyectos. Los

plugins permiten encapsular clases, filtros, *helpers*, archivos de configuración, tareas, módulos y extensiones para el modelo.

Widgets: representa un campo de un formulario.

2.3. Características de GCPA

GCPA es uno de los 8 subsistemas del proyecto SGF. En él se encuentran procesos y funcionalidades que intervienen en la Fiscalía General de la República de Cuba.

GCPA está formado por 6 módulos que garantizan el cumplimiento de los objetivos siguientes:

- **Capacitación:** Garantiza registrar, modificar y visualizar los datos de capacitación de los cuadros y de las categorías docentes y/o fiscales.
- **Captación de Plantillas:** Le permite al "Fiscal Jefe de Dirección de Cuadro y Capacitación": registrar los datos de captación de plantilla y de modificarlos.
- **Generalidades:** Para visualizar las notificaciones que envía el sistema y las acciones de los usuarios del sistema. También permite a los usuarios del sistema cambiar su contraseña y ver su perfil.
- **Gestión de Cuadros:** Permite registrar, modificar, visualizar los datos relacionados con los cuadros (datos laborales, datos de reserva y evaluaciones), administrar los permisos del cuadro, registrar los datos relacionados con las resoluciones y acuerdos de los cuadros e imprime los mismos de ser necesario. Permite hacer una búsqueda simple.
- **Personal de Apoyo:** Permite registrar y modificar los datos de captación del Asistente Fiscal y de otras personas. Puede realizar búsquedas simples de personas en el sistema.
- **Reportes y Búsquedas:** Permite realizar búsquedas avanzadas de los cuadros, las capacitaciones, generar expedientes y currículos de los cuadros y del personal de apoyo y realizar reportes que abarquen las diferentes provincias y municipios del país. Permite además imprimir cada uno de los reportes, expedientes y currículos del cuadro y del personal de apoyo generados.

GCPA cuenta con un total de 69 diagramas de casos de uso del sistema de los cuales el 80 % son casos de usos críticos. Tiene un modelo de datos que contiene 95 tablas. La seguridad del subsistema GCPA está garantizada a nivel de usuario. Para ello cada usuario del sistema cuenta con una contraseña, un nivel de accesibilidad según las credenciales de cada rol de cada usuario del sistema y un historial donde se guardan las acciones que realiza el mismo en el sistema.

2.4. Estructura de los Proyectos de Symfony

Un elemento necesario a la hora de realizar la migración de una aplicación de *Symfony* es conocer la estructura predefinida de los proyectos. *Symfony* proporciona una estructura en forma de árbol de archivos para organizar de forma lógica todos los contenidos de un proyecto web, además de ser consistente con la arquitectura MVC utilizada. Cada vez que se crea un nuevo proyecto, aplicación o módulo, se genera de forma automática la parte correspondiente de esa estructura.

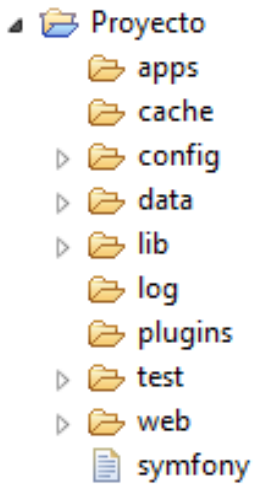

Imagen 1: *Estructura de los proyectos de Symfony.*

Directorio	Descripción
apps/	Contiene un directorio por cada aplicación del proyecto (normalmente, frontend y backend para la parte pública y la parte de gestión

	respectivamente)
cache/	Contiene la versión cacheada de la configuración y (si está activada) la versión cacheada de las acciones y plantillas del proyecto. El mecanismo de cache utiliza los archivos de este directorio para acelerar la respuesta a las peticiones web. Cada aplicación contiene un subdirectorio que guarda todos los archivos PHP y HTML preprocesados
config/	Almacena la configuración general del proyecto
data/	En este directorio se almacenan los archivos relacionados con los datos, como por ejemplo el esquema de una base de datos, el archivo que contiene las instrucciones SQL para crear las tablas e incluso un archivo de bases de datos de *SQLite*.
lib/	Almacena las clases y librerías externas. Se suele guardar todo el código común a todas las aplicaciones del proyecto. Contiene un subdirectorio llamado: model/ que guarda el modelo de objetos del proyecto.
log/	Guarda todos los archivos de log generados por *Symfony*. También se puede utilizar para guardar los logs del servidor web, de la base de datos o de cualquier otro componente del proyecto. Symfony crea un archivo de log por cada aplicación y por cada entorno.
plugins/	Almacena los *plugins* instalados en la aplicación.
test/	Contiene las pruebas unitarias y funcionales escritas en PHP y compatibles con el *framework* de pruebas de *Symfony*. Cuando se crea un proyecto, *Symfony* crea algunas pruebas básicas
web/	La raíz del servidor web. Los únicos archivos accesibles desde Internet son los que se encuentran en este directorio.

Tabla 1: *Directorios en la raíz de los proyectos Symfony*

2.5. Migración de Symfony 1.0 a Symfony 1.1

2.5.1. Pasos para realizar la migración de la versión 1.0 a la versión 1.1

Symfony 1.1 sólo es compatible con las versiones de PHP superiores a 5.1

Para actualizar un proyecto realizado con *Symfony 1.0* de modo que sea compatible con *Symfony 1.1*, se deben realizar los siguientes pasos:

1. Si no se utiliza una herramienta de gestión de código fuente (SCM) tipo Subversion, Perforce o Visual Source Safe, se debe realizar una copia de seguridad de todo el proyecto.
Como *Symfony* reemplaza algunos archivos durante la actualización es necesario tener una copia de los archivos originales para poder luego restaurar los cambios que habías realizado en esos archivos.

2. Actualizar el archivo llamado `symfony` que se encuentra en el directorio raíz del proyecto, cambiando las siguientes tres líneas:

```
chdir(dirname(__FILE__));
include('config/config.php');
include($sf_symfony_data_dir.'/bin/symfony.php');
```

por las siguientes cuatro líneas:

```
chdir(dirname(__FILE__));
require_once(dirname(__FILE__).'/config/ProjectConfiguration.class.php');
$configuration = new ProjectConfiguration();
include($configuration->getSymfonyLibDir().'/command/cli.php');
```

3. Crear un archivo llamado `ProjectConfiguration.class.php` en el directorio `config/`(este directorio almacena toda la configuración del proyecto) del proyecto y se añade el siguiente contenido:

```
require_once '##SYMFONY_LIB_DIR##/autoload/sfCoreAutoload.class.php';
sfCoreAutoload::register();

class ProjectConfiguration extends sfProjectConfiguration
{
  public function setup()
  {
  }
}
```

Después, se reemplaza el valor **`##SYMFONY_LIB_DIR##`** por la ruta al directorio `lib/` de la instalación de *Symfony 1.1*. Esta es la nueva forma de cambiar la versión de *Symfony* para un proyecto.

4. Ejecutar la tarea `project:upgrade1.1` en el directorio raíz del proyecto a través de las líneas de comandos en una consola para realizar una actualización automática:

```
$ php symfony project:upgrade1.1
```

Esta tarea se puede ejecutar varias veces sobre un mismo proyecto sin consecuencias negativas. La tarea de actualización hay que ejecutarla en cada proyecto que quieras actualizar a la versión *Symfony 1.1* beta o final.

5. Si no se va a realizar la actualización de la parte de validación de formularios o el sistema de envío de correo electrónico, hay que activar la opción de compatibilidad con la versión anterior en el archivo `settings.yml`:

```
all:
  .settings:
    compat_10: on
```

Cuando se activa la opción de compatibilidad con la versión anterior, se habilitan los siguientes componentes:

- Los bridges a los *frameworks Zend* y *ezComponents.*
- sfProcessCache.
- Sistema de validación (archivo `validate.yml`, clases de validación).
- Filtro *fillin* para el relleno automático de datos en los formularios.
- Envío de correos electrónicos mediante *sfMail* y *phpmailer.*

Una vez realizados los pasos de migración anteriores la aplicación se encuentra sobre la nueva versión de *Symfony*. Aplicar pruebas de caja negra permite garantizar el correcto funcionamiento del proyecto.

A continuación se describen los principales cambios introducidos por la versión 1.1 de *Symfony* al código de cualquier aplicación.

2.3.2. Principales cambios de Symfony 1.1

La tarea de migración `project:upgrade1.1` realiza los siguientes cambios de forma automática en el código de la aplicación:

- **Atributos flash**

  Los atributos flash ahora se gestionan directamente desde `sfUser`. La clave `flash` del archivo de configuración `filters.yml` desaparece porque el filtro `sfFlashFilter` ya no existe.

- **Métodos de `sfComponent` declarados obsoletos**

Varios métodos de `sfComponent` se han eliminado y ahora se pueden acceder mediante la clase `sfController`.

- **Enrutamiento**

El archivo `factories.yml` incluye una nueva configuración del sistema de enrutamiento.

Para incluir parámetros por defecto en las rutas, ahora se utiliza el método `->setDefaultParameter()` en vez de la opción de configuración `sf_routing_defaults`:

```
$this->context->getRouting()->setDefaultParameter($clave, $valor);
```

- **Sistema de log**

El archivo `factories.yml` incluye una nueva configuración del sistema de *logs* que permite manejarlos ahora a través de este archivo por lo que el archivo de configuración `logging.yml` ya no se utiliza y todos los cambios.

Se ha creado una nueva opción de configuración llamada `logging_enabled` en el archivo `settings.yml`. Esta opción permite deshabilitar completamente los *logs* en el entorno de producción.

Los niveles de log ahora se establecen mediante constantes:

```
sfLogger::INFO
```

- **Sistema de Cache**

La clase `sfFunctionCache` ya no hereda de `sfFileCache`. Además, es necesario pasar un objeto de tipo caché al constructor. El primer argumento de `->call()` ahora tiene que ser una función ejecutable de PHP.

El nombre de algunas opciones de configuración de `sfCache` ahora utiliza guiones bajos.

- **Depuración web**

La opción de configuración `web_debug` desaparece del archivo `filters.yml` porque el filtro `sfWebDebugFilter` ya no existe. La barra de depuración web se incluye en la respuesta mediante un *listener*.

- **Controladores frontales**

Son actualizados todos los controladores frontales. Las constantes `SF_DEBUG`, `SF_APP`, `SF_ENVIRONMENT` y `SF_ROOT_DIR` desaparecen. Para utilizar alguna de estas constantes hay que utilizar en su lugar el método `sfConfig::get('')`:

Constante	Método
SF_ROOT_DIR	`sfConfig::get('sf_root_dir')`
SF_ENVIRONMENT	`sfConfig::get('sf_environment')`
SF_APP	`sfConfig::get('sf_app')`
SF_DEBUG	`sfConfig::get('sf_debug')`

Tabla 2: *Actualización de los Controladores Frontales.*

Si se ha hecho alguna modificación en los controladores, *Symfony* muestra un mensaje de aviso y no los actualiza. Para este caso, se puede utilizar como base del nuevo controlador frontal el esqueleto que se encuentra en el archivo `/ruta/hasta/symfony/lib/task/generator/skeleton/app/web/index.php`

De manera general los cambios introducidos por la versión 1.1 son notables, esto responde a la nueva arquitectura de clases desacopladas del *framework Symfony*. Muchos son los archivos por lotes se han declarado obsoletos por lo que desaparecen y se transforman en tareas de la línea de comandos (CLI). La nueva línea de tareas permite añadir fácilmente comandos a los proyectos *Symfony*, para lo cual es necesario consultar el capítulo 16 de la *Guía de Symfony 1.1* para comprender los detalles de las tareas.

Singletons

Los objetos `sfI18N`, `sfRouting` y `sfLogger` se han transformado en factorías (definición de una clase) y por tanto ya no son singletons.

Para obtener estos objetos desde el código de la aplicación, se debe utilizar el objeto `sfContext`:

```
sfContext::getInstance()->getI18N()
sfContext::getInstance()->getRouting()
sfContext::getInstance()->getLogger()
```

Internacionalización (i18n)

El archivo de configuración `factories.yml` incluye una nueva configuración del sistema de i18n por lo que el antiguo archivo de configuración `i18n.yml` ya no se utiliza. Todos los cambios en la configuración del mecanismo de i18n se deben realizar en el archivo `factories.yml`.

La única excepción es la opción de configuración `default_culture` que ahora se configura en el archivo `settings.yml` y ya no depende del mecanismo de i18n. Si el proyecto tiene otras opciones específicas, hay que pasarlas del archivo `i18n.yml` al archivo `factories.yml` y añadir la cultura por defecto de la aplicación en el archivo `settings.yml`.

La clase `sfI18N` ya no incluye los métodos `getGlobalMessageSource()` y `getGlobalMessageFormat()`. Ahora se utilizan `getMessageSource()` y `getMessageFormat()` respectivamente.

Carga automática de clases

La opción de configuración `autoloading_function` del archivo `settings.yml` ya no se utiliza. La clase de configuración de la aplicación permite registrar las funciones y clases que se cargan automáticamente.

Con el nuevo método `sfAutoload::autoloadAgain`(), ya no es necesario borrar la caché cada vez que se crean o modifican las clases de un proyecto. Este método detecta automáticamente todos los cambios y borra la caché de carga automática de clases.

Versión

El archivo `lib/VERSION` ya no existe. Para obtener la versión actual de *Symfony*, hay que utilizar la constante `SYMFONY_VERSION` que está definida en `autoload/sfCoreAutoload.class.php`

Métodos de `sfAction` declarados obsoletos

Los siguientes métodos de `sfAction` se han declarado obsoletos y lanzan una excepción de tipo `sfConfigurationException` si la opción de configuración `sf_compat_10` vale `false`:

```
->validate()
->handleError()
```

Métodos de `sfRequest` declarados obsoletos

Los siguientes métodos de `sfRequest` se han declarado obsoletos y lanzan una excepción de tipo `sfConfigurationException` si la opción de configuración `sf_compat_10` vale `false`:

```
->getError()
->getErrors()
->getErrorNames()
->hasError()
->hasErrors()
->setError()
->setErrors()
->removeError()
```

Métodos de `sfWebRequest` declarados obsoletos

Los siguientes métodos de `sfWebRequest` se han declarado obsoletos y lanzan una excepción de tipo `sfConfigurationException` si la opción de configuración `sf_compat_10` vale `false`:

```
->getFile()
->getFileError()
->getFileName()
->getFileNames()
->getFilePath()
->getFileSize()
->getFileType()
->hasFile()
->hasFileError()
->hasFileErrors()
->hasFiles()
->getFileValue()
->getFileValues()
->getFileExtension()
->moveFile()
```

Métodos `->initialize()`

La mayoría de clases del núcleo de *Symfony* se inicializan mediante un método llamado `->initialize()`. En *Symfony 1.1* este método se invoca de forma automática desde `__construct`(), por lo que ya no es necesario invocarlo de forma manual.

Carga de archivos de configuración

Algunas de las principales clases de *Symfony* se pueden configurar mediante un archivo de tipo `.yml`:

Clase	Archivo de Configuración
sfAction	`security.yml`
sfAutoload	`autoload.yml`
sfConfigCache	`config_handlers.yml`
sfContext	`factories.yml`
sfController	`generator.yml` y `module.yml`
sfDatabaseManager	`databases.yml`
sfFilterChain	`filters.yml`
sfI18N	`i18n.yml`
sfPatternRouting	`routing.yml`
sfPHPView	`view.yml`
sfViewCacheManager	`cache.yml`

Tabla 3: *Principales Clases y Archivo de Configuración*

En *Symfony 1.1* la carga del archivo de configuración de algunas de sus partes se realiza mediante el método `loadConfiguration()`, para que sea más fácil desacoplar esas partes y utilizarlas de forma independiente al resto del *framework*:

```
sfDatabaseManager
sfI18N
sfPatternRouting
```

Tiempo de validez de las sesiones

La opción de configuración `sf_timeout` ya no se utiliza. Para modificar el tiempo de validez de las sesiones, ahora se debe modificar el archivo `factories.yml` en vez de `settings.yml`.

Configuración del sistema de enrutamiento

Las opciones `sf_suffix`, `sf_default_module` y `sf_default_action` ya no se utilizan. Para cambiar el sufijo, módulo o acción que se utilizan por defecto, se modifica el archivo `factories.yml` en vez de `settings.yml`.

Archivo de configuración `php.yml`

El archivo de configuración `php.yml` ya no existe. La única opción de configuración que se debe comprobar manualmente es `log_errors`, cuyo valor era `on` en el archivo `php.yml`. El archivo `php.yml` se reemplaza por la utilidad `check_configuration.php` que se puede encontrar en el directorio `data/bin` del proyecto y que comprueba si los sistemas son compatibles con los requisitos de *Symfony* (por ejemplo la versión de PHP que esté instalada). La aplicación se puede ejecutar desde cualquier directorio:

```
$ php /ruta/hasta/symfony/data/bin/check_configuration.php
```

Aunque esta utilidad se puede ejecutar mediante la línea de comandos, se recomienda que se ejecute desde un navegador copiándola en el directorio web raíz de la aplicación, ya que PHP puede utilizar diferentes archivos de configuración `php.ini` para la línea de comandos y para la web.

Directorio `$sf_symfony_data_dir`

En *Symfony 1.1*, el directorio `$sf_symfony_data_dir` desaparece. Todos los archivos y directorios importantes de `data` se han movido al directorio `lib`:

Ruta anterior	Nueva ruta
data/config	lib/config/config
data/i18n	lib/i18n/data
data/skeleton	lib/task/generator/skeleton
data/modules/default	lib/controller/default
data/web/errors	lib/exception/data
data/exception.*	lib/exception/data

Tabla 4: *Directorios de Symfony*

Todas las clases del núcleo de *Symfony* han modificado las rutas de acceso anteriores.

sfLoader

Todos los métodos estáticos de `sfLoader` (excepto `::getHelperDirs()` y `::loadHelpers()`) se han movido a las clases `sfProjectConfiguration` y `sfApplicationConfiguration`.

sfCore

sfCore desaparece y su código se reparte entre las clases `sfProjectConfiguration`, `sfApplicationConfiguration` y `sfContext`.

config.php

Todos los archivos de configuración `config.php` se han eliminado. En su lugar, se utiliza la clase `ProjectConfiguration` y las clases de configuración de cada aplicación. Deben ser adaptados los cambios hecho en el archivo `config.php` a este nuevo archivo de configuración.

Estructura de directorios

Se han eliminado todas las constantes de `sfConfig` que acaban en `_dir_name`.

Claves de la caché

Los métodos `sfViewCacheManager::removePattern()` y `sfToolkit::clearGlob()` ya no se pueden utilizar para borrar de una vez varias partes de la caché. No obstante, el método `sfViewCacheManager::remove()` ahora permite utilizar URL internas con comodines. Este nuevo método permite borrar los contenidos de cualquier caché, no sólo los que almacena `sfFileCache`.

Layout

Las variables de las plantillas ya no están disponibles en los *layouts*. Por lo tanto, los *layouts* sólo tienen acceso a las variables globales (todas las variables que empiezan por `sf_`) y a las variables registradas mediante el evento `template.filter_parameters`.

Tareas

Se han modificado los nombres de todas las tareas de la línea de comandos de *Symfony*. El nuevo formato del nombre simula el uso de *namespaces*. No obstante, los nombres anteriores siguen funcionando a modo de alias del nuevo nombre de cada tarea. Al igual que en *Symfony 1.0*, si quieres ver la lista completa de tareas disponibles, sólo tienes que ejecutar el comando `symfony` sin ningún argumento.

2.6. Migración de Symfony 1.1 a Symfony 1.2

2.6.1. Pasos para realizar la migración de la versión 1.1 a la versión 1.2

Symfony 1.2 es compatible con las versiones de PHP superiores a 5.2.4, aunque también podría funcionar con PHP 5.2.0 o 5.2.3.

Para actualizar un proyecto realizado con *Symfony 1.1* de modo que sea compatible con *Symfony 1.2*, se deben realizar los siguientes pasos:

1. Si no se utiliza una herramienta de gestión del código fuente (SCM) tipo *Subversion*, *Perforce* o *Visual Source Safe*, se debe realizar una copia de seguridad de todo el proyecto.

2. En el directorio raíz del proyecto y ejecuta la tarea `project:upgrade1.2` a través de las líneas de comandos en una consola para realizar una actualización automática:

   ```
   $ php symfony project:upgrade1.2
   ```

   Esta tarea se puede ejecutar varias veces sobre un mismo proyecto sin consecuencias negativas. La tarea de actualización hay que ejecutarla en cada proyecto que para actualizar a la versión *Symfony 1.2* beta o final.

3. Reconstruir todas las clases del modelo y todos los formularios debido a los cambios que se describen más adelante:

   ```
   $ php symfony propel:build-model
   $ php symfony propel:build-forms
   $ php symfony propel:build-filters
   ```

4. Limpiar la caché ejecutando la siguiente tarea a través de una consola:

   ```
   $ php symfony cc
   ```

5. Actualizar el Propel: consiste en modificar el archivo de configuración `config/databases.yml` para utilizar la nueva sintaxis de *PDO*. El archivo `config/propel.ini` es modificado con la nueva *DSN* en formato *PDO* y con las opciones de configuración actualizadas.

   Localizar las siguientes líneas en tu archivo `config/propel.ini`:

```
[ini]
propel.database             = mysql
propel.database.createUrl   = mysql://username:password@localhost/
propel.database.url         = mysql://username:password@localhost/ejemplo
```

   Sustituir las líneas anteriores por las siguientes:

```
[ini]
propel.database             = mysql
propel.database.driver      = mysql
propel.database.url         = mysql:dbname=ejemplo;host=localhost
propel.database.user        = username
propel.database.password    = password
propel.database.encoding    = utf8
```

2.6.2. Principales cambios de Symfony 1.2

Propel

En *Symfony 1.2 Propel* se ha actualizado hasta la versión 1.3, lo que implica la sustitución de la capa de abstracción a datos: *Creole* por *PDO*. Como *Creole* ya no se utiliza, las siguientes clases ya no existen:

Nombre de la clase	Clase equivalente
sfCreoleDatabase	sfPropelDatabase
sfDebugConnection	DebugPDO
sfMessageSource_Creole	sfMessageSource_PDO
sfCreoleSessionStorage	sfPDOSessionStorage

Tabla 5: *Nuevas Clases de Propel*

La tarea `propel:build-db` también se ha eliminado porque *Propel 1.3* todavía no proporciona la funcionalidad de crear una Base de Datos.

La *API* de las transacciones ha cambiado ligeramente, ya que `->begin` ahora se llama `->beginTransaction()` y `->rollback()` ahora se llama `->rollBack()`.

El método para realizar consultas SQL mediante Propel `::doSelectRS` ahora se llama `::doSelectStmt`. La documentación oficial de *Propel* incluye más información sobre los cambios introducidos por *Propel 1.3*.

Además, todos los archivos de *Propel* se han pasado de `lib/propel` a `lib`. La tarea de actualización modifica el archivo de configuración `config/propel.ini` para tener en cuenta todos estos cambios.

Peticiones

Las opciones de configuración `path_info_array`, `path_info_key` y `relative_url_root` se han trasladado del archivo `settings.yml` al archivo `factories.yml` (se ha colocado dentro de la sección `param` de la configuración de la factoría `request`). Este cambio ha permitido eliminar la dependencia entre las clases `sfRequest` y `sfConfig`. Estas tres opciones de la petición ahora se pasan al constructor como cuarto argumento. Los formatos también se pasan como una opción, en vez de pasarlos como atributos.

Las constantes de `sfRequest` referidas a los métodos de la petición ahora son cadenas de texto en vez de valores numéricos y el método `sfRequest::NONE` se ha eliminado:

Constante	Valor anterior	Nuevo valor
GET	2	GET
POST	4	POST
PUT	5	PUT
DELETE	6	DELETE
HEAD	7	HEAD
NONE	1	-

Tabla 6: *Constantes de `sfRequest`.*

Los métodos `getMethod()` y `getMethodName()` ahora devuelven el mismo valor, por lo que `getMethodName()` se ha declarado obsoleto.

Se ha eliminado el método `sfAction::getMethodNames()` y su código relacionado en `sfValidationExecutionFilter` del *plugin* `sfCompat10Plugin`. Este método se declaró como obsoleto en *Symfony 1.1* y no era muy útil en *Symfony 1.0.*

Validadores

Los valores de la constante `sfValidatorSchemaCompare` se han modificado. Aunque no es necesario ningún cambio en el código, esta modificación permite utilizar atajos muy interesantes.

En *Symfony 1.1*, los validadores `sfValidatorI18nChoiceCountry` y `sfValidatorI18nChoiceLanguage` requerían una opción llamada `culture`. Como estos validadores no utilizan la cultura, esta opción `culture` se ha declarado obsoleta. No obstante, la opción no se ha eliminado para mantener la compatibilidad con las versiones anteriores de *Symfony*.

Además, se han añadido dos nuevos métodos a la clase `sfValidatorBase` que permiten establecer los mensajes por defecto de los errores `required` y `invalid`.

Forms

En *Symfony 1.1*, el archivo `BaseFormPropel` se generaba en un directorio erróneo (`lib/form/base/`). Por tanto, es necesario moverlo al directorio `lib/form/`.

Widgets

Los formularios *Propel* generados automáticamente ahora utilizan por defecto el *widget* `sfWidgetFormChoice` en vez de `sfWidgetFormSelect`. Para utilizar estos *widgets* avanzados es necesario que generes de nuevo todos los formularios:

```
$ php symfony propel:build-forms
```

Respuesta

La factoría `response` dispone de una nueva opción llamada `send_http_headers`. Esta opción vale `true` por defecto, salvo en el entorno de ejecución `test`, donde PHP no debe enviar las cabeceras (en *Symfony*

1.1 este mismo comportamiento se conseguía mediante la opción `sf_test`). Este cambio ha permitido eliminar la dependencia entre `sfResponse` y `sfConfig`.

Los métodos `getStylesheets()` y `getJavascripts()` ahora devuelven todas las hojas de estilos y todos los archivos de *JavaScript* ordenados por posición si se pasa el valor `sfWebResponse::ALL` como primer argumento.

El valor `sfWebResponse::ALL` ahora también es el valor por defecto para el argumento de la posición. Como en *Symfony 1.1* el valor por defecto es una cadena de texto vacía, los métodos sólo devuelven los archivos registrados para la posición por defecto, lo que no es muy intuitivo.

En *Symfony 1.1* se pueden obtener todos los archivos pasando como posición el valor `ALL`. Este comportamiento todavía está disponible utilizando el valor `sfWebResponse::RAW`.

Además, ahora todas las posiciones (`first`, `(posición vacía)` y `last`) están disponibles como constantes:

```
sfWebResponse::FIRST  === 'first'
sfWebResponse::MIDDLE === ''
sfWebResponse::LAST   === 'last'
```

Los métodos `removeStylesheet()` y `removeJavascript()` ahora sólo requieren un argumento, que es el archivo que se debe eliminar de la respuesta. El archivo se elimina en todas las posiciones disponibles. En *Symfony 1.1* la posición se indicaba como segundo argumento.

Prototype y Scriptaculous

Symfony continúa eliminando sus dependencias con el software externo que incluye. En *Symfony 1.2*, las librerías *Prototype* y *Scriptaculous* y sus *helpers* asociados (`JavascriptHelper`) se incluyen en forma de *plugin* incluido por defecto, también llamado *core plugin*. Estos *plugins* incluidos por defecto se comportan como cualquier otro *plugin* de *Symfony*, siendo su única diferencia que no hay que descargarlos. De esta forma, los archivos *CSS* y *JavaScript* del *plugin* `sfProtoculousPlugin` (así es como se ha decidido llamar a la combinación de *Prototype* y *Scriptaculous*) se comportan igual que los archivos de cualquier otro *plugin*. Por tanto, estos archivos ahora se encuentran en `web/sfProtoculousPlugin` en vez de `web/sf`

(como sucedía en *Symfony 1.0* y *1.1*). La opción `prototype_web_dir` ahora también apunta a ese directorio.

Además, se ha creado el grupo de *helpers* `JavascriptBaseHelper`, que incluye unos *helpers* de *JavaScript* muy básicos que funcionan con cualquier *framework* de *JavaScript* y que por tanto, se incluye dentro del núcleo del *framework*.

Por último, `javascript_tag()` ahora se comporta como un `slot`(), lo que permite crear código como el siguiente:

```
<?php javascript_tag() ?>
alert('All is good')
<?php end_javascript_tag() ?>
```

Los *plugins* normalmente se instalan mediante una tarea y esta tarea invoca la creación de un enlace simbólico en el directorio web.

Navegador

Las clases `sfBrowser` y `sfTestBrowser` se han refactorizado en cuatro clases:

- `sfBrowserBase`: la clase base del navegador. No está directamente relacionada con Symfony, aunque utiliza algunas clases de lo que se conoce como "*Plataforma Symfony*".
- `sfBrowser`: hereda de `sfBrowserBase` e incluye los métodos específicos de *Symfony*.
- `sfTestFunctionalBase`: la clase base de las pruebas funcionales. Implementa métodos para pruebas independientes de *Symfony*.
- `sfTestFunctional`: hereda de `sfTestFunctionalBase` e implementa los métodos para pruebas específicas de *Symfony*.
- `sfTestBrowser`: una clase que sólo existe para mantener la compatibilidad con las versiones anteriores de *Symfony*. Esta clase es la misma que `sfTestFunctional`, pero tiene un constructor compatible con el que se utilizaba en *Symfony 1.1.*

El motivo por el que se han refactorizado las clases es que ahora `sfTestFunctional` es una clase sólo para pruebas, no un navegador. Por tanto, esta clase utiliza como argumentos un navegador y un objeto para pruebas.

La clase `sfTestFunctional` actúa como un intermediario de la clase del navegador, lo que significa que todos los métodos del navegador son accesibles desde el objeto utilizado para la prueba funcional.

Esta refactorización no provoca ninguna incompatibilidad con *Symfony 1.1*.

Las clases del navegador ahora añaden la cabecera *HTTP_REFERER* en todas las peticiones.

Pruebas (Testers)

La clase `sfTestFunctionalBase` ahora delega la ejecución de las pruebas a las clases de tipo `sfTester`, de las que *Symfony 1.2* incluye las siguientes:

- **request:** `sfTesterRequest`
- **response:** `sfTesterResponse`
- **user:** `sfTesterUser`
- **view_cache:** `sfTesterViewCache`

Todos los métodos disponibles anteriormente en el navegador de pruebas se han trasladado a alguna clase de tipo *tester*:

Nombre del método	Clase tipo tester	Nombre del método nuevo
isRequestParameter	sfTesterRequest	isParameter
isRequestFormat	sfTesterRequest	isFormat
isStatusCode	sfTesterResponse	isStatusCode
responseContains	sfTesterResponse	contains
isResponseHeader	sfTesterResponse	isHeader
checkResponseElement	sfTesterResponse	checkElement
isUserCulture	sfTesterUser	isCulture
isCached	sfTesterViewCache	isCached

isUriCached	sfTesterViewCache	isUriCached

Tabla 7: *Métodos de Prueba*

Las clases tester también disponen de los siguientes métodos nuevos:

Clase tester	Nombre del nuevo método
sfTesterRequest	hasCookie
sfTesterRequest	isCookie
sfTesterRequest	isMethod
sfTesterUser	isAuthenticated
sfTesterUser	hasCredential
sfTesterUser	isAttribute
sfTesterUser	isFlash

Tabla 8: *Clases de Prueba*

Aunque los métodos viejos se han declarado obsoletos, no muestran ningún mensaje de aviso y no se han eliminado para mantener la compatibilidad con las versiones 1.0 y 1.1 de *Symfony*.

Enlaces

Cuando se simula la pulsación de un botón o de un enlace, se utiliza su nombre con el método `click()`. El problema es que no se puede diferenciar dos enlaces o dos botones con el mismo nombre.

A partir de la versión 1.2 de *Symfony*, el método `click()` acepta un tercer argumento con el que se pueden pasar algunas opciones.

Se puede pasar por ejemplo una opción llamada `position` para seleccionar la posición del enlace que se quiere pinchar:

```
$b->
  click('/', array(), array('position' => 1))->
  // ...
;
```

Por defecto *Symfony* siempre pincha sobre el primer enlace que encuentra en la página. También se puede pasar una opción llamada `method` para modificar el método del enlace o formulario que se está pulsando. Esta última opción es muy útil cuando un enlace se convierte mediante *JavaScript* en un formulario generado dinámicamente.

Acciones

Por defecto, cuando se utilizan los métodos `redirectIf()` o `redirectUnless()` en las acciones, *Symfony* modifica automáticamente el código de estado de HTTP a `302`. En *Symfony 1.2* estos dos métodos disponen de un argumento opcional que permite cambiar este código de estado de `HTTP::`

```
$this->redirectIf($condition, '@homepage', 301);
$this->redirectUnless($condition, '@homepage', 301);
```

El método `redirect()` ya disponía de esta característica.

`sfParameterHolder`

El método `has()` de `sfParameterHolder` se ha modificado para ser más correcto semánticamente. Ahora este método devuelve true incluso cuando el valor es `null`:

```
$ph = new sfParameterHolder();
$ph->set('variable1', 'valor');
$ph->set('variable2', null);


$ph->has('variable1') === true;
$ph->has('variable2') === true; // en Symfony 1.0 y 1.1 devuelve false
```

El método `sfParameterHolder::has()` lo utilizan `hasParameter()` y `hasAttribute()`, que a su vez están disponibles en numerosas clases del núcleo de *Symfony*.

Tareas

Algunas tareas toman como argumento el nombre de la aplicación, ya que requieren conectarse con la base de datos. El nombre de la aplicación es imprescindible porque la configuración se puede redefinir en cada aplicación y todo el sistema de configuración de *Symfony* se basa en el nivel de la aplicación.

Este argumento se ha eliminado en todas esas tareas y ahora se utiliza una opción llamada `application`. Si no se indica la opción `application` *Symfony* utiliza la configuración de la base de datos existente en el archivo `databases.yml` del proyecto.

La declaración de las siguientes tareas se ha modificado para cumplir con todo lo anterior:

```
propel:build-all-load
propel:data-dump
propel:data-load
```

Este comportamiento es posible porque `sfDatabaseManager` ahora utiliza la configuración de un proyecto o de una aplicación. Este funcionamiento es debido a que `sfDatabaseConfigHandler` ahora devuelve una configuración estática o dinámica en función de un `array` de archivos de configuración (ver los métodos `execute()` y `evaluate()`).

La tarea `propel:insert-sql` elimina todos los datos existentes en la base de datos utilizada. Como es una tarea que destruye información ahora solicita al usuario la confirmación de si realmente quiere ejecutar la tarea. Esta confirmación también la solicitan las tareas `propel:build-all` y `propel:build-all-load`, ya que las dos invocan a la tarea `propel:insert-sql`.

Si se utiliza la tarea en un script, puedes evitar esta confirmación utilizando la opción `no-confirmation`:

```
$ php symfony propel:insert-sql --no-confirmation
```

En las versiones anteriores a *Symfony 1.2*, la tarea `propel:insert-sql` era la única que obtenía la configuración de la base de datos en el archivo `propel.ini`. Desde *Symfony 1.2*, esta tarea obtiene la configuración en el archivo `databases.yml`. De esta forma, si se utilizan diferentes conexiones en el modelo, esta tarea las tiene en cuenta. Además, gracias a este nuevo comportamiento se puede utilizar la opción `--connection` sólo para cargar las sentencias SQL de una conexión concreta:

```
$ php symfony propel:insert-sql --connection=propel
```

También se puede utilizar las opciones `--env` y `--application` para seleccionar la configuración concreta que se utiliza:

```
$ php symfony propel:insert-sql --env=prod --application=frontend
```

Además, la opción `non-atomic-actions` de la tarea `propel:generate-module` se ha eliminado y se han añadido las siguientes nuevas opciones:

- **singular:** el nombre en singular de las acciones y plantillas.
- **plural:** el nombre en plural de las acciones y plantillas.
- **route-prefix:** el prefijo que utilizan las rutas.
- **with-propel-route:** si se deben generar rutas de *Propel*.

Para facilitar la depuración, las tareas `propel:build-model`, `propel:build-all` y `propel:build-all-load` no eliminan los esquemas XML si se les pasa la opción `--trace`.

Helpers para URL

La opción `post` del helper `link_to` se ha declarado obsoleta, aunque todavía se puede utilizar:

```
// la siguiente instrucción es obsoleta
<?php echo link_to('@some_route', array('post' => true)) ?>


// la siguiente instrucción es equivalente a la anterior
<?php echo link_to('@some_route', array('method' => 'post')) ?>
```

Los *helpers* `url_for()` y `link_to()` también se pueden utilizar de otra forma. Además de la URI interna completa, ahora también pueden tomar como argumentos el nombre de la ruta y un `array` de parámetros:

```
echo url_for('@article', array('id' => 1));
echo link_to('Link to article', '@article', array('id' => 1));
```

La forma anterior de utilizar estos dos *helpers* todavía sigue funcionando, por lo que no es necesario realizar ningún cambio en el código.

Helper de las imágenes

En *Symfony 1.0* y *1.1*, el *helper* `image_tag` genera el atributo `alt` de la etiqueta <img /> a partir del nombre del archivo de la imagen. En *Symfony 1.2*, este comportamiento sólo se produce si está activada la compatibilidad con las versiones anteriores (`sf_compat_10` vale `on`). De esta forma, ahora es mucho más fácil utilizar un validador XHTML para encontrar los atributos `alt` vacíos. Además, existe una opción

llamada `alt_title` que establece los atributos `alt` y `title` al mismo valor, lo que es útil para mostrar mensajes de tipo `tooltip` que funcionan en cualquier navegador.

Vista

Ahora se puede redefinir el método `sfViewCacheManager::generateCacheKey()` si se utiliza la opción `sf_cache_namespace_callable`. En *Symfony 1.2*, el elemento ejecutable se invoca con un argumento adicional que es la instancia del gestor de la caché de la vista.

Configuración

En las versiones anteriores a *Symfony 1.2*, se cargaban todos los *plugins* instalados en el directorio `plugins` y todos los *plugins* incluidos por defecto en el *framework*.

A partir de *Symfony 1.2*, es obligatorio habilitar los *plugins* a utilizar en cada proyecto. La activación se realiza en la clase `ProjectConfiguration`.

Para activar varios *plugins* simultáneamente se utiliza un `array` con el nombre de todos los *plugins*. El orden en el que se cargan los *plugins* se modifica mediante el método `setPlugins`.

La opción `orm` del archivo `settings.yml` se ha declarado obsoleta y ahora se establece de forma automática cuando se carga el *plugin* correspondiente a un ORM.

De la misma forma, la opción `compat_10` del archivo `settings.yml` también se ha declarado obsoleta y también se establece de forma automática cuando se carga el *plugin* `sfCompat10Plugin`. Por tanto, para activar la compatibilidad con *Symfony 1.0*, hay que activarla en la configuración del proyecto. *Symfony* sólo activa por defecto el *plugin Propel*.

Una forma más cómoda de activar todos los *plugins* instalados consiste en utilizar el método `enableAllPluginsExcept` dentro del archivo `ProjectConfiguration.class.php`.

La clase `sfLoader` se ha declarado obsoleta porque los métodos `getHelperDirs()` y `loadHelpers()` ahora son parte de la clase `sfApplicationConfiguration`.

Configuración de los plugins

Los *plugins* ahora pueden incluir una clase de configuración del proyecto. Estas clases de configuración de los *plugins* establecen la carga automática de clases de los *plugins* y se instancian en `sfProjectConfiguration`. Esto significa que las tareas de *Symfony 1.2* ya no requieren el nombre de la aplicación para las clases de los *plugins*. Además, esto permite que los *plugins* puedan acceder a los eventos de tipo `command`.*, lo que no era posible hasta el momento.

Internacionalización

La clase `sfCultureInfo` ahora sólo se utiliza internamente como un singleton. Como es posible saltarse el método `getInstance()` e instanciar un nuevo objeto directamente, se ha declarado como obsoleto. El rendimiento mejora ligeramente por utilizar un singleton, ya que en cada petición sólo se instancia un objeto con la información de la cultura. Además, ahora también es posible redefinir globalmente parte de la información de la cultura desde las clases de configuración.

Los métodos `sfCultureInfo::getCountries()`, `sfCultureInfo::getCurrencies()` y `sfCultureInfo::getLanguages()` ahora aceptan un tercer argumento que permite restringir el valor devuelto:

```
// sólo devuelve el nombre de Francia y España en inglés
$paises = sfCultureInfo::getInstance('en')->getCountries(array('FR', 'ES'));
```

El método `sfCultureInfo::getCurrencies()` ahora devuelve un `array` con el nombre de las divisas. En las versiones anteriores de *Symfony*, este método devolvía un `array` con el símbolo y el nombre de cada divisa. Para mantener el comportamiento anterior, puedes pasar el valor true como segundo argumento del método:

```
$divisas = sfCultureInfo::getInstance('en')->getCurrencies(null, true);
```

Para obtener la traducción del nombre de un único país, idioma o moneda, se pueden utilizar respectivamente los métodos `getCountry()`, `getCurrency()` y `getLanguage()` de la clase `sfCultureInfo`.

Plantillas de error para las excepciones

Symfony ahora tiene en cuenta el formato de la petición cuando muestra el mensaje de error de alguna excepción. Se puede personalizar cada formato añadiendo una plantilla en el directorio `config/error` del proyecto o de la aplicación.

Si por ejemplo se produce un error durante una petición XML, se muestra la plantilla `config/error/exception.xml.php` cuando la aplicación se ejecuta en el entorno de desarrollo y `config/error/error.xml.php` cuando la aplicación está en producción.

2.7. Migración de Symfony 1.2 a Symfony 1.3

2.7.1. Pasos para realizar la migración de la versión 1.2 a la versión 1.3

La actualización a *Symfony 1.3* de un proyecto creado con *Symfony 1.2* comprende los siguientes pasos:

1. Comprobar que todos los *plugins* que utiliza tu proyecto son compatibles con *Symfony 1.3*. Para realizar este paso puede dirigirse al sitio oficial de *Symfony*.

2. Si no utiliza una herramienta de control de código fuente (tipo *git* o *Subversion*), haga una copia de seguridad del proyecto.

3. Actualice los *plugins* a la versión compatible con *Symfony 1.3* mediante la tarea:

```
$ php plugin:upgrade
```

4. Ejecute la tarea `project:upgrade1.3` desde el directorio raíz del proyecto para realizar la actualización automática. Esta tarea se puede ejecutar varias veces un mismo proyecto sin consecuencias negativas.

```
$ php symfony project:upgrade1.3
```

5. Vuelva a construir los modelos y formularios para que tengan en cuenta los cambios que se explican más adelante:

```
# Para Doctrine
$ php symfony doctrine:build --all-classes
```

```
# Para Propel
$ php symfony propel:build --all-classes
```

6. Limpiar la caché con la tarea:

```
$ php symfony cc
```

7. Las antiguas clases constructoras de Propel se han sustituido por los nuevos *behavior* o comportamientos de *Propel 1.4*. Para aprovechar esta mejora, es imprescindible actualizar el archivo de configuración `propel.ini`.

Eliminar las antiguas clases constructoras:

```
; builder settings
propel.builder.peer.class                                                    =
plugins.sfPropelPlugin.lib.builder.SfPeerBuilder
propel.builder.object.class                                                  =
plugins.sfPropelPlugin.lib.builder.SfObjectBuilder
propel.builder.objectstub.class                                              =
plugins.sfPropelPlugin.lib.builder.SfExtensionObjectBuilder
propel.builder.peerstub.class                                                =
plugins.sfPropelPlugin.lib.builder.SfExtensionPeerBuilder
propel.builder.objectmultiextend.class                                       =
plugins.sfPropelPlugin.lib.builder.SfMultiExtendObjectBuilder
propel.builder.mapbuilder.class                                              =
plugins.sfPropelPlugin.lib.builder.SfMapBuilderBuilder
```

Y añadir los nuevos *behavior* o comportamientos:

```
; behaviors
propel.behavior.default                         = symfony,symfony_i18n
propel.behavior.symfony.class                                                =
plugins.sfPropelPlugin.lib.behavior.SfPropelBehaviorSymfony
propel.behavior.symfony_i18n.class                                           =
plugins.sfPropelPlugin.lib.behavior.SfPropelBehaviorI18n
propel.behavior.symfony_i18n_translation.class                               =
plugins.sfPropelPlugin.lib.behavior.SfPropelBehaviorI18nTranslation
propel.behavior.symfony_behaviors.class                                      =
plugins.sfPropelPlugin.lib.behavior.SfPropelBehaviorSymfonyBehaviors
propel.behavior.symfony_timestampable.class                                  =
plugins.sfPropelPlugin.lib.behavior.SfPropelBehaviorTimestampable
```

La tarea `project:upgrade` intenta hacer este cambio automáticamente, pero puede que no sea posible hacerlo si has realizado algún cambio en el archivo `propel.ini`.

2.7.2. Principales cambios de Symfony 1.3

Carga automática de clases

Desde *Symfony 1.3* los archivos que se encuentran bajo el directorio `lib/vendor/` ya no se cargan automáticamente. Para cargar de forma automática algunos subdirectorios de `lib/vendor`/, se debe añadir una nueva entrada en el archivo de configuración `autoload.yml` de la aplicación.

```
autoload:
  vendor_mi_libreria:
    name:      vendor_mi_libreria
    path:      %SF_LIB_DIR%/vendor/directorio_mi_libreria
    recursive: on
```

La carga automática de archivos de `lib/vendor/` presentaba problemas con una librería que dispone, haciendo procesar dos veces todos los archivos y añadía mucha información inútil en la cache. Otro problema era que si el directorio no se llama exactamente `lib/vendor/symfony/`, el cargador automático de clases del proyecto volvía a procesar todo el directorio de *Symfony* pudiendo producir errores. Además, la carga automática de clases de *Symfony 1.3* ya no distingue mayúsculas de minúsculas.

Enrutamiento

Varios métodos han sido mejorados proporcionando una conformidad a la hora de manejar el enrutamiento en los proyectos. Para ello los siguientes métodos ya no devuelven las rutas como un `array`, tal y como hacían en las versiones anteriores.

- `sfPatternRouting::setRoutes()`
- `sfPatternRouting::prependRoutes()`
- `sfPatternRouting::insertRouteBefore()`
- `sfPatternRouting::connect()`

La opción `lazy_routes_deserialize` se ha eliminado porque ya no es necesaria.

La cache del enrutamiento se encuentra desactivada, ya que se trata de la mejor opción para la mayoría de proyectos en lo que respecto al rendimiento. Si no ha sido modificada la cache del enrutamiento, se desactivará automáticamente para todas las aplicaciones. Si después de actualizar a *Symfony 1.3* el proyecto se ejecuta más lentamente es producto a que la cache del enrutamiento no se encuentra activada.

Archivos JavaScript y hojas de estilos

Eliminación del filtro común

El filtro `sfCommonFilter` encargado de enlazar de forma automática los archivos *JavaScript* y las hojas de estilos utilizados en la página se ha declarado obsoleto y ya no se utiliza por defecto. Desde ahora hay que incluir de forma explícita estos tipo de archivos mediante llamadas a los *helpers* `include_stylesheets()` y `include_javascripts()` en el layout:

```
<?php include_javascripts() ?>
<?php include_stylesheets() ?>
```

Las razones por las que se ha eliminado son las siguientes:

- Se dispone de una solución mejor, más sencilla y más flexible.
- Utilizar un control más preciso de los archivos que se incluyen en el `layout` y de dónde se incluyen (las hojas de estilos dentro de la etiqueta `<head>` y los archivos de *JavaScript* justo antes de la etiqueta `</body>` de cierre).
- Siempre es mejor ser explícito y no implícito (mejor no utilizar tanta magia y provocar tantos quebraderos de cabeza a los usuarios).
- Mejora ligeramente el rendimiento.

Para realizar este cambio solo se modifica manualmente el `layout` que se encuentra en directorios `templates/` y que no contengan una etiqueta `<head>` o cualquier página que no tenga un `layout` pero que incluya archivos *JavaScript* y hojas de estilos, ya que de modo contrario la tarea `project:upgrade1.3` realiza los cambios anteriores de forma automática.

Cabe aclarar que la clase `sfCommonFilter` todavía se incluye en *Symfony 1.3*, por lo que puede ser utilizarla en los archivos `filters.yml` si es necesario.

Tareas

Se han renombrado las siguientes clases de las tareas:

Symfony 1.2	Symfony 1.3
sfConfigureDatabaseTask	sfDoctrineConfigureDatabaseTask o sfPropelConfigureDatabaseTask
sfDoctrineLoadDataTask	sfDoctrineDataLoadTask
sfDoctrineDumpDataTask	sfDoctrineDataDumpTask
sfPropelLoadDataTask	sfPropelDataLoadTask
sfPropelDumpDataTask	sfPropelDataDumpTask

Tabla 9: *Clases de las Tareas*

El uso de las tareas `*:data-load` se ha modificado, por lo que ahora se indica como argumento los archivos o directorios específicos. Además, la opción `--dir` se ha eliminado.

Mecanismo de escape

El método `esc_js_no_entities()`, utilizado por `ESC_JS_NO_ENTITIES`, se ha actualizado para que procese correctamente los caracteres que no sean de tipo ANSI. Antes de este cambio, se escapaban todos los caracteres salvo los que tienen un valor ANSI de entre `37` y `177`. Ahora sólo aplica el mecanismo de escape a las contrabarras `\`, comillas `'` y `"` y los saltos de línea `\n` y `\r`. No obstante, es muy improbable que este cambio afecte las aplicaciones.

Integración con Doctrine

Las referencias externas con el proyecto *Doctrine* se han actualizado para utilizar su versión 1.2 más reciente.

Borrado en la parte de administración

El borrado masivo de la parte de administración se ha modificado para que primero obtenga los registros y después los borre individualmente con el método `delete()` de cada registro, en vez de utilizar una consulta SQL para borrarlos todos a la vez. El motivo del cambio es que de esta forma se puede notificar el evento de borrado para cada registro individual.

Redefinir los esquemas Doctrine de los plugins

Para redefinir los modelos de datos de los *plugins* simplemente se define ese mismo modelo en el esquema de datos local. Si por ejemplo para añadir una columna llamada `email` en el modelo `sfGuardUser` del *plugin* `sfDoctrineGuardPlugin`, se añade lo siguiente en el archivo `config/doctrine/schema.yml`:

```
sfGuardUser:
  columns:
    email:
      type: string(255)
```

La opción `package` es una característica de *Doctrine* y se utiliza en los esquemas de los *plugins* de *Symfony*. Esto no significa que la característica `package` pueda utilizarse libremente para empaquetar los modelos. De hecho, solamente se puede utilizar en los *plugins* de *Symfony*.

Mensajes de log de las consultas

Los mensajes de log de las consultas de *Doctrine* se notifican mediante el uso de `sfEventDispatcher` en lugar de acceder al objeto `logger` directamente. Además, la variable `subject` de estos eventos es o la conexión o la sentencia que está ejecutando la consulta. Los mensajes de log se generan mediante la clase `sfDoctrineConnectionProfiler`, que se puede acceder a través de un objeto de tipo `sfDoctrineDatabase`.

Plugins

En la clase `ProjectConfiguration` además utilizar el método `enableAllPluginsExcept()` para controlar los *plugins* habilitados se deben ordenar los *plugins* en función de su nombre para asegurar la consistencia en las diferentes plataformas.

Widgets

La clase `sfWidgetFormInput` ahora es abstracta. Los campos de formulario de tipo texto ahora se crean con la clase `sfWidgetFormInputText`. Este último cambio se ha realizado para facilitar la introspección de las clases.

Mailer

Symfony 1.3 dispone de una nueva factoría de tipo *mailer*. Cuando se crea una nueva aplicación, el archivo `factories.yml` define algunas opciones de configuración lógicas para los entornos `test` y `dev`. Sin embargo, si es actualizando un proyecto que no tenga esas mismas opciones de configuración a continuación se muestra la configuración de `factories.yml` para el *mailer* de estos entornos:

```
mailer:
  param:
    delivery_strategy: none
```

La configuración anterior hace que los emails no se envíen de verdad, pero que si generen mensajes de log. Además, con las opciones anteriores también funciona el *tester* llamado `mailer` en las pruebas funcionales.

Pruebas

El archivo de inicialización de las pruebas, `test/bootstrap/unit.php`, se ha actualizado para mejorar la carga automática de las clases del proyecto. La actualización consiste en añadir las siguientes líneas al script:

```
$autoload                                                                    =
sfSimpleAutoload::getInstance(sfConfig::get('sf_cache_dir').'/project_autoload.cach
e');
```

```
$autoload->loadConfiguration(sfFinder::type('file')->name('autoload.yml')-
>in(array(
  sfConfig::get('sf_symfony_lib_dir').'/config/config',
  sfConfig::get('sf_config_dir'),
)));
$autoload->register();
```

La tarea `project:upgrade` intenta hacer este cambio automáticamente, pero puede que no sea posible hacerlo si ya se realizó algún cambio en el archivo `test/bootstrap/unit.php`.

YAML

El componente `sfYAML` ahora es más compatible con la especificación 1.2 del estándar *YAML*. El principal cambio que se realiza en los archivos de configuración es que ahora los valores *booleanos* sólo se pueden representar mediante las cadenas de texto `true` o `false`.

La tarea `project:upgrade` te indica dónde utilizar la sintaxis antigua pero no la corrige (para evitar la pérdida de los comentarios por ejemplo). Deben ser corregidos estos valores a mano.

Propel

En *Symfony 1.2* la clase `BaseFormFilterPropel` se generaba incorrectamente en el directorio `lib/filter/base`. En *Symfony 1.3* se ha corregido este error y la clase ahora se genera en `lib/filter`. La tarea `project:upgrade` se encarga de mover la clase al directorio correcto.

2.8. Pruebas a la Migración

En cada una de las etapas de la migración del subsistema Gestión de Cuadros y Personal de Apoyo se realizaron pruebas de caja negra con el objetivo de verificar el funcionamiento correcto de la aplicación.

Las pruebas de caja negra se aplican a la interfaz del software, centrándose en los requisitos funcionales del sistema. Estas permiten obtener conjuntos de condiciones de entrada que ejerciten completamente todos los requisitos funcionales de un programa. Entre sus objetivos se encuentran encontrar errores en funciones incorrectas o ausentes, en la interfaz de usuario, en estructuras de datos o en accesos a base de datos externas, en el rendimiento y errores de inicialización y terminación.

Al subsistema GCPA se le aplicaron tres iteraciones de pruebas de caja negra en cada una de las migraciones con el propósito verificar el funcionamiento de la aplicación ante la nueva versión del *framework*.

En la primera iteración del último proceso de migración se detectaron problemas procedentes de los cambios introducidos por la actualización de Propel. Esto dio al como resultado errores con las consultas de las bases de datos dentro de la aplicación. Muchas consultas fueron reeditadas para arreglar esta inconsistencia.

La segunda iteración del último proceso de migración mostró problemas con el *plugin sfSslRequirementPlugin* presente por defecto en la instalación del *framework Symfony*. La solución fue descargar directamente el *Plugin* de la página oficial de Symfony.

Una tercera iteración reveló los siguientes resultados:

- ✓ El sistema responde a las funcionalidades descritas en la descripción de los CU.
- ✓ Se logra el objetivo de cada caso de uso en su flujo básico y están correctamente estructurados los pasos que corresponden al mismo así como a los del flujo alterno.

2.9. Conclusiones

A pesar de que se encuentren descritos cada uno de los pasos para realizar la migración y los cambios de arquitectura del *framework* como del código, estos difieren en algunos puntos en la práctica de la realización del proceso de migración. Un ejemplo de esto es la poca documentación respecto a los cambios introducidos por la actualización de *Propel*, lo cual provoca errores en las consultas a la Base de Datos.

La Guía de Migración para aplicaciones Web del *framework Symfony* es un artefacto generado durante el desarrollo de la migración del subsistema GCPA. Este documento recoge los pasos para realizar la migración de una aplicación Web creada sobre el *framework Symfony*, además de los errores que se pueden presentar después de migrar. Cada uno de los errores están descritos y solucionados aportando una importante información que no es encuentra documentada.

Este capítulo concluye con la satisfactoria migración del código del subsistema Gestión de Cuadros y Personal de Apoyo del proyecto Sistema de Gestión Fiscal a la versión 1.3 del *framework Symfony* y la

realización de pruebas de caja negra que permitieron develar la existencia de errores en las funcionalidades de la aplicación y corregirlos eficientemente.

Conclusiones Generales

Con el desarrollo del trabajo se considera que se ha cumplido con el objetivo y las tareas propuestas inicialmente. Por lo que se concluye expresando que:

- ✓ Se analizaron las herramientas y métodos para realizar la migración de aplicaciones web desarrolladas en el *framework Symfony*.
- ✓ En el documento se escogen los métodos y herramientas para realizar la migración del Subsistema GCPA con las justificaciones pertinentes.
- ✓ Se realizó la migración al código del subsistema GCPA, incorporándole nuevas funcionalidades y beneficios de seguridad antes mencionados.
- ✓ Se plasmaron las experiencias y los errores encontrados en el proceso de migración del *framework Symfony* en una guía para ayuda a futuros desarrolladores que lo necesiten .
- ✓ El documento recoge los resultados de las pruebas de caja negra que le fueron realizadas al subsistema GCPA para verificar el correcto funcionamiento de la aplicación.

Finalmente se considera que los elementos abordados a través del presente trabajo puedan asistir el proceso de migración de aplicaciones web desarrolladas sobre *Symfony* a versiones superiores hasta 1.3. Se pretende que tanto como el presente documento como la guía de migración, constituyas una herramienta de apoyo para este tipo de migración.

Recomendaciones

- En este trabajo se proponen un grupo de herramientas y métodos para realizar la migración de aplicaciones de Symfony. Las cuales ayudan de manera significativa a realizar el proceso de migración. Con lo cual son de especial recomendación.

- Se recomienda el uso de la Guía de migración para realizar dicho proceso en aplicaciones web desarrolladas en versiones de *Symfony* anteriores a 1.3.

Bibliografía

Wage, Jonathan H. y Vesterinen, Kosgna. *Doctrine ORM for PHP.* France : Sensio SA, 2009.

SensioLab. *What's new in symfony 1.2?* France : Sensio SA, 2009.

Potencier, Fabien. UPGRADE TO 1.2. *Symfony Project.* [En línea] http://trac.symfony-project.org/browser/branches/1.2/UPGRADE_TO_1_2.

—. UPGRADE 1.1. *Symfony Project.* [En línea] http://svn.symfony-project.com/branches/1.1/UPGRADE.

Potencier, Fabien y Zaninotto, François. *Symfony 1.2 La guía Definitiva.* France : Sensio SA, 2008.

Potencier, Fabien. *Practical symfony 1.2 for Doctrine.* France : Sensio SA, 2008.

Patricia Lozano, Diana. *Diseño y Construcción de un Módulo Inteligente para el Proyecto MINERA.* Santiago de Cali : Universidad del Valle, Santiago de Cali, Colombia, 2008.

Muñoz Tamayo, Francisco y Roberto Aballí Mor, Félix. *Herramienta para la migración de datos de Microsoft Access a PostgreSQL.* Ciudad de la Habana : Universidad de las Ciencias Informáticas, 2009.

Eguiluz, Javier. Se publica Symfony 1.2.7. [En línea] 3 de Mayo de 2009. http://www.symfony.es/2009/05/03/se-publica-symfony-127/.

—. Mejoras importantes en el rendimiento de Symfony. [En línea] 5 de Mayo de 2009. http://www.symfony.es/2009/04/05/mejoras-importantes-en-el-rendimiento-de-symfony/.

—. Actualización de Symfony 1.2 a Symfony 1.3 / 1.4. *Symfony.es.* [En línea] http://www.symfony.es/documentacion/actualizacion/actualizacion-de-symfony-1-2-a-symfony-1-3-1-4/.

Alshanetsky, Ilia. es.phpsolmag.org. *xombra.* [En línea] 05 de 11 de 2006. http://www.xombra.com/go_articulo.php?nota=67.

Security must be taken seriously. **Potencier, Fabien.** France : s.n., 2009, http://www.symfony-project.org/, pág. 1.

Potencier, Fabien y Zaninotto, François. *La Guía Definitina de Symfony 1.2.* France : Sensio.SA.

Anexos