Universidad de las Ciencias Informáticas

Ingeniería en Informática

Facultad 4

Título: Línea Base Arquitectónica para el Polo Sistemas Tributarios y de Aduanas

TRABAJO PARA OBTAR POR EL TITULO DE INGENIERO INFORMÁTICO

AUTOR:

José Antonio Cobo Rodríguez

TUTOR:

Msc. Julio C. Diaz Vera

Ciudad de La Habana
junio de 2008

DECLARACIÓN DE AUTORÍA

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

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

José Antonio Cobo Rodríguez

_____
Firma del Autor

Msc. Julio Cesar Díaz Vera

_____
Firma del Tutor

DATOS DE CONTACTO

Graduado de Ing. en Telecomunicaciones y Electrónica en el año 2003 en la universidad central de Las Villas, se titula como Máster en Gestión de Proyectos en la UCI en el año 2007, Participo en el 1er Taller de Minería de datos Louisiana – Cuba en el año 2005, ha participado como ponente en las dos ediciones de UCIENCIA publicando en las memorias de ambos eventos.

AGRADECIMIENTOS

QUISERA AGRADECER,

EN PRIMER LUGAR A MI MADRE, QUE ES MI MEJOR AMIGA Y COMPAÑERA, POR SU APOYO, CONFIANZA Y AMOR INCONDICIONAL.

A LA PERSONA QUE EN ESTOS ÚLTIMOS DOS AÑOS HA ESTADO A MI LADO EN TODOS LOS MOMENTOS, A TI YANA POR TU AMOR.

A MIS AMIGOS MÁS CERCANOS, LOS QUE SIEMPRE ME HAN AYUDADO Y APOYADO, ESTANDO CERCA O LEJOS, ESE PEQUEÑO CÍRCULO QUE SIEMPRE HA ESTADO AHÍ PARA LO QUE FUESE, LUIS, GUSTAVO, YILO, ALEXIS, LAURA, PIRI, ALEXEI, PARRA, MAYLING, FELIX, CESAR Y GISELLE.

A ESA GRAN PERSONA QUE NUNCA DEJÓ DE CONFIAR EN MÍ Y QUE SIEMPRE HE APRECIADO Y RESPETADO POR SU APOYO Y PACIENCIA, JULITO.

A LAS PERSONAS DEL PROYECTO PROFESORES, ESTUDIANTES Y TRABAJADORES DE LA ADUANA QUE TANTO APOYO HAN DADO EN TODO MOMENTO.

A TODOS LOS QUE SIEMPRE ME HAN INSPIRADO A SER UNA MEJOR PERSONA, MIS FAMILIARES, MIS AMIGOS, MIS COMPAÑEROS.

A LA REVOLUCIÓN POR DARME LA POSIBILIDAD DE SER QUIEN SOY.

A TODOS MUCHAS GRACIAS.

DEDICATORIA

QUISERA DEDICAR ESTE TRABAJO A MI FAMILIA,

A MI MADRE, QUE SIEMPRE A LUCHADO JUNTO A MI PARA QUE ESTE MOMENTO LLEGARA, AQUÍ ESTÁ EL MAYOR DE LOS REGALOS, PARA TI.

A MI ABUELA NOHEMA QUIEN ME QUIERE MÁS QUE A NADA Y ME HA DADO SIEMPRE SU AMOR INCONDICIONAL.

A MI PADRE Y MARLENE PARA QUE SIGAN SINTIENDOSE ORGULLOSOS DE MÍ.

A MI PADRASTRO, QUE AUNQUE NO ESTÉ HOY ENTRE NOSOTROS SIEMPRE PUSO GRAN EMPEÑO EN AYUDARME A SALIR ADELANTE.

A MI HERMANA Y OSMELITO PARA QUE LES SIRVA DE INSPIRACIÓN A SEGUIR ADELANTE.

A MIS TIOS QUE TANTO ME QUIEREN Y AYUDAN, OMAR, TATI, OSMEL, MODESTO Y ORLANDO.

A MIS ABUELOS MIMA Y PIPO A QUIERES QUIERO Y APRECIO MUCHO.

A MIS DEMÁS PRIMOS, PRIMAS, TIOS Y TIAS.

Y ESPECIALMENTE A YANARA, QUE AUNQUE NO CONTINUEMOS JUNTOS ES COMO PARTE DE MI FAMILIA…

A TODOS, HAGO SUYOS ESTE TRABAJO.

RESUMEN

La implementación de aplicaciones web con tecnología PHP tienen cada vez más presencia en la red, dada su integración con los distintos Sistemas Gestores de Base de Datos, la flexibilidad, eficiencia y seguridad que ha alcanzado a medida que el código ha ido mejorando en su desarrollo.

El objetivo principal de este trabajo es definir un marco arquitectónico de referencia para los sistemas integrados al Polo Productivo Sistemas Tributarios y de Aduanas (PSTA). La idea subyacente consiste en permitir a los sistemas que se creen en el futuro y a los que son desarrollados actualmente integrarse, utilizando las tendencias estándares en el desarrollo de sistemas con estas tecnologías y la implementación de patrones arquitectónicos y de diseño que garanticen eficacia y calidad en el desarrollo. Para ello se han realizado una profunda revisión del estado del arte y se han propuesto mejoras a las implementaciones de un grupo de Frameworks PHP (Symfony y Propel) que en su conjunto conforman el núcleo duro de la propuesta arquitectónica además en trabajo define un grupo de lineamientos asociados a los patrones arquitectónicos y de diseño a ser utilizados en la solución.

PALABRAS CLAVES: Arquitectura, Desarrollo Web, PHP, MVC, Symfony.

TABLA DE CONTENIDO

AGRADECIMIENTOS

DEDICATORIA

RESUMEN

INTRODUCCIÓN 1

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA 4

1.1 INTRODUCCIÓN 4
1.2 ARQUITECTURA DE SOFTWARE 4
1.2.1 Definiciones de Arquitectura de Software 5
1.2.2 ¿Por qué es importante una arquitectura de software? 7
1.3 COMPONENTE 10
1.3.1 Componente de Software 11
1.3.2 Interfaz 11
1.4 PATRONES DE SOFTWARE 11
1.4.1 Clasificación de los patrones 13
1.4.2 Beneficio del uso de Patrones. 18
1.5 FRAMEWORKS 19
1.5.1 Definición de Framework 20
1.6 FRAMEWORKS PARA PHP 21
1.6.1 CakePHP 21
1.6.2 Kumbia 24
1.6.3 Zend Framework 26
1.6.4 ERPFAR 27
1.7 SYMFONY 28
1.7.1 ¿Por qué Symfony? 32
1.7.2 Propel 36
1.8 CONCLUSIONES 38

CAPÍTULO 2: PROPUESTA DE SOLUCIÓN 39

2.1 INTRODUCCIÓN 39
2.2 REQUERIMIENTOS NO FUNCIONALES 39
2.2.1 Usabilidad 39
2.2.2 Rendimiento 39
2.2.3 Fiabilidad 40

2.2.4 Eficiencia *40*
2.2.5 Soporte *40*
2.2.6 Hardware *40*
2.2.7 Software *41*
2.2.8 Portabilidad *42*
2.2.9 Seguridad *42*
2.2.10 Estándares Aplicables *42*
2.2.11 Reutilización de Componentes *43*
2.3 DEFINICIÓN DE LA SOLUCIÓN ARQUITECTÓNICA 44
2.3.1 Programación Orientada a Objetos *44*
2.3.2 Symfony, Framework Arquitectónico *44*
2.3.3 Propel, Persistencia de los Datos *44*
2.3.4 Ext JS, Presentación y Comunicación *45*
2.3.5 WEB 2.0, Tendencia de Desarrollo Web *45*
2.4 PATRONES DE SOFTWARE 46
2.4.1 Arquitectónico *46*
2.4.2 Diseño *49*
2.5 COMUNICACIÓN 53
2.5.1 Comunicación entre capas. *53*
2.5.2 Comunicación Interna. *54*
2.6 CONCLUSIONES 54

CAPÍTULO 3: DESCRIPCIÓN ARQUITECTÓNICA 55

3.1 INTRODUCCIÓN 55
3.2 MVC SEGÚN SYMFONY 55
3.2.1 El Controlador *58*
3.2.2 La Vista *63*
3.2.3 El Modelo *70*
3.3 ORGANIZACIÓN DEL CÓDIGO 74
3.3.1 Proyecto *74*
3.3.2 Aplicación *74*
3.3.3 Módulo *74*
3.3.4 Acción *74*
3.3.5 Estructura del Directorio *75*
3.4 CONFIGURACIÓN 77
3.4.1 Configuración en PHP *77*
3.4.2 Configuración en Symfony *77*
3.4.3 Otras configuraciones *78*
3.5 PLUGINS 79
3.5.1 Instalación *80*

3.5.2 Activación *81*
3.5.3 Desinstalación *81*
3.5.4 Estructura de un Plugins *82*
3.5.5 Plugins Utilizados *83*
3.6 SEGURIDAD 87
3.6.1 Seguridad del Sistema de Directorios *87*
3.6.2 Seguridad de la Acción *88*
3.6.3 Métodos de Validación y Manejo de Errores *88*
3.6.4 Filtros *89*
3.6.5 Mecanismo de escape *90*
3.6.6 SQL Injection. Uso de Criteria *91*
3.6.7 Sistema de Enrutamiento *91*
3.7 LOGS 93
3.7.1 Logs de PHP *94*
3.7.2 Logs de Symfony *94*
3.7.3 Configuración de los Logs *95*
3.7.4 Mensajes Personalizados *95*
3.7.5 Rotación y borrado de los Logs *96*
3.8 CONCLUSIONES 97
CONCLUSIONES 98
RECOMENDACIONES 99
ANEXOS 100
BIBLIOGRAFÍA 119
GLOSARIO DE TÉRMINOS 121

ÍNDICE DE TABLAS

Tabla 1: Algunos de los sistemas que utiliza la Aduana General de la República de Cuba y las tecnologías que utilizan. 100
Tabla 2: Comparación entre CakePHP, Symfony y Zend Framework en distintos aspectos. 104
Tabla 3: Comparación entre las características de las acciones y los componentes. 105
Tabla 4: Equivalencia entre SQL y la clase Criteria 106
Tabla 5: Descripción de los directorios utilizados por un Proyecto 108
Tabla 6: Descripción de los directorios utilizados por una aplicación. 108
Tabla 7: Descripción de los directorios utilizados por un módulo. 108
Tabla 8: Listado de los Archivos de Configuración de Symfony 111
Tabla 9: Listado de las variables para manejar los errores de PHP. 118

ÍNDICE DE FIGURAS

Fig. # 1: La arquitectura de Software como un puente 5
Fig. # 2: Definición de Arquitectura de Software 7
Fig. # 3: Composición de la estructura de Symfony por otros Frameworks. 30
Fig. # 4: Estructura de funcionamiento de Propel 37
Fig. # 5: Integración de los Componentes y Tecnologías en la Solución Arquitectónica. 46
Fig. # 6: Separación de las capas lógicas del MVC. 48
Fig. # 7: Ejemplo de Implementación del patrón Decorator en Symfony 50
Fig. # 8: Implementación del Patrón MVC por Symfony 56
Fig. # 9: Ejemplo de utilización de componente en una plantilla. 68
Fig. # 10: Decoración de una plantilla mediante Slots. 69
Fig. # 11: Ejemplo de la Estructura de las Aplicaciones con Symfony en el PSTA 75
Fig. # 12: Organización física para las aplicaciones con Symfony 76
Fig. # 13: Instalación de un Plugins desde SVN de Symfony 80
Fig. # 14: Instalación de un Plugins por un canal de PEAR 80
Fig. # 15: Instalación de un Plugins desde un Archivo en el Disco Duro. 80
Fig. # 16: Activación de un Plugins en el archivo settings.yml. 81
Fig. # 17: Desinstalación de un Plugins en Symfony mediante la consola. 81
Fig. # 18: Atributo package en el archivo miPlugin/config/schema.yml 82
Fig. # 19: Configuración del Servidor Web para permitir el acceso solo al directorio "web". 87
Fig. # 20: Secuencia de ejecución de los Filtros en Symfony 90
Fig. # 21: Imprimir un mensaje desde cualquier parte del controlador. 95
Fig. # 22: Imprimir un mensaje desde cualquier parte de la vista. 95
Fig. # 23: Imprimir un mensaje desde cualquier otra parte de la aplicación. 95
Fig. # 24: Ejemplo de configuración para la rotación automática en el archivo logging.yml. 96
Fig. # 25: Ejemplo de comandos para la rotación manual de una aplicación. 96
Fig. # 26: Tendencia a utilizar CakePHP, Symfony y Zend Framework a nivel mundial 101
Fig. # 27: Tendencia a utilizar CakePHP, Symfony y Zend Framework en Estados Unidos 101
Fig. # 28: Tendencia a utilizar CakePHP, Symfony y Zend Framework en Francia. 102
Fig. # 29: Estructura de Directorios de un Plugins 116
Fig. # 30: Flujo de Trabajo del proceso de Validación y Manejo de Errores para una acción. 117

INTRODUCCIÓN

En estos momentos en la Aduana funcionan distintos sistemas que permiten el control y gestión de diversos procesos, implementados en diversos leguajes y atados a plataformas que trabajan sobre distintos sistemas operativos, especialmente los gestores de base de datos. Esta amalgama de tecnologías introduce un grupo de características no deseables para el desempeño integral de los sistemas aduanales del país. Dentro de las más nocivas, por solo señalar algunas, la escasa posibilidad de interacción entre ellos, producto de lo difícil que son de comunicar, la latencia en las actualizaciones debido a la necesidad de modificar componentes de software escritos en distintos lenguajes y para distintas plataformas y el desaprovechamiento del hardware existente por la necesidad de mantener funcionando sistemas operativos y tecnologías prácticamente en desuso en el mundo. En la tabla 1 se muestra un esquema de la distribución de algunos de estos sistemas, plataformas y lenguajes en los que han sido desarrollados y los Sistemas Gestores de Base de Datos (SBD) en los que persisten sus datos operacionales. Ver Anexo I .

El SUA[1] es la respuesta de la Aduana General de la República (AGR[2]) a la situación antes descrita. Ejecutado por el Centro de Automatización para la Información y la Dirección (CADI[3]) en un esfuerzo de integrar todos los servicios que se prestan en la Aduana en un único sistema, de forma tal que se garantice la comunicación de forma rápida, eficiente y segura entre ellos. La implementación se llevó a cabo en colaboración con la UCI[4], sin embargo el sistema ha presentado un grupo de problemas de mantenimiento, usabilidad y rendimiento debido a que se ha desarrollado de forma estructurada y que por tanto no han sido utilizadas prácticas de programación, probadas como aceptadas, y formalizadas en patrones de diseño tanto arquitectónicos como a nivel de aplicación. Otro de los problemas detectados radica en la mezcla del código de la lógica del negocio tanto en la presentación como en la base de datos violando el principio de compartimentación prácticamente un tabú en el desarrollo de software en estos días.

Durante 3 años la UCI participó en el desarrollo del SUA, manteniendo la línea tecnológica impuesta por el CADI, que implica la utilización de PHP como lenguaje de programación y Oracle 8i como gestor de Base de Datos. Fuera de esto no existían en ese momento más restricciones por parte del CADI que dictaminaran el método de implementación de la aplicación.

Esta relación de trabajo promovió la utilización de los paradigmas actuales en cuanto a programación en los nuevos componentes de software que se comenzaron a desarrollar y el acuerdo tácito de poco a poco ir sustituyendo las versiones pasadas de los sistemas por implementaciones de los mismos bajo estos estándares. Como base para el desarrollo se definió la utilización del paradigma de la Programación Orientada a Objetos (POO), la utilización del Modelo-Vista-Controlador (MVC) y de una capa de abstracción de datos como mecanismo de separación de la lógica de la aplicación. Surgiendo de esta forma la necesidad de desarrollar un marco de trabajo arquitectónico que facilite el desarrollo de nuevas aplicaciones, poniendo en manos del equipo de trabajo un marco referencial que viabilice la implementación de los sistemas, el acercamiento al estado del arte del mismo implica la utilización de Frameworks Arquitectónicos PHP que implementen MVC.

Basándose en todo lo anterior se hace necesario desarrollar un estudio de los diferentes Frameworks arquitectónicos de PHP con posibilidades de acceso a múltiples gestores de Base de Datos con la inclusión obligatoria del Oracle 8i y PostgreSQL, y la confección de un estado del arte que refleje el desarrollo en el tiempo de los mismos con énfasis en las tecnologías utilizadas a lo largo de su implementación y realizando un compendio de los principales patrones tanto arquitectónicos como de diseño, que son utilizados en la solución que brinda cada Frameworks. Logrando de esta forma una incidencia positiva en la implementación del marco arquitectónico de referencia para el PSTA. Lo que se convierte en la meta principal a lograr en este trabajo.

El problema a resolver parte de la necesidad de definir una Línea Base Arquitectónica para los nuevos sistemas que serán desarrollados en el Polo, teniendo en cuenta las restricciones tecnológicas impuestas por el cliente:

- Soporte para tecnología Linux, Apache, PHP y Oracle 8i.
- Utilización de un MVC bien definido.
- Utilización de estilos arquitectónicos (Patrones de Diseño).
- Facilitar la migración a distintas Base de Datos. Sobre todo, PostgreSQL 8.x.

Al mismo tiempo, lo suficientemente flexible para poder saltar rápidamente a tecnologías de punta, una vez que estas restricciones tecnológicas desaparezcan sin necesidad de modificar lo implementado hasta el momento.

El Objeto de Estudio principal de este trabajo se centra en los Frameworks Arquitectónicos de PHP que sean capaces de cumplir con las necesidades de la arquitectura que se desea definir:

- Posibilidades de acceso a múltiples SBD.

- Tecnologías utilizadas en su implementación.
- Implementación patrones arquitectónicos y de diseño.
- Posibilidad de integración con otros sistemas.

El Campo de Acción está enmarcado en la Arquitectura del Polo Productivo Sistemas Tributarios y de Aduanas.

Con la creación de este trabajo se esperan los siguientes **Resultados**:

- Definición de los estilos Arquitectónicos y los patrones de diseño a utilizar en la línea base.
- Definición de la línea base de arquitectura del Polo expresada en la utilización de Frameworks.

Y para dar cumplimiento a estos resultados se deben cumplir las siguientes **Tareas**:

- Realización del estado del arte de los Frameworks PHP disponibles en Software Libre.
- Definición del estilo arquitectónico, patrones de diseño y Frameworks a utilizar.
- Definición de las adaptaciones necesarias al Frameworks para asegurar los requerimientos de rendimiento y productividad.

El presente trabajo se compone de tres capítulos:

Capítulo 1: Es la fundamentación teórica del trabajo, aquí se exponen algunos conceptos importantes para el correcto entendimiento de la solución propuesta como son Arquitectura, Componente, Interfaz, Patrones de Software y Framework. Se realiza una comparación entre los Frameworks para PHP más resonados a nivel mundial y se explica las razones de la selección del Marco de Trabajo Arquitectónico para PHP, Symfony para la solución.

Capítulo 2: En este capítulo se hace referencia a los Requerimientos No Funcionales con los cuales debe cumplir la Arquitectura planteada y se hace una propuesta de solución Arquitectónica para las aplicaciones desarrolladas por el Polo Sistemas Tributarios y de Aduanas, explicando sus características más importantes.

Capítulo 3: Se adentra un poco más en la explicación de la solución propuesta entrando en los detalles más importantes a tener en cuenta a la hora de desarrollar aplicaciones sobre esta arquitectura y se explica cómo se continúa cumpliendo con los Requisitos No Funcionales planteados en el Capítulo anterior.

Capítulo 1: FUNDAMENTACIÓN TEÓRICA

1.1 Introducción

En este capítulo se realiza un estudio sobre las definiciones arquitectónicas, los marcos de trabajo arquitectónicos y Frameworks para aplicaciones Web que trabajan sobre tecnología PHP y los patrones de diseño y arquitectónicos que se utilizan en la definición de la arquitectónica, exponiendo sus principales características de forma tal que se esclarezcan los conceptos y las razones que llevan a la utilización de las herramientas y tecnologías seleccionadas en la confección de la línea base de la arquitectura para el Polo Productivo Sistema Tributario y de Aduanas.

1.2 Arquitectura de Software

Desde el surgimiento de la informática, esta se emplea para diseñar e implementar sistemas para resolver problemas y procesos de datos. A medida que los sistemas se hacen más grandes y complejos y los datos se hacen más voluminosos, a tal punto de hacer que los sistemas informáticos se asocien, pues el número de elementos aumenta y la organización y estructura de estos elementos en el sistema se hace más complicado. Rara vez se desarrollan sistemas para resolver problemas simples. Hoy, se diseña e implementa los sistemas para resolver clases de problemas –lo que es llamado el espacio de un problema-, a pesar de la relación de este.

Una arquitectura de software describe la estructura de una solución que se firma al espacio de un problema. Una arquitectura de software es el resultado de descomponer el espacio de un problema en pequeñas partes, identificando las características comunes entre las partes, buscando una solución para un grupo de partes que tienen propiedades comunes, y asociando componentes con cada una de estas partes. El espacio del problema abarca un grupo de problemas que tienen características mutuas. Así que la arquitectura debe proveer un conjunto de soluciones para resolver un grupo de problemas. Concretamente, una arquitectura de software describe como un sistema está compuesto por sus componentes. Esto especifica las características de los componentes, las interacciones entre los componentes y la topología de comunicación entre estos.

Según las notas de Garlan (2000), una arquitectura de software sirve como un puente entre los requerimientos del sistema y su actual implementación, como describe en la Fig. # 1**.** Como el tamaño y la complejidad del problema aumentan, los algoritmos y las estructuras de los datos se van perdiendo importancia frente a la necesidad de proveer de la estructura correcta a los sistemas. Específicamente la arquitectura de software de los sistemas de información se convierte en un problema de diseño crítico en sí (1). Así pues, una arquitectura de software describe no solo la estructura de la ambas soluciones, abstracta y concretamente, sino el conjunto de decisiones hechas acerca de la misma (Ejemplo: que está adentro y que está afuera) (2)

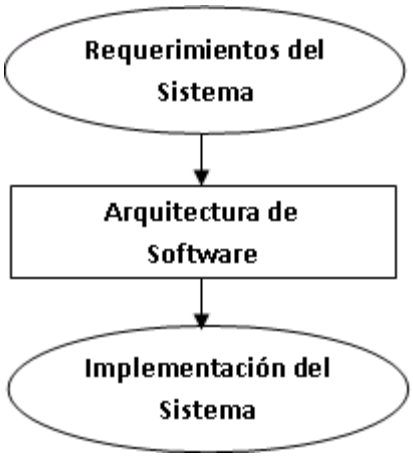

Fig. # 1: La arquitectura de Software como un puente

1.2.1 Definiciones de Arquitectura de Software

No existe una sola definición de Arquitectura de Software, a continuación se analizan varias de las más reconocidas a lo largo de la historia aunque existen más de mil.

La primera vez que se habló de Arquitectura de Software como definición fue en 1992, cuando Perry y Wolf (3) la definieron como "*un conjunto de elementos que tienen una forma común*", ellos propusieron tres tipos de elementos: procesamiento, datos y conexión. Conectando los elementos se podía distinguir una arquitectura de otra, la forma de la arquitectura está dada por enumerar las propiedades y relaciones de los diferentes elementos.

En 1996 Garlan y Shaw la conceptualizaron como "*una colección de componentes y conectores unidos con una descripción de la interacción entre esos componentes y conectores"* (4). Independientemente de estas definiciones, no se puede hablar de las definiciones de Arquitectura de Software sin mencionar el Software Engineering Institute (5). Este instituto colecciona la más vasta gama de definiciones en su sitio web oficial, entre ellas se encuentran las "definiciones clásicas", donde junto a las antes mencionadas, se encuentran las brindadas por Booch, Rumbaugh, y Jacobson en 1999: "*La arquitectura es el conjunto de decisiones significantes acerca de la organización de un sistema, la selección de los elementos estructurales y las interfaz por las que el sistema está compuesto…*" (6).

La definición "oficial" de arquitectura de software se ha acordado que sea la que brinda el documento de IEEE Std. 1471-2000 que se formula así: "*La Arquitectura de Software es la organización fundamental de un sistema encarnada en sus componentes, las relaciones entre ellos y el ambiente y los principios que orientan su diseño y evolución*".

Finalmente, Bass, Clements y Kazman (1998) ofrecieron la siguiente definición: "*La arquitectura de Software de un programa o sistema de computación es la estructura o estructura del sistema la cual comprende los elementos de software, las propiedades externamente visibles de estos elementos y las relaciones entre ellos*" (7), según el SEI, esta es la definición moderna de arquitectura. Algunas de las implicaciones de esta definición en más detalle. En cuanto a "propiedades externamente visibles" se refieren a esas asunciones que otros elementos pueden hacer de un elemento, tal como servicios proporcionados, características de funcionamiento, manejo de errores, compartido del recurso, etcétera, y así resumir diciendo que los "*elementos de Software*" pueden ser llamados "*componentes*". Si se le agrega al concepto la "*configuración*" que presenta el sistema, entonces llevaría a la redefinición del concepto para un mejor entendimiento "*La arquitectura de Software de un sistema es la estructura del sistema y su configuración la cual comprende los componentes, sus funciones y la comunicación entre ellos*".

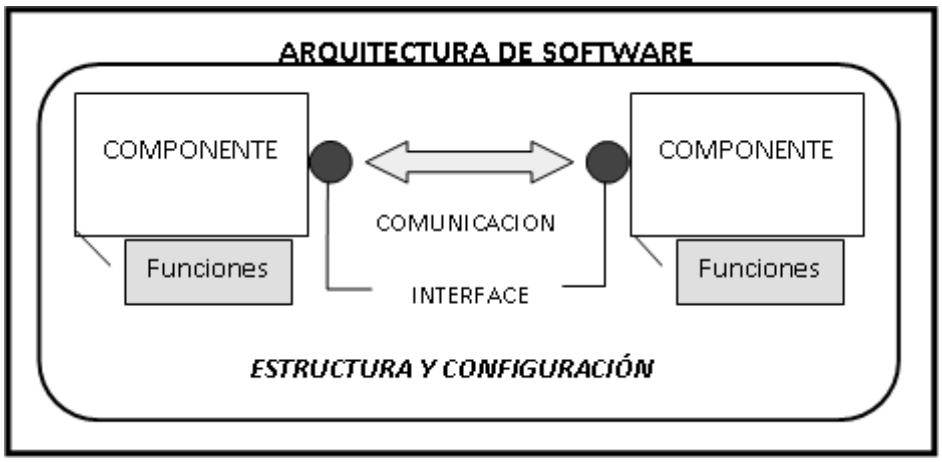

Fig. # 2: Definición de Arquitectura de Software

La arquitectura personifica información acerca de como los elementos se relacionan entre ellos. Esto significa que puede omitir cierta información de los elementos si esta no es relativa a su interacción. Así, una arquitectura es ante todo, una abstracción de un sistema que suprime los detalles de los elementos que no afecta como ellos se usan, por quien se usan, que sean relativos a, o que interactúen con otros elementos. En casi todos los sistemas modernos los elementos interactúan con cada uno de los otros por medio de interfaces que dividen los detalles acerca de un elemento en partes públicas y privadas.

1.2.2 ¿Por qué es importante una arquitectura de software?

Una Arquitectura de Software es usada como un mecanismo descriptivo, provee de información tanto para un propósito técnico como organizativo, además de que incluye la información referente al software, hardware y comunicación utilizados por y para el sistema.

La principal motivación para utilizar una arquitectura de software es que una buena arquitectura es como un buen diseño, si el tamaño y la complejidad del sistema aumenta la estructura del sistema se va a hacer más importante que la selección del algoritmo o la estructura de datos apropiada. Un buen diseño de la estructura que soportará el sistema traerá como consecuencia un mejor comportamiento y funcionalidad del mismo. Sin embargo un mal diseño arquitectónico puede anonadar la ejecución del mejor de los algoritmos. Además de que un diseño pobre del algoritmo y la estructura de datos puede dejar cojo una buena arquitectura. Una buena arquitectura de software ayuda a identificar puntos de

cómputo y cuellos de botella en los flujos de datos, así que esto permite describir las características del algoritmo y la estructura de datos adecuados para el mejor funcionamiento del sistema.

La carencia de un buen mapa como la arquitectura en sistemas grandes y complejos impide el proceso de diseño y dificulta la comunicación durante y después que el diseño haya sido implementado. Sin esta penetración en la estructura total del sistema, se toman decisiones que a menudo tienen un impacto en un subconjunto de diseños durante la implementación. Como se ha afirmado por Booch y Rumbaugh (6), es más sencillo erradicar los errores que se detectan durante el proceso de levantamiento de requisitos y documentación que durante el proceso de desarrollo del sistema. Seguramente sin una vista del concepto global del sistema como la arquitectura del sistema, problemas que pueden ser detectados rápidamente tienen gran posibilidad de ser pasados por alto. Una buena Arquitectura de Software es el resultado de la buena definición de un conjunto de principios y prácticas que son aplicadas a lo largo del tiempo de vida del proyecto. Estas son elásticas como lo son los cambios ocurridos en los procesos de levantamiento de requerimientos, funciones del negocio y las tecnologías. Y estas facilitan la guía por todo el ciclo de vida del sistema.

Uso técnico

Desde un punto de vista técnico la arquitectura describe la solución a una clase de problema. Para alcanzar esto, esta describe el conjunto de componentes, conectores, sus respectivos atributos, la configuración de esos componentes y conectores, y la semántica de la interacción entre los componentes y los conectores.

Una arquitectura de software permite la flexibilidad al arquitecto del sistema la distribución funcional de subsistemas de hardware, software y componentes de redes sin el requerimiento de un rediseño de los subsistemas que son reasignados. Esta flexibilidad ayuda a minimizar los costos de soporte de mantenimiento y evoluciona una determinada aplicación sobre su completo tiempo de vida por anticipar el tipo de cambio que va a ocurrir en el sistema, asegurando que el diseño completo del sistema facilite sus cambios, y localizando hasta donde sea posible el efecto de esos cambios. Esto además promueve la reutilización de ciertos componentes que ya existen, librerías de clases, herencias o aplicaciones de terceros, etcétera.

Una arquitectura de software además brinda información que ayuda a asegurar que los subsistemas son desarrollados en beneplácito con las mejores prácticas asociadas con los sistemas de ingeniería, desarrollo de software y gestión de programas.

Uso Organizacional

Una arquitectura de software provee una descripción de alto nivel de los subsistemas que comprenden la arquitectura del sistema a partir de la perspectiva arquitectónica. Esto ayuda al personal a entender donde existe mayor información tecnológica activa para el desarrollado y que operaciones del negocio se soportan. Al mismo tiempo, esto ayuda que los clientes principales, incluyendo los ejecutivos, a entender como las principales operaciones del negocio son soportadas. La comunicación entre los clientes en el desarrollo de una aplicación es esencial para el éxito. Ellos deben ser capaces de comunicar sus necesidades y como ellos esperan que esas necesidades sean satisfechas.

Una arquitectura de software proporciona una descripción que es descompuesta dentro de subestructuras que son relativamente independientes, tiene claras responsabilidades, y comunica con cada una de las otras a través de un grupo de interfaces bien definidas. Esto ayuda en el planeamiento y la asignación de los recursos, porque el trabajo desarrollado puede ser particionado a través de los subsistemas y el esfuerzo del desarrollo individual puede ser procesado en paralelo. La integración ocurre en hitos específicos e interfaces específicas. Cuando el equipo de desarrollo está disperso geográficamente, esto puede reducir al mínimo el número de las interacciones requeridas y fomentar el desarrollo de los sistemas según la habilidad para las funciones específicas en cada uno de los locales en particular.

La arquitectura de software provee una descripción de alto nivel de los subsistemas que comprenden la arquitectura del sistema. Está atada a comprender cómo las mayores operaciones del negocio son soportadas. Ayuda a la planificación de recursos y la asignación de tareas, debido a que el trabajo de desarrollo puede ser particionado a través de los subsistemas y los esfuerzos de desarrollo individual pueden proceder en paralelo. Cuando los equipos de desarrollo están geográficamente dispersos puede minimizar el número de interacciones requeridas.

1.3 Componente

Un componente es un bloque de construcción reusable de software: una pieza de código encapsulada de una aplicación que puede ser combinada con otros componentes y con códigos adicionales para producir una aplicación específica. Los componentes pueden ser simples o complejos. No existe un acuerdo general sobre qué es o qué no es un componente. Ellos vienen en una variedad de formas y tamaños. Un componente puede ser muy pequeño, como lo es un GUI Widget[5] (por ejemplo, un botón), o éste puede implementar un complejo servicio de una aplicación, tal como una función para administrar la red.

Un componente de Software es un modelo de software basado en objetos que apunta a incrementar la eficiencia del desarrollo de software. La idea principal es romper las aplicaciones monolíticas utilizando componentes binarios que puedan ser desarrollados, distribuidos y actualizados de forma independiente. Para lograr esto se necesita suministrar mecanismos estándares que permitan la comunicación entre aplicaciones y componentes. Si los componentes pueden ser combinados para construir grandes aplicaciones de forma flexible e incremental.

El desarrollo de software basado en componentes mejora la productividad de los programadores y la calidad del software porque:

- Debe ser escrito menos código nuevo para producir el mismo resultado.
- Los componentes desarrollados en la línea de montaje deben estar mejor terminados y por tanto deben ser más confiables que los realizados de forma artesanal.

Una aplicación está compuesta por una colección de componentes. Las aplicaciones proveen un ambiente, Glue Code[6] o un esqueleto de código, dentro del cual los componentes son insertados. Los componentes interactúan con su ambiente y pudieran interactuar con el sistema operativo de la computadora sobre la cual ellos son ejecutados. Además, ellos interactúan unos con otros a través de su respectiva interfaz. Cambiar una interfaz de un componente pudiera requerir cambiar el componente que usa esta interfaz. Sin embargo, cambiar la implementación del componente no debería requerir cambiar a los clientes de ese componente.

1.3.1 Componente de Software

Un componente de software es un independiente paquete entregable de servicios de software, en el mayor sentido general. Philippe Kruchten of *Rational Software* cree que un componente no es trivial, que es casi independiente y una parte reemplazable de un sistema que satisface una función clara en el contexto de una arquitectura bien definida. Un componente es para conformar y proveer la realización física de un conjunto de interfaz. Por otro lado Szyperski (8), ve a un componente de software como una unidad de composición con interfaz especificadas contractualmente y un explícito contexto de dependencias. Un componente de software puede ser desarrollado independientemente y estar sujeto a composición de terceras partes. De esta manera, un componente de negocio representa la implementación de software de una operación autónoma de negocio o proceso.

1.3.2 Interfaz

Una interfaz resume el comportamiento y responsabilidades que algún componente que tendrá que adherirse a este si es para participar como parte de un ensamblaje que representa alguna aplicación. La interfaz especifica la API[7] usada por el componente cliente que demanda el servicio de otro componente, pero no especifica la implementación. Esta separación hace posible que se pueda reemplazar la implementación de un servicio sin que ocasione ningún cambio en la interfaz y agregar nuevos servicios sin cambiar la implementación existente (2).

1.4 Patrones de Software

Los patrones de software son una forma de resolver los problemas de la ingeniería de software que ha emergido de la comunidad de la programación orientada a objetos. Un patrón de diseño es una solución abstracta a un problema determinado. Hoy en día existen diversas categorías de patrones a través de disciplinas técnicas y no técnicas.

Los patrones en los últimos años se han convertido en solución por excelencia, constituyéndose como una importante técnica para construir software orientado a objetos. El valor principal de estos se encuentra en el aprovechamiento de la experiencia de los expertos que los han identificado y

documentado. El conocimiento de los patrones de software facilita la identificación de determinados problemas a resolver y a buscar una solución rápida y elegante; son una herramienta la cual, basándose en experiencias anteriores, resuelve ciertos problemas que son frecuentes permitiendo que los sistemas sean mucho más adaptables al cambio. Surgen en base a la Arquitectura en donde se presentan una serie de problemas de forma recurrente y la forma de atacarlos es similar.

Los patrones indican repetición, si algo no se repite probablemente no sea un patrón. Vale aclarar que la repetición no es la única característica importante. También se necesita probar que un patrón es adaptable y que es útil. La repetición es un concepto que se puede medir, es cuantificable, pero la adaptabilidad y la utilidad son conceptos cualitativos. Para mostrar la repetición, por ejemplo se puede tomar el criterio que se presente en por lo menos tres sistemas existentes; para mostrar la adaptabilidad habrá que explicar como el patrón es exitoso y por último para mostrar la utilidad se deberá explicar porque es exitoso y beneficioso.

Varios autores han listado las cualidades que son deseables en un patrón, a continuación se presenta un resumen de las listadas en el libro "*Christopher Alexander: an Introduction for Object-Oriented Designers*" de Doug Lea (9):

- Encapsulamiento y abstracción: cada patrón encapsula un problema bien definido y su solución en un dominio particular. Los patrones deberían de proporcionar límites claros que ayuden a cristalizar el entorno del problema y el entorno de la solución.
- Extensión y variabilidad: cada patrón debería ser abierto por extensión de tal forma que pueden aplicarse junto con otros patrones para solucionar problemas de mayor complejidad. Un patrón debería ser también capaz de realizar una variedad infinita de implementaciones (de forma individual, y también en conjunción con otros patrones).
- Generatividad y composición: cada patrón, una vez aplicado, genera un contexto resultante, el cual concuerda con el contexto inicial de uno o más de uno de los patrones del lenguaje. Está secuencia de patrones podría luego ser aplicada progresivamente para conseguir la generación de una solución completa. Los patrones se encuentran jerárquicamente relacionados, la mayoría admiten tanto una composición ascendente como una descendente dentro de la jerarquía.
- Equilibrio: cada patrón debe realizar algún tipo de balance entre sus efectos y restricciones.

1.4.1 Clasificación de los patrones

Existen diversas categorías en los que se clasifican los patrones de software basándose en sus niveles de abstracción. Riehle y Züllighoven (10) sugieren tres tipos de estos patrones:

Conceptual: patrones cuya forma es descrita por el uso de términos y conceptos de formas en el dominio de una aplicación. Tales patrones pueden exhibir un diseño complejo para una aplicación completa o subsistema.

Diseño: patrones cuya forma es descrita usando diseño de software construidos como objetos, clases o módulos. Tales patrones facilitan soluciones a problemas de diseño en general en un contexto en particular.

Programación: patrones cuya forma es descrita por el significado de construcciones en un lenguaje de programación específico. Estos patrones facilitan la reusabilidad de componentes tales como listas enlazadas y tablas hash.

Existen otras clasificaciones que se encuentran en el medio de las tres antes descritas, como son los patrones GoF[8]. Estos patrones tienen sus tres propias clasificaciones:

Creación: estos patrones se centran en la creación de objetos –como son las clases u otros objetos-.

Estructurales: patrones que utilizan herencia para componer un clases o la manera específica de ensamblar un objeto.

Comportamiento: usan la herencia para describir algoritmos y flujos de control o para describir como un grupo de objetos pueden cooperar entre sí para lograr mejorar la realización de una tarea que un objeto solo no podría realizar.

En el marco de la ingeniería de software, existen diferentes ámbitos donde se pueden aplicar patrones. La siguiente es una de las posibles clasificaciones (9):

- Arquitectura
- Negocios
- Análisis
- Diseño
- Procesos y Organizacionales
- Ambientes Distribuidos
- Idiomas

En el ámbito del presente trabajo vale abundar un poco más en dos tipos de patrones en específico, los arquitectónicos y los de diseño.

Patrones Arquitectónicos

Este tipo de patrones son aquellos que tratan con vistas de alto nivel de un sistema en el cual sus primitivos son componentes y conectores. Los patrones arquitectónicos son aproximadamente lo mismo que lo que se acostumbra a definir como *estilos arquitectónicos* y ambos términos se utilizan indistintamente en la literatura. F. Buschmann propone un conjunto de patrones en esta categoría (11):

Layers
Permite estructurar aplicaciones que se pueden descomponer en grupos de sub-tareas, donde cada grupo está en un determinado nivel de abstracción.

Pipes & Filtres
Provee una estructura para sistemas que procesan un flujo de datos. Cada etapa del proceso es encapsulada como un filtro. Los datos se pasan entre filtros adyacentes mediante “Pipes”. Re-combinando filtros se obtienen familias de sistemas relacionados.

Blackboard
Útil para sistemas en que no se conoce una solución o estrategia determinada. Varios subsistemas especializados ensamblan su conocimiento para construir una posible solución parcial.

Broker
Permite estructurar sistemas distribuidos desacoplados que interaccionan mediante invocación de servicios remotos. Un componente Broker es responsable de coordinar la comunicación, así como de transmitir resultados y excepciones.

Model-View-Controller
Divide una aplicación interactiva en 3 componentes. *Model* contiene la información y funcionalidad principal. *Views* muestran información al usuario. *Controllers* gestionan la entrada de usuario. Un

mecanismo de propagación de cambios asegura la consistencia entre el modelo y la interfaz de usuario.

Presentation-Abstraction-Control
Estructura una aplicación interactiva como una jerarquía de agentes que cooperan. Cada agente es responsable de un determinado aspecto de la funcionalidad y consta de tres componentes: Presentación, Abstracción y Control, que separan la interacción con el usuario de la funcionalidad central y la comunicación con otros agentes.

Microkernel
Separar un mínimo núcleo funcional de funcionalidad extendida y partes específicas del cliente. El *Microkernel* también sirve como punto donde engarzar estas piezas y coordinar su colaboración.

Reflection
Proporciona un mecanismo para cambiar la estructura y el comportamiento del sistema dinámicamente. La aplicación se divide en dos partes: un meta-nivel y un nivel-base. El meta-nivel hace al software autoconsciente.

Patrones de Diseño

Un patrón de este tipo identifica, abstrae y nombra los aspectos elementales de una estructura de diseño, donde los componentes, son las clases y objetos, y sus mecanismos de interacción son mensajes.

Cada patrón prescribe una estructura de clases, sus roles y colaboraciones, y una adecuada asignación de métodos para resolver un problema de diseño en una manera flexible y adaptable. La aplicación de los patrones en un diseño consiste en identificar el patrón que resuelve el problema de diseño encontrado y aplicar la solución abstracta prescrita por el patrón a dicho problema. Los patrones de diseño ayudan a elegir diseños alternativos que hacen un sistema reutilizable y evitan alternativas que comprometan la reutilización. El libro que ha popularizado los patrones de diseño es el conocido "*Design Patterns: Elements of Reusable Object Oriented Software*" (12). A continuación se listan los patrones de este catálogo, con una breve descripción del propósito de cada uno de ellos:

Abstract Factory

Proporciona una interfaz para crear familias de objetos relacionados sin especificar sus clases concretas.

Adapter

Convierte la interfaz de una clase en otra distinta, que es la que esperan los clientes.

Bridge

Desacopla una abstracción de su implementación, de manera que puedan variar de forma independiente.

Builder

Separa la construcción de un objeto complejo de su representación.

Chain of Responsibility

Evita acoplar el emisor de una petición a su receptor, al dar a más de un objeto la posibilidad de responder a la petición.

Command

Encapsula una petición en un objeto, permitiendo así entre otras cosas parametrizar a los clientes con distintas peticiones, encolar o llevar un registro de las mismas.

Composite

Combina objetos en estructuras de árboles para representar jerarquías de parte-todo.

Decorator

Añade dinámicamente nuevas responsabilidades a un objeto. Alternativa de la herencia.

Facade

Proporciona una interfaz unificada para un conjunto de interfaz de un subsistema.

Factory Method

Define una interfaz para crear un objeto, pero deja que sean las subclases las que decidan que clase instanciar.

Flyweight

Usa el compartimiento para permitir un gran número de objetos de grano fino de forma eficiente.

Interpreter

Dado un lenguaje, define una representación de su gramática junto con un intérprete que usa dicha representación para interpretar las sentencias del lenguaje.

Iterator

Proporciona un modo de acceder secuencialmente a los elementos de un objeto agregado sin exponer su representación interna.

Mediator

Define un objeto que encapsula cómo interactúan un conjunto de objetos.

Memento

Representa y externaliza el estado interno de un objeto sin violar su encapsulación.

Observer

Defina una dependencia de uno a muchos entre objetos, de forma que cuando un objeto cambia de estado, se notifican y se actualizan automáticamente todos los objetos que dependen de él.

Prototype

Especifica los tipos de objetos a crear por medio de una instancia prototípica, y crea nuevos objetos copiando este prototipo.

Proxy

Proporciona un sustituto o representante de otro objeto para controlar el acceso a este.

Singleton

Garantiza que una clase solo tenga una instancia y proporciona un punto de acceso global a la misma.

State

Permite que un objeto modifique su comportamiento cada vez que cambia su estado interno.

Strategy

Define una familia de algoritmos, encapsula cada uno de ellos y los hace intercambiables.

Template Method

Define en una operación el esqueleto de un algoritmo delegando en las subclases algunos de sus pasos.

Visitor

Representa una operación sobre los elementos de una estructura de objetos. Permite definir una nueva operación sin cambiar las clases de los elementos sobre los que opera.

1.4.2 Beneficio del uso de Patrones.

Los patrones de software hacen más fácil reutilizar con éxito los diseños y arquitecturas. Expresando estas técnicas verificadas como patrones se hacen más accesibles para los diseñadores de nuevos sistemas. Los patrones ayudan a elegir diseños variados que hacen un sistema reutilizable y evitan alternativas que comprometan la reutilización. Los patrones de software pueden incluso mejorar la documentación y mantenimiento de sistemas existentes. Es decir, ayudan a un diseñador a conseguir un diseño correcto rápidamente.

Costos, satisfacción del cliente, productividad y reducción de los tiempos de desarrollo, son tal vez los puntos más críticos en todo desarrollo de software. Los patrones contribuyen indirectamente en cada uno de ellos:

Productividad: La productividad de un diseñador novato se ve mejorada por el uso de patrones, al proveer experiencia en el dominio, los patrones acortan el circuito de descubrimiento interno

para muchas importantes estructuras de diseño. Pero más importante aun es hecho que el uso de patrones evita la reconstrucción, ocasionada por inexpertas decisiones de diseño.

Reducción de los tiempos de desarrollo: Los patrones de software permiten la reutilización de soluciones de diseño, lo que ocasiona la reducción del tiempo requerido para la etapa de diseño, ya que los desarrolladores al utilizar un patrón solo deben adaptarlo para el caso y no pensar una solución desde cero.

Costos: La reducción de costos es proporcional a los dos puntos anteriores.

Satisfacción del cliente: Es mayormente el resultado de los factores anteriores.

En general los patrones de software:

1. Ayudan a construir la experiencia colectiva de Ingeniería de Software.
2. Son una abstracción de "problema – solución".
3. Se ocupan de problemas recurrentes.
4. Identifican y especifican abstracciones de niveles más altos que componentes o clases individuales.
5. Proporcionan vocabulario y entendimiento común.

1.5 Frameworks

Se han realizado importantes progresos orientados a la reusabilidad del software a través de la aplicación del paradigma de la programación orientada a objetos y mediante el uso de los componentes tecnológicos. Sin embargo, estas tecnologías solo proveen el re-uso en el nivel individual, frecuentemente a menor escala. Con la aparición de los patrones de diseño se ha demostrado la reutilización de soluciones para resolver problemas de escala mayor enfocados en la solución de problemas sencillos. Sin embargo, el más complejo problema de la reutilización de soluciones es en el nivel de los grandes componentes, para que estos se puedan adaptar para las solicitudes individuales.

Recientemente, el interés por la reutilización de software ha pasado de la reutilización de los componentes de forma independiente a la de todo el diseño de un sistema o la estructura de una aplicación. Un sistema de software que es capaz de permitir la reusabilidad a este nivel, permitiendo la realización de aplicaciones completas es llamado Framework. Un Framework debe estar basado en la

idea de facilitar la producción de un conjunto de sistemas específicos pero similares, dentro de una estructura genérica. Brevemente, los Frameworks arquitectónicos son arquitecturas genéricas integradas por un extensible conjunto de componentes. Además, los Frameworks pueden contener sub Frameworks los cuales representen subconjuntos de componentes de un sistema más grande.

1.5.1 Definición de Framework

Un Framework es un conjunto de clases o estructuras que implementan los componentes de una aplicación genérica, así como también componentes concretos que cumplen a cabalidad tareas concretas. Para desarrollar programas completos, los desarrolladores buscan e instancian los componentes apropiados.

Realmente no existe una definición oficial de Framework, pero todos los autores coinciden en la utilización de un tema común: la reutilización. Una definición dada por R. E. Johnson, and B. Foote en 1988 en su publicación “Designing Reusable Classes (13):
"*Un Framework es un conjunto de clases que personifican un diseño abstracto para soluciones de una familia de problemas relacionados*..."

Otro punto de vista, según R. E. Johnson y V. F. Russo en “Reusing Object-Oriented Designs” incluyen (14):

“Una clase abstracta es un diseño para un objeto simple. Un Framework es el diseño de un conjunto de objetos que colaboran para llevar a cabo un conjunto de responsabilidades. De esta manera los Frameworks son diseños a escalas más grandes que las clases abstractas. Los Frameworks son una forma de reutilizar el diseño a un nivel superior.”

Entre las demás definiciones encontradas de Framework para todas queda claro que el propósito general de crear y utilizar un Framework es ayudar a la reutilización de los distintos componentes prediseñados centrándose en brindar solución a un sistema en general.

1.6 Frameworks para PHP

Desde el momento que la programación en el lenguaje PHP ha comenzado a tomar prestigio y a propagarse entre las aplicaciones Web empresariales a nivel mundial, se ha provisto a PHP del enfoque de la Programación Orientada a Objetos de forma eficiente en el manejo de los objetos y la seguridad de estos –a partir de la aparición de PHP 5-. A medida que sus implementadores le han proporcionado al lenguaje más funcionalidades y seguridad se ha comenzado a pensar en una forma más fácil, rápida y robusta de crear aplicaciones. Nada mejor para esto que la creación de Frameworks, soluciones ya probadas en otros lenguajes como Ruby[9], Python[10], Perl[11] y Java[12], que permiten la creación de aplicaciones a partir de herramientas y librerías ya prediseñadas y que permiten ser reutilizar de forma eficiente.

Existen actualmente aproximadamente 57 Frameworks implementados en PHP que utilizan una arquitectura MVC que pudieran ayudar a la implementación de un marco de trabajo arquitectónico. En este trabajo se presentan como los más relevantes (dado sus características y la presencia en internet de aplicaciones implementadas por ellos, ver Anexo II CakePHP, Kumbia, Zend Framework y Symfony. A continuación se describen cada uno de ellos de manera general.

1.6.1 CakePHP

Es un Framework implementado para PHP 4 y PHP 5, está patentado bajo la licencia del MTI[13], brinda soporte para los SBD MySQL, PostgreSQL, SQlite, MS SQL, además de otros soportes como ADOdb o PEAR::DB[14].

Según sus creadores "*Cake es un rápido Framework de desarrollado para PHP que normalmente usa algunos de los más conocidos patrones de diseño como ActiveReccord, Mapeo de Datos, Controlador Frontal y MVC. Nuestro principal objetivos es suministrar la estructura de un Framework que habilite a los usuarios de PHP a todos los niveles aplicaciones web rápidas y robustas sin que pierda la flexibilidad*". (15)

Algunos de los factores positivos de utilizar este Framework son los siguientes:

- Es ligero, no tiene sobre peso de código fuente, solo el necesario para la aplicación.
- Corre en PHP4 y PHP5 como ya se mencionó.
- No necesita configuración –solo algunas pequeñas configuraciones en los archivos de configuración de la base de datos y algunas pocas constantes las cuales pueden ser modificadas- se puede comenzar la implementación de la aplicación literalmente en menos de cinco minutos luego de su instalación.
- Soporte para asociaciones entre tablas extendidas, permitiendo la creación de una arquitectura de la base de datos bastante compleja.
- Una estructura de directorio sumamente lógico y funcional –para algunos autores, mejor que Rails[15].
- Mejorado soporte para AJAX[16], a través de la vista de HELPERs[17]s de AJAX y Javascript[18].
- Cuenta con una comunidad muy activa y bastantes sitios de ayuda.

Otras características que presenta el Cake son las URLs limpias y amigables, debido al despachador de peticiones con el que cuenta. Presenta sistemas de validación a los largo de todo el Framework, verificación de ingreso de datos permitidos (*Data Sanitization*): permite determinar qué datos pueden ser ingresados y darle el formato adecuado a aquellos que no cumplen las reglas de validación. Generación de plantillas de manera rápida y flexible: usando la sintaxis de PHP y con asistentes o HELPERs. Incorporación de asistentes de construcción de vistas: para la automatización de la generación de código en AJAX, Java Script, formularios HTML, entre otros. Componentes de seguridad, manejo de sesiones y de peticiones: que reúnen las mejoras prácticas estandarizadas por la industria del software. Listas de control de acceso flexibles: para gestionar el ingreso de usuarios a la aplicación construida con el Framework. Almacenamiento en caché de las vistas: para acelerar la descarga de las páginas web. Trabaja en cualquier subdirectorio de un servidor web: requiere poca o nula configuración del servidor Apache donde se instalará. CakePHP es un Framework adecuado para todo tipo de aplicaciones web, desde pequeñas y personales hasta las empresariales y más complejas.

Algunos de los sitios que utilizan CakePHP son:

- Mozilla Addons (http://addons.mozilla.org/)
- Scratch by MIT (http://scratch.mit.edu/)
- Yale Daily News (http://www.yaledailynews.com/)
- The Onion Store (http://store.theonion.com/)

- NoseRub (http://noserub.com/)
- foamee (http://foamee.com/)
- twimbler (http://twimbler.com/)
- The Bakery (http://bakery.cakephp.org/)
- Island Cruises (http://www.islandcruises.com/)
- Fileshifter (http://www.fileshifter.se/en/index.html)
- lov.li (http://lov.li/)
- dishola (http://www.dishola.com/)
- CheeseCake Photoblog (http://cheesecake-photoblog.org/)
- SlideShowPro Director (http://www.slideshowpro.net/)
- PokerInside (http://www.pokerinside.com/)
- basil (http://www.getbasil.com/)
- mingle2 (http://mingle2.com/)

CakePHP en estos momentos goza en el mundo de un gran prestigio y se encuentra compitiendo entre los primeros cinco Frameworks de PHP más usados, no obstante debido a algunos de los inconvenientes que presenta su utilización no es conveniente su utilización en el diseño arquitectónico del Polo. Las causas principales son:

- No presenta un soporte oficial de internacionalización aunque será incluido en próximas versiones.
- No presenta soporte para Base de Datos Oracle, en ninguna de sus versiones.
- No presenta soporte para envío de correos ni generación de contenido sindicalizado (RSS[19]).
- No tiene almacenamiento de logs para el mantenimiento del sistema ni mantiene en la caché[20] la configuración de las aplicaciones.
- No utiliza completamente las ventajas que brinda PHP5, ya que lo que es una ventaja al ser usado en PHP4 lo amarra a la migración y utilización de las mejoras y comodidades que brinda PHP5.
- La documentación oficial todavía necesita algunas mejoras, aunque ahora parece estar bastante completa y exhaustiva.

1.6.2 Kumbia

Kumbia es un web Framework escrito en PHP5. Está patentizado bajo licencia GNU/GPL. Actualmente contiene adaptadores para manejar las conexiones a los siguientes gestores de base de datos: MySQL, PostgreSQL (BETA[21]) y Oracle (BETA).

Kumbia está basado en las mejores prácticas de desarrollo web, usado en software comercial y educativo, fomenta la velocidad y eficiencia en la creación y mantenimiento de aplicaciones web, reemplazando tareas de codificación repetitivas por poder, control y placer. A pesar de ser un Framework de creación muy reciente es un gran esfuerzo pues combina excelentes prácticas de programación y patrones de diseño, además de ser el Framework de la comunidad en Español mejor elaborado y documentado hasta el momento.

Entre sus funcionalidades más relevantes presenta las siguientes características:

- Sistema de Plantillas sencillo
- Administración de Cache
- Scaffolding[22] Avanzado
- Modelo de Objetos y Separación MVC
- Soporte para AJAX
- Generación de Formularios
- Componentes Gráficos
- Seguridad

Además de esto Kumbia cuenta con una capa de Abstracción de los Datos, implementada bajo el patrón de diseño *ActiveRecord*, esto le da una gran flexibilidad a la hora de establecer la comunicación entre el negocio y la parte física de los datos. La implementación de este patrón en Kumbia permite (16):

- Trabajar las entidades del Modelo más Naturalmente como objetos.
- Las acciones como Insertar, Consultar, Actualizar, Borrar, etc. de una entidad del Modelo están encapsuladas así que se reduce el código y se hace más fácil de mantener.
- Código más fácil de Entender y Mantener.

- Reducción del uso del SQL en un 80%, con lo que se logra un alto porcentaje de independencia del motor de base de datos.
- Menos “detalles” más practicidad y utilidad.

ActiveRecord protege en un gran porcentaje de ataques de inyección SQL que puedan llegar a sufrir las aplicaciones escapando caracteres que puedan permitir estos ataques.

Además de estas características, Kumbia integra otras prácticas y herramientas de los demás Frameworks, como la utilización de HELPERs, generación de reportes en PDF, WORD, EXCEL y Formularios Web a partir de las lecturas realizadas a la Base de Datos, eventos de llamadas del lado del cliente y del servidor (callbacks[23]), sistemas de trazas y Debugging desde la pantalla del navegador para los desarrolladores, organización del código en paquetes (namespaces[24]), soporte para AJAX, Prototype[25] y JSON[26], además de traer una integración con el Framework para JavaScript, Script.Aculo.Us[27], entre otras utilidades.

A pesar de todas las ventajas que ofrece este Framework y de tener una muy activa comunidad en español, por su origen latinoamericano, la idea de que los adaptadores para conectarse a las Bases de Datos PostgreSQL y Oracle estén en una fase BETA de su realización, deja de ser el favorito para la utilización en una arquitectura que da soporte a aplicaciones empresariales donde intervienen procesos de tanta importancia como los aduanales. Además de que el patrón ActiveRecord[28] tiene sus limitantes a la hora de trabajar modelos de datos demasiado grandes y complejos, como se acostumbran a presentar en las aplicaciones del perfil de gestión desarrolladas por el Polo Productivo Sistemas Tributarios y de Aduana. Esta debilidad está enmarcada a la hora de realizar consultas donde intervienen relaciones a varias tablas (binarias, ternarias, etc.), ya que no existe una clase que maneje este tipo operaciones, y las consultas y búsquedas deben ser realizadas a partir de un grupo de conexiones a la base de datos para pedir los datos de manera separada. Esta taxativa restricción hace que el Framework no sea apto para el desarrollo de aplicaciones con las características descritas.

1.6.3 Zend Framework

Zend Framework, es un intento dirigido por la compañía responsable del desarrollo del lenguaje PHP, Zend Technologies Ltd. Bajo la licencia BSD[29], y mantenido por una comunidad de voluntarios. A pesar de ello, a la fecha aún no consigue niveles de eficacia y adopción similares a los Frameworks en PHP ya existentes. Zend Framework destaca el hecho de que no sólo busca facilitar la programación a través del patrón MVC, sino también automatizar tareas más específicas, como el acceso a base de datos, el filtrado de datos ingresados a la aplicación o la búsqueda en un sitio web ordenando resultados por relevancia.

Entre sus metas se encuentran:

- Proveer un repositorio de componentes de alta calidad y que cuenten con soporte activo.
- Proveer un sistema completo para el desarrollo de aplicaciones web elaboradas en PHP5.
- Facilitar el aprendizaje en el uso del Framework sin tener que aprender un nuevo lenguaje de programación.
- Organizar la colaboración de la comunidad para una programación avanzada en PHP5.

Los componentes con los que cuenta este Framework pueden ser agrupados en las siguientes categorías:

- Infraestructura del núcleo del Framework: componentes requeridos por otros bloques del Framework, como memoria caché, configuración del ambiente de trabajo, operación por línea de comandos, registro de actividades y gestión de memoria, entre otros.
- Autentificación y autorización de acceso: responsables de la configuración de listas de control de acceso, autentificación de usuarios y manejo de sesiones.
- Base de datos: clases de acceso, modificación de tablas, obtención de datos mediante consultas SQL y lectura en formato XML.
- Internacionalización y Localización: para configurar la fecha y hora, la ubicación geográfica que tomará como base la aplicación web, las unidades de medida a usar y la posibilidad de traducir la información a otros idiomas.
- Emails, formatos y búsquedas: generación de archivos PDF, mensajes de correo electrónico en formato de texto y MIME (Multi-Purpose Internet Mail Extensions, Extensiones de correo Internet multipropósito)

- Modelo-Vista-Controlador: centra su atención en el desarrollo de controladores genéricos y específicos: de acciones, de atención de peticiones, de generación de URLs; además de una clase para las vistas.
- Servicios Web: permite que la aplicación web pueda hacer uso de servicios web ofrecidos por aplicaciones externas y exponga servicios web propios.
- Documentación: intenta facilitar la lectura del código fuente, brindar ejemplos de su uso, promover equipos de traducción de la documentación y elaborar un tutorial del desarrollo de una aplicación web. (17)

Sin embargo, a pesar de estar siendo desarrollado por el gigante de las tecnologías PHP, Zend Technologies Ltd. Este Framework se queda por debajo en un grupo importante de utilidades en cuanto a el resto de los más importantes Frameworks como Symfony y CakePHP. Entre estas faltas resaltan que Zend Framework no presenta una capa de abstracción de Acceso a Datos, que apruebe a realizar CRUD[30] u ORM[31] –aunque esto no quiere decir que no pueda integrarse con otros Frameworks específicos de Acceso a Datos como Propel o Doctrine[32]-. No permite realizar una estructura por defecto para sus aplicaciones (Scaffolding). No tiene soporte para la utilización de motores de plantillas como Smarty ni la implementación de ayudantes de plantillas HELPERs. No permite la verificación de la salida generada en HTML por procesamiento de peticiones (Data Sanitization). No admite almacenamiento en caché de las vistas ni configuraciones de la aplicación. No brinda la posibilidad de generación de código PHP, ni la utilización de una interfaz de líneas de comandos para la creación y mantenimiento de las aplicaciones, además de no mantener un almacenamiento de Logs de funcionamiento del Framework. Otro pronunciado defecto es la no posibilidad de integrarse con otras herramientas ni ampliar sus funcionalidades mediante la utilización de plugins. Otro aspecto a tener en cuenta en la valoración del Zend Framework es que la utilización del mismo requiere de PHP 5.2 o superior.

1.6.4 ERPFAR

En la UCI se han realizado varios esfuerzos para definir marcos de trabajo arquitectónicos en aplicaciones Web sobre tecnología PHP, se puede mencionar la implementación del ERPFAR en la Facultad 4, como la solución más cercana a los estándares internacionales en el desarrollo de Frameworks, al presentar un sistema bastante amplio que contiene una arquitectura robusta. Este

marco de trabajo se forma con los componentes de interfaz de usuario y estilo de aplicaciones, los componentes de lógica de negocio y acceso a datos principalmente, y brinda un soporte para las distintas aplicaciones que implementa dicho proyecto.

Está implementado utilizando el estilo arquitectónico en capas, cada una de las estas claramente definida. Aunque de manera general se trabaja Orientado a Objetos, la comunicación entre la capa de presentación y el negocio es mediante XML y arreglos de datos.

En la capa de Acceso a Datos se implementa patrones de diseño como *Table Data Gateway*[33], *Row Data Gateway*[34] y *Factory.* El acceso a la base de datos se realiza a partir de capa de la abstracción implementada en PDO[35], sin embargo las consultas y operaciones sobre la base de datos se realizan en código duro, específico para base de datos PostgreSQL, por lo que la utilización de este marco de trabajo para otro tipo de gestor se hace engorroso debido a la complejidad de su implementación y la fuerte acoplamiento que tiene con el referido gestor de base de datos. Además esta arquitectura no se ajusta a las restricciones para el desarrollo en el Polo.

1.7 Symfony

Symfony es un Framework para PHP5 patentado bajo licencia MTI, es compatible con la mayoría de gestores de bases de datos, MySQL, PostgreSQL, Oracle y SQL Server de Microsoft, entre otros en dependencia del tipo de abstracción de la Base de Datos que se utilice.

Este Framework simplifica el desarrollo de una aplicación mediante la automatización de algunos de los patrones utilizados para resolver las tareas comunes. Además proporciona una estructura al código fuente, forzando al desarrollador a crear código más legible y más fácil de mantener. Por último, facilita la programación de aplicaciones, ya que encapsula operaciones complejas en instrucciones sencillas.

Symfony está completamente diseñado para optimizar, gracias a sus características, el desarrollo de las aplicaciones web. Proporciona varias herramientas y clases encaminadas a reducir el tiempo de

desarrollo de una aplicación web compleja. Automatiza las tareas más comunes, permitiendo al desarrollador dedicarse por completo a los aspectos específicos de cada aplicación. Symfony está desarrollado completamente con PHP 5. Se puede ejecutar tanto en plataformas *nix (Unix, Linux, etc.) como en plataformas Windows.

Algunos de los sitios que utilizan Symfony, solamente en los Estados Unidos de América, son:

- PrimoCart(http://primocart.com/)
- Kevo(http://www.kevo.com/)
- Varsity Management(http://www.varsitymanagement.com/)
- George P. Mann and AssociatesStereo(http://greencard-us.com/)
- iStore(http://www.jakesmall.com/)
- National Digs(http://www.nationaldigs.com/)
- ThemBid.com(http://www.thembid.com/)
- GentleGiantLtd.comInfernal Machinery(http://www.infernalmachinery.com/)
- TED.com(http://www.ted.com/)
- AggregaPandora(http://www.pandora.com/)
- W3CounterAwio Web Services LLC()http://www.awio.com/
- SweetTunes(http://www.sweettunes.com/)
- TakeLessons.com(http://takelessons.com/)
- Find&GoSeekBluehouse Group(http://www.findandgoseek.net/)
- ibeatyou(http://www.ibeatyou.com/)
- Affordable Resume Services(http://www.affordableresumeservices.com/)
- Yahoo! Bookmarks(http://bookmarks.yahoo.com/)
- Varsity Management(http://www.varsitymanagement.com/)

Symfony es utilizado en el resto del mundo por miles de sitios, una lista más actualizada se puede encontrar en su sitio oficial, (18).

Symfony cuenta con 9 años de experiencia de sus desarrolladores y está inspirado en los mejores Frameworks de su tiempo, como Mojavi[36], Propel, Prado y Ruby on Rails, y utilizando lo mejor de cada uno de ellos. En más detalle Symfony se basa en:

- **Ruby on Rails**: sistema de enrutamiento, HELPERs, scaffolding, archivos de configuración en formato YAML[37] y la herramienta Rake[38] (que en Symfony se llama Pake).

- **Django**[39]: el nuevo mecanismo de formularios y widgets que incorpora la versión 1.1 de Symfony.
- **Propel** y **Doctrine**: son los dos ORM principales de Symfony, sobre todo Propel, que se encuentra completamente integrado.
- **Prado**[40]: todo lo relacionado con la internacionalización (i18n) y la localización (l10n).
- **Cocoa**[41]: el nuevo sistema de eventos de Symfony 1.1 se basa completamente en este Framework de Apple[42].
- **Test::More**[43]: La herramienta Lime[44], para crear pruebas unitarias, está basada en el Framework Test::More de Perl.
- **Prototype**, script.aculo.us, TinyMCE[45]: utilidades relacionadas con AJAX y JavaScript.

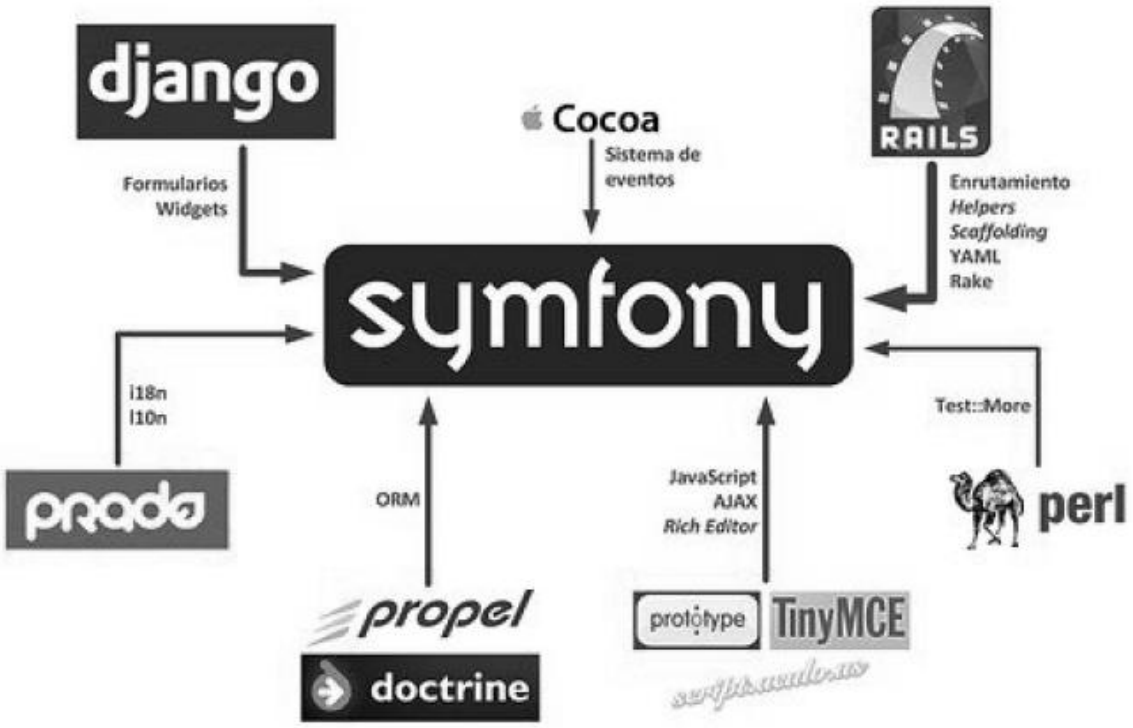

Fig. # 3: Composición de la estructura de Symfony por otros Frameworks.

Construido específicamente para Sitios web Empresariales, de necesidades complejas y para mejorar los procesos y las prácticas de desarrollo. Los tres pilares de su desarrollo son:

1. Reunir las empresas mundiales y el mundo del código abierto[46].
2. Desarrollo rápido.
3. No reinventar la rueda.

Symfony brinda muchas facilidades, entre ellas:

- Fácil de instalar y configurar en la mayoría de las plataformas (y con la garantía de que funciona correctamente en los sistemas Windows y *nix estándares) y por diferentes vías, entre las que se encuentran PEAR y SVN[47].
- Sencillo de usar en la mayoría de casos, pero lo suficientemente flexible como para adaptarse a los casos más complejos
- Código fácil de leer que incluye comentarios de phpDocumentor[48] y que permite un mantenimiento muy sencillo
- Fácil de extender, lo que permite su integración con librerías desarrolladas por terceros además de la utilización de HELPERs que permiten una integración con código para HTML, Javascript, AJAX y JSON.
- Posee scripts que facilitan y automatizan muchas de las tareas que realiza el Frameworks desde cualquier CLI[49].
- La interfaz de línea de comandos automatiza la instalación de las aplicaciones entre servidores.
- Posee una amplia documentación en español y una comunidad muy activa en este lenguaje.
- Los listados son más fáciles de utilizar debido a la paginación automatizada, el filtrado y el ordenamiento de datos.

Symfony implementa características que amplían su usabilidad y flexibilidad:

- Los plugins, las factorías (patrón de diseño “Factory”) y los “mixin[50]” permiten realizar extensiones a medida de Symfony.
- Excelente mecanismo de configuración mediante ficheros YML y XML.
- Alto desempeño en aplicaciones empresariales permitiendo optimizar la caché de la aplicación.
- Diseñado para aplicaciones empresariales, y adaptable a las políticas y arquitecturas propias de cada empresa, además de ser lo suficientemente estable como para desarrollar aplicaciones a largo plazo.
- La autenticación y la gestión de credenciales simplifican la creación de secciones restringidas y la gestión de la seguridad de usuario.
- Independiente del sistema gestor de bases de datos.
- El soporte de e-mail incluido y la gestión de APIs permiten a las aplicaciones web interactuar más allá de los navegadores.

- La capa de internacionalización que incluye Symfony permite la traducción de los datos y de la interfaz, así como la adaptación local de los contenidos

En eficiencia y rendimiento:

- Sigue la mayoría de *mejores prácticas* y patrones de diseño para la web.
- La capa de presentación utiliza plantillas y Layouts[51] que pueden ser creados por diseñadores HTML sin ningún tipo de conocimiento del Frameworks. Los HELPERs incluidos permiten minimizar el código utilizado en la presentación, ya que encapsulan grandes bloques de código en llamadas simples a funciones.
- Los formularios incluyen validación automatizada y relleno automático de datos ("repopulation"), lo que asegura la obtención de datos correctos y mejora la experiencia de usuario.

Cuenta con un grupo de herramientas integradas que permiten realizar un grupo de tareas para alcanzar una mayor eficiencia en el trabajo de la aplicación:

- Administración de proyectos.
- Generación automática de código.
- CRUD, Admin Generator[52], Data Access Objects[53].
- Testeo funcional y por unidades.
- Implementación (rsync[54], producción).
- Seguridad (Protección XSS[55] y CSRF[56] por defecto).
- Plugins.

1.7.1 ¿Por qué Symfony?

En estos momentos la meta fundamental a alcanzar por el Polo de Sistemas Tributarios y de Aduanas es la de poder realizar la mayor cantidad de aplicaciones en el menor tiempo posible, de manera profesional y competitiva, dando respuesta a las necesidades de los clientes, garantizando su satisfacción y al mismo tiempo manteniendo la calidad, usabilidad y eficiencia de los productos. Realizar esto desde aplicaciones implementadas desde cero es algo prácticamente imposible, por lo que es inminente la utilización de un Framework de entre los que existen en el ámbito de la

programación en PHP. Debido a esto, se une la realidad de que las aplicaciones de gestión realizadas por el PSTA, deben ser cada día más complejas y profesionales.

Una de las restricciones más limitantes en esta arquitectura surge por la necesidad del cliente de mantener funcionales las Bases de Datos Oracle 8i. Producto de esto es necesario utilizar la versión 5.0.4 de PHP, la última de las versiones de este lenguaje que poseían bibliotecas de conexión para este tipo de Base de Datos. Implementar el acceso a los datos a servidores Oracle 8i con versiones superiores al PHP 5.0.4 implica recompilado del PHP de las bibliotecas de conexión Oracle (oci8.dll y oracle.dll) lo cual añadiría un costo extra que debe ser considerado a la hora de tomar una decisión como esta. Esta restricción es temporal, ya que la transición de Oracle a la versión 11g se está llevando a cabo pero este tiempo está estimado en un año mínimo, y el tiempo de producción e implantación de los productos no puede detenerse durante este tiempo.

Otras de las características que deben presentar los sistemas desarrollados por el Polo, es su alto grado de flexibilidad y adaptabilidad a los distintos cambios propuestos por los clientes, lo que permita no tener que desarrollar aplicaciones específicas para cada uno de ellos, sino que se pueda realizar software empresariales a medida reutilizando los componentes que se vayan desarrollando a lo largo del tiempo de vida del proyecto. Permitir que las aplicaciones sean configurables durante el proceso de desarrollo disminuye las posibilidades de dejar huecos de seguridad y permite que estas se adapten mejor a cualquier tipo de entorno. Lo que tome un poco más de tiempo en aprender a realizar, puede garantizar elevar la eficiencia del producto final.

Para facilitar y agilizar el desarrollo de aplicaciones es necesario que las herramientas utilizadas garanticen la generación de parte del código utilizado durante la implementación, esto agiliza el proceso de producción de software y evita la inserción de errores por parte de los desarrolladores, además de que garantiza que el código generado sea el más óptimo para cada uno de los casos. Durante el proceso de investigación se identifican tres tipos de código generados, generación de código para HTML y Javascript utilizado para crear componentes Web en la parte de Interfaz de Usuario; código PHP perteneciente a las clases, creado en un grupo de archivos y directorios durante la creación de aplicaciones, junto con esto pueden aparecer otros tipos de archivos como configuración; y por último el código SQL generado para distintos gestores de Bases de Datos que permitan optimizar las operaciones realizadas sobre la misma.

Otro requisito que se impone para la selección de un marco arquitectónico referencial es la necesidad de que presente una buena documentación y personas con experiencia en su desarrollo que avalen los resultados óptimos obtenidos con el mismo. Para esto es necesario contar con ciertos elementos en ámbito de documentación como una amplia guía que sirva para conocer el funcionamiento de la herramienta al detalle, el API de las librerías utilizadas para conocer los procesos internos del núcleo y la mejor forma de utilizarlos, ejemplos de desarrollo de aplicaciones con el Framework y una comunidad activa tanto de desarrolladores como de entusiastas que puedan agilizar la necesidad de buscar soluciones a los posibles problemas de implementación que se puedan presentar.

Nuevas posibilidades que debe brindar el marco de trabajo a utilizar debe ser la posibilidad de integrarse con otros sistemas, ampliar sus funcionalidades mediantes las experiencias alcanzadas por otros desarrolladores en el mundo y permitir optimizar el código producido y probado como eficiente por terceros. No se puede ignorar la facilitación que debe brindar un Framework para poder utilizar tendencias actuales de la programación Web en PHP como AJAX, comunicación por JSON y XML, publicación de sus funcionalidades como Servicios Web, integración con los paradigmas de la WEB2.0, etc. Es importante también garantizar un mecanismo de internacionalización y localización para los usuarios que utilicen las aplicaciones sin necesidad de que los desarrolladores tengan que implementar nuevamente interfaz de usuarios en distintos idiomas para poder brindar estos servicios.

En estos momentos uno de las mejores soluciones para el desarrollo de aplicaciones web empresariales en PHP es Zend Framework, pero la necesidad de utilizar PHP 5.2 o superior hace que no cumpla con las restricciones necesarias del Polo, además de que la curva de aprendizaje en la utilización del Framework es muy alta, debido a lo complicado que es comenzar a trabajar con él. Además de que no brinda una arquitectura robusta para sus aplicaciones, ya que se compone de un grupo de librerías que brindan funcionalidades específicas.

CakePHP, tiene una gran popularidad a nivel mundial y un gran uso en aplicaciones robustas pero no de corte empresarial, más bien en aplicaciones comerciales. Esto no da la medida de que esté lo suficientemente probado para soportar las demandas de una aplicación de esta envergadura. A esto se le añade el inconveniente de no presentar soporte para Bases de Datos Oracle.

Kumbia es otro de los Frameworks que al parecer está subiendo con mucha fuerza, pero debido a la temprana edad del mismo y que aún no cuenta con el apoyo de la comunidad por la poca experiencia que brinda no es conveniente arriesgar la implementación de productos que deben brindar una alta estabilidad a los usuarios por su utilización. Además de que la capa del modelo que utiliza el Framework no permite adaptarla a diseños Bases de Datos complejos y con mucho volumen de datos.

En el estudio realizado por los Frameworks más importantes en este momento y que pueden facilitar la producción del Polo, se llega a la conclusión de que la decisión correcta para el desarrollo del polo en estos momentos es Symfony, ya que satisface todas las necesidades expresadas anteriormente de la siguiente manera:

- Symfony está diseñado para desarrollar aplicaciones empresariales. El autor del Framework aconseja que si lo que se desea es crear una aplicación sencilla de pocas páginas, entonces se utilice otros Frameworks.
- Hasta la versión 1.0.8 se puede utilizar la versión de PHP 5.0.4, la transición al PHP y de otras versiones del Framework superiores es de forma transparente para los desarrolladores y los usuarios.
- Symfony permite una configuración en cascada que permite que cada uno de sus elementos sean adaptables a las necesidades de los desarrolladores. Esto le da flexibilidad y fortalece al mismo tiempo a las aplicaciones.
- Las herramientas de generación de código brindadas implícitamente por el Framework permiten generar código tanto SQL como PHP de manera óptima, así como HTML y Javascript mediante la utilización de HELPERs.
- Presenta una amplia documentación tanto de su utilización como de su núcleo, además de mencionar la aceptación a nivel mundial y la tendencia a utilizarlo.
- Symfony trae una amplia gama de funcionalidades, pero quizás su mayor poder está en la posibilidad de ampliar estas de forma ilimitada mediante la utilización de plugins. Uno de los más actuales permite publicar las funciones desarrolladas como Servicios Web, algo que no se podía realizar hasta hace poco.
- Los HELPERs ayudan a integrarse muy fácilmente a los desarrolladores con las tecnologías de comunicación asincrónica como AJAX y JSON.
- Trae sistemas de traducción de interfaz de usuario de forma incluida y automática a partir de la selección cultural del usuario.

1.7.2 Propel

Propel es un Framework programado en PHP5 de persistencia de Datos y consulta, lo que significa que brinda un mecanismo para almacenar objetos PHP en una base de datos y un sistema para búsqueda y restauración de objetos PHP desde una base de datos, todo esto a partir de un mecanismo de abstracción de los datos y utilizando ORM. Propel permite realizar consultas complejas y manipulación de datos sin escribir una sola cláusula SQL. Hace más fácil la escritura de aplicaciones, el despliegue y mucho más migrar la aplicación a otro gestor de Base de Datos si alguna vez la situación lo amerita. La última versión estable es la 1.2 y corre sobre PHP 5.0.4.

Propel puede ser descrito como un mapeado objeto-relacional, una capa DAO, o una capa objeto persistente. Su implementación está basada principalmente en los patrones *Row Data Gateway* y *Table Data Gareway*.

Propel está dividido en dos componentes principales:

1. Un motor generador para construir sus clases y archivos SQL (*generador-propel*).
2. Un ambiente de ejecución que proporciona herramientas para construir consultas SQL, ejecutando consultas compiladas, y herramientas para el manejo de conexiones para múltiples bases de datos simultáneamente.

La parte de la generación se basa en la creación de un conjunto de clases que se corresponden con la estructura relacional de la Base de Datos, y permite trabajar con ellas a nivel de la aplicación realizando mediante ellas las operaciones CRUD correspondientes a cada una de las tablas. Todas estas clases son generadas de forma automática a partir de la lectura de un archivo XML, en el que está descrita la distribución de la Base de Datos. La otra parte de la aplicación está basada en la posibilidad de realizar cualquier tipo de consulta, por compleja que sea mediante el manejo de un grupo de clases que se encargan de ello. Estas clases son capaces de manejar relaciones entre tablas –incluso en relaciones entre tablas de muchos a muchos (*many to many*), lo que en comparación con implementaciones de otras capas de Acceso a Datos que utilizan el patrón ActiveRecord es algo

todavía ni siquiera imaginable-, de manera transparente al programador, como si se trataran de Objetos que son creados y contenidos por las clases.

El ambiente de ejecución proporciona una capa de abstracciones y encapsulación de bases de datos reglas lógicas de negocios. Las clases Propel representan la capa modelo del tradicional MVC, diseñado para encapsular cualquier nivel de validación de datos necesitado por cualquier aplicación. El siguiente diagrama ilustra como Propel existe en relación a Creole y las subyacentes bases de datos.

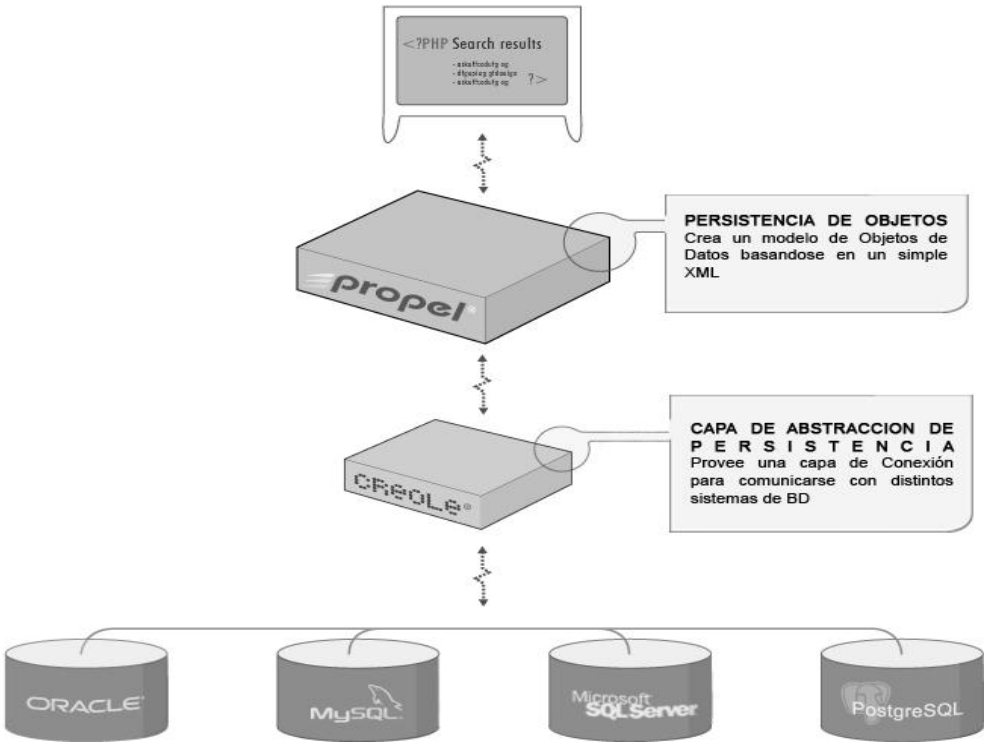

Fig. # 4: Estructura de funcionamiento de Propel

De manera resumida las ventajas de utilizar Propel pueden describirse en:

- Independencia de la aplicación con la Base de Datos
- Permite realizar consultas sobre la Base de Datos de Alta complejidad sin utilizar ningún código SQL.
- Implementación de los patrones de diseño *Data Row Gareway* y *Data Table Gareway*.
- Genera completamente el ORM de la Base de Datos a partir de un fichero de configuración XML.

- Realiza todas las operaciones sobre la BD de manera transparente para la lógica del negocio y de manera Orientada a Objetos.
- Realiza operaciones sobre tablas que se relacionan de mucho a mucho.

Propel viene integrado con el Framework Symfony, y este a su vez permite a los desarrolladores la opción de utilizarlo o no. En caso positivo tiene las herramientas necesarias para a partir de un motor de generación crear el archivo XML necesario para el ORM de la Base de Datos a partir de la lectura de la misma. Esta ventaja quita de encima la compleja tarea de tener que escribir el fichero de configuración a mano cada vez que sea necesario crear un nuevo modelo, haciendo mucho más fácil esta tarea.

Además de Propel, Symfony trabaja muy bien integrado con Doctrine, otro Framework de persistencia de datos muy similar a Propel. El inconveniente actual de utilizar Doctrine es que necesita de PHP 5.2.3+ para su correcto funcionamiento, pero la utilización de este para versiones futuras de la implementación de soluciones está en estudio en estos momentos, pues brinda nuevas prestaciones, al igual que las ya anunciada por la futura versión 1.3 de Propel. Además se debe tener en cuenta que el temprano nacimiento de Doctrine no le da la madurez necesaria para utilizarla en aplicaciones de alta envergadura.

1.8 Conclusiones

En este capítulo se presentaron los conceptos más significativos para la correcta comprensión de este trabajo. Se realizó un estudio en el que se demuestra que el desarrollo de aplicaciones Web hoy en día está regida por la utilización de Frameworks implementados bajo el patrón arquitectónico MVC que faciliten el diseño e implementación de los sistemas y el estudio realizado por los Frameworks para PHP más significantes demuestra que el que más se ajusta a las necesidades de las aplicaciones empresariales de gestión que requieran de Base de Datos Oracle 8i y PostgreSQL es Symfony utilizando Propel para manejar la persistencia de los datos.

Capítulo 2: PROPUESTA DE SOLUCIÓN

2.1 Introducción

En este capítulo se exponen los Requisitos No Funcionales especificados por el cliente que deben tenerse en cuenta a la hora de definir la solución arquitectónica para el desarrollo de las soluciones. Se exponen las características principales de la solución arquitectónica propuesta para el PSTA detallando los compontes a utilizar, los paradigmas en los que se basa la implementación de las soluciones, los patrones de software utilizados y la comunicación entre los componentes internos y entre las capas arquitectónicas.

2.2 Requerimientos No Funcionales

2.2.1 Usabilidad

Los sistemas desarrollados por el PSTA deben prestar facilidades de usabilidad que satisfagan las necesidades de los usuarios, deben contar con un menú que le permita a los usuarios acceder a las principales funciones que son de su interés. Así como brindar a los usuarios la posibilidad de interactuar con los productos sin tener previa preparación, solo con los conocimientos necesarios del negocio, en este aspecto no se incluye la parte administrativa de las aplicaciones que si requerirán de una preparación para su manipulación. La resolución de la página se adaptará la resolución de pantalla en el cliente.

2.2.2 Rendimiento

Los sistemas en general deben poseer un elevado nivel de eficiencia, teniendo en cuenta que las transacciones de datos pueden ser elevadas, así como el volumen de la Base de Datos. Por tanto se requiere del diseño, desarrollo de mecanismos y técnicas de programación óptimas para el lenguaje de programación seleccionado.

2.2.3 Fiabilidad

Disponibilidad: Las aplicaciones deben ser capaces de estar operativas durante el mayor tiempo posible para brindar sus servicios a los usuarios ininterrumpidamente durante el tiempo que estos lo necesiten. El tiempo para la mantención de las aplicaciones no debe exceder el 10% con respecto al tiempo que la aplicación debe estar disponibles.

- *Tiempo medio entre fallos*: El tiempo mínimo de disponibilidad ante posibles fallos será en dependencia de las especificaciones de cada sistema.
- *Tiempo medio de reparación*: La recuperación ante los fallos será en dependencia de las especificaciones de cada sistema.
- *Exactitud*: La exactitud de las aplicaciones no debe exceder 1 min.

2.2.4 Eficiencia

- Tiempo de respuesta por transacción 1.09 segundos.
- Rendimiento: 1000 transacciones por segundos, cantidad de datos que pueden ser transferidos.
- Capacidad 80 clientes que pueden estar conectados.
- Utilización de recursos 18 % memoria y 20 % en disco duro.

2.2.5 Soporte

- Las aplicaciones clientes deben ser capaz de correr sobre cualquier plataforma, para el caso de Windows se recomienda XP por la experiencia acumulada por los usuarios. Para la parte servidora se recomienda que corra sobre plataforma Linux.
- Ser programado en PHP 5.0.4 y con un gestor de base de datos Oracle 8i o superior, o PostgreSQL 8.x

2.2.6 Hardware

Cliente: Las aplicaciones son desarrolladas para que las PC clientes de los usuarios puedan usar las aplicaciones con el menor requerimiento posible. Estos requerimientos mínimos son:

- Procesador Pentium II o superior, 333 MHz mínimo pero recomendado 1.8 GHz.
- 10 GB o más de capacidad.
- Mínimo de memoria RAM 128 DIM
- Adaptador de Red y conectividad.

Se recomienda la utilización de clientes ligeros para aprovechar las facilidades de mantenimiento que esto brinda.

Servidor: Debido a que la cantidad de datos manejados por este tipo de aplicaciones se requiere separar el servidor de aplicación y el de Base de Datos. Cada uno de ellos requiere de características de Hardware específicas

Servidor de Aplicación:

- CPU = 2 Dual Core AMD Opteron 64 bits a 2.2 GHz
- RAM = 2 GB
- Almacenamiento = 1 Discos SATA de 73 GB en RAID 1

Servidor de Base de Datos:

- CPU = 2 Dual Core AMD Opteron 64 bits a 2.2 GHz
- RAM = 4 GB
- Almacenamiento = 2 Discos SATA de 73 GB en RAID 1

2.2.7 Software

Los las usuarios deben tener instalado como navegador el Mozilla Firefox 1.5, el Internet Explorer 6 o superior. Sistema Operativo Windows NT o Linux, para PC o MacOS para Macintosh. Nada de esto sería necesario de utilizarse los clientes ligeros.

El servidor de aplicación debe tener instalado PHP 5.x o superior, Servidor Web Apache 2.0 o superior. En la Base de Datos es necesario que esté instalado una versión de Oracle 8i o superior, o un PostgreSQL 8.x.

2.2.8 Portabilidad

La utilización de PHP facilita que los sistemas sean completamente portables y corran tanto en entornos Windows, como en entornos *nix.

2.2.9 Seguridad

Debido a que la información con la que se va a trabajar es confidencial los sistemas deben contar con las políticas de seguridad adecuadas, la misma debe estar basada en niveles de acceso a la información de acuerdo con el nivel jerárquico que tenga el usuario autorizado por los administradores y seguridad para la transmisión de los datos. Garantizar que las transacciones de datos se ejecuten por vías seguras, ya sea en la red o por soporte físico.

Como parte de las políticas de seguridad, todas las operaciones de inserción, alteración y borrado de los datos serán auditadas para su posterior análisis en caso de que sea necesario.

2.2.10 Estándares Aplicables

XHTML

XHTML, acrónimo inglés de e**X**tensible **H**ypertext **M**arkup **L**anguage (lenguaje extensible de marcado de hipertexto), es el lenguaje de marcado pensado para sustituir a HTML como estándar para las páginas web. XHTML es la versión XML de HTML, por lo que tiene, básicamente, las mismas funcionalidades, pero cumple las especificaciones, más estrictas, de XML. Su objetivo es avanzar en el proyecto del World Wide Web Consortium de lograr una web semántica, donde la información, y la forma de presentarla estén claramente separadas. En este sentido, XHTML serviría únicamente para transmitir la información que contiene un documento, dejando para hojas de estilo (como las hojas de estilo en cascada) su aspecto y diseño en distintos medios (computadoras, PDAs, teléfonos móviles, impresoras...) y para JavaScript su comportamiento.

HR-XML

Esquemas XML desarrollados por el Consorcio HR-XML Inc. para soportar una variedad de procesos de negocios relacionados con la administración de los recursos humanos. Estos incluyen esquemas para representar información referente a la nómina de pago, información sobre currículo laboral, información de formación, etc.

HR-XML Inc. es un consorcio independiente, sin fines de lucro que desarrolla libremente formas de inter-operatividad específicas para la gestión de los recursos humanos entre ordenadores. La mira de este trabajo es el desarrollo de especificaciones electrónicas para el intercambio entre proveedores de servicios de RH.

Estándares en Sistemas Aduanales

Existen un gran número de estándares de información para la transmisión de datos electrónicos referentes a los procesos aduanales. Estos se utilizan en distintas latitudes y en la gestión de diferentes procesos. Las aplicaciones desarrolladas por el PSTA deben estar a la altura de los sistemas que rigen la transmisión de información de procesos aduanales a nivel mundial, por esta razón cada solución según sus características específicas deben cumplir con los requerimientos de permitir generar, interpretar y transmitir la información necesaria por los medios establecidos mundialmente.

2.2.11 Reutilización de Componentes

Los sistemas realizados para los procesos aduanales incluyen el manejo de varios asuntos que se enlazan entre sí, por lo que constantemente se necesita acceder a información común para varios de estos procesos, es aquí donde entra la reutilización de dicha información, de forma tal que los datos no aparezcan duplicados.

2.3 Definición de la Solución Arquitectónica

2.3.1 Programación Orientada a Objetos

La implementación de una arquitectura MVC en un lenguaje de programación que no está orientado a objetos puede encontrarse con problemas de *namespaces* y código duplicado, dificultando la lectura del código de la aplicación.

Las posibilidades que ofrece la combinación del patrón MVC con los lenguajes de programación modernos para trabajar con objetos permiten simplificar la programación, ya que los objetos pueden encapsular la lógica, pueden heredar métodos y atributos entre diferentes objetos y proporcionan una serie de convenciones claras sobre la forma de nombrar a los objetos. La orientación a objetos permite a los desarrolladores trabajar con objetos de la vista, objetos del controlador y clases del modelo, transformando sus funciones en métodos. Se trata de un requisito obligatorio para las arquitecturas de tipo MVC.

2.3.2 Symfony, Framework Arquitectónico

La selección de un marco de trabajo arquitectónico depende de varios factores como se mostró durante la explicación de la Fundamentación Teórica en el Capítulo 1. Por lo que la utilización del Framework Symfony trae varias ventajas que no son posibles de igualar por ningún otro componente de este tipo y que además cumpla con los Requisitos No Funcionales.

2.3.3 Propel, Persistencia de los Datos

Propel es un proyecto independiente de Symfony, pero están fuertemente vinculados y la unión de ambos trae como consecuencia una excelente implementación del patrón MVC. Por lo que esta combinación alcanza una usabilidad óptima para el uso en aplicaciones de este tipo.

2.3.4 Ext JS, Presentación y Comunicación

Ext JS es un Framework de JavaScript que permite realizar aplicaciones Web enriquecidas basándose en tecnología AJAX, JSON, DHTML[57] y DOM[58]. Ext 2.0 está patentado bajo licencia LGPL[59] lo que posibilita su uso para aplicaciones empresariales privativas de código cerrado.

Ext JS brinda la posibilidad de utilizar un gran número de componentes visuales que mejoran considerablemente la calidad de las aplicaciones. Brinda la posibilidad de validaciones de formularios de todo tipo, basándose en expresiones regulares y tipos de datos. Trae implícitos componentes como vista en arboles, arrastrado y soltado, cambio de tamaño de imágenes, rejillas, paginado, agrupado de objetos, *tabs*[60], asistentes, entre otros muchos.

La utilización de un Framework de Javascript como Ext JS facilita la separación de las capas de la vista con la del controlador desde el punto de vista productivo ya que el código utilizado en la primera es solamente JavaScript y no es necesario utilizar ningún tipo de código PHP, así los desarrolladores pueden centrarse más en el aprendizaje de un solo lenguaje. Además al soportar serialización de objetos mediante tecnología JSON permite que los datos enviados desde el controlador como respuesta a la vista contengan solo las propiedades de dichos objetos, pero no el comportamiento, minimizando los posibles errores de programación y los accidentes de que los objetos sean modificados erróneamente desde la vista.

2.3.5 WEB 2.0, Tendencia de Desarrollo Web

La Web 2.0 es la representación de la evolución de las aplicaciones tradicionales hacia aplicaciones web enfocadas al usuario final. El Web 2.0 es una actitud y no precisamente una tecnología.

La Web 2.0 es la transición que se ha dado de aplicaciones tradicionales hacia aplicaciones que funcionan a través de la Web enfocadas al usuario final. Se trata de aplicaciones que generen colaboración y de servicios que reemplacen las aplicaciones de escritorio. El uso del término de Web 2.0 está de moda, dándole mucho peso a una tendencia que ha estado presente desde hace algún tiempo. En Internet las especulaciones han sido causantes de grandes burbujas tecnológicas y han hecho fracasar a muchos proyectos. El Web 2.0 no es precisamente una tecnología, sino es la actitud

con la que se debe trabajar para desarrollar en Internet. Tal vez allí está la reflexión más importante del Web 2.0.

Finalmente se puede definir la solución arquitectónica como la utilización del Framework Arquitectónico Symfony para lograr obtener como resultado un estilo Modelo-Vista-controlador, con Propel para manejar el modelo de la arquitectura mediante un Objetc-Relational-Model y Creole para manejar la abstracción de la Base de Datos. Para manejar la concepción de los paradigmas del WEB 2.0 y la seguridad de las aplicaciones se utiliza el Framework para Javascript Ext JS permitiendo la creación de efectos visuales y validaciones del lado del cliente.

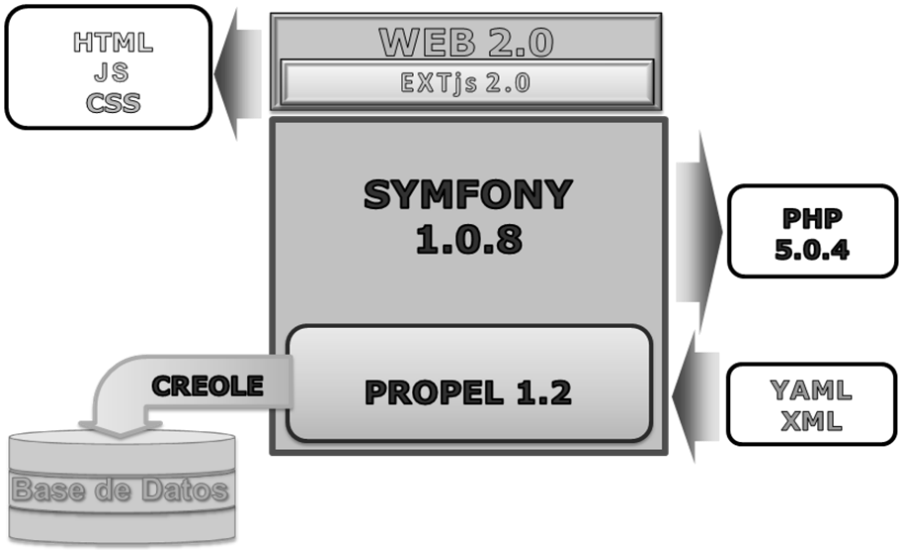

Fig. # 5: Integración de los Componentes y Tecnologías en la Solución Arquitectónica.

2.4 Patrones de Software

2.4.1 Arquitectónico

La selección del patrón arquitectónico utilizado se basa en la selección del Framework a utilizar, a continuación se da una pequeña descripción de la implementación descrita por el patrón Modelo Vista Controlador.

MVC

Es un patrón de diseño de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos. El patrón MVC se ve frecuentemente en aplicaciones Web, donde la vista es la página HTML y el código que provee de datos dinámicos a la página.

Modelo: Esta es la representación específica del dominio de la información sobre la cual funciona la aplicación. El modelo es otra forma de llamar a la capa de dominio. La lógica de dominio añade significado a los datos; por ejemplo, calculando si hoy es el cumpleaños del usuario o los totales, impuestos o portes en un carrito de la compra.
Vista: Esta presenta el modelo en un formato adecuado para interactuar, usualmente un elemento de la interfaz de usuario. Ejemplo un Formulario.
Controlador: Este responde a eventos, usualmente acciones del usuario e invoca cambios en el modelo y probablemente en la vista. Muchas aplicaciones utilizan un mecanismo de almacenamiento persistente (como puede ser una base de datos) para almacenar los datos.

MVC no menciona específicamente esta capa de acceso a datos. Es común pensar que una aplicación tiene tres capas principales: presentación (IU), dominio, y acceso a datos. En MVC, la capa de presentación está partida en controlador y vista. La principal separación es entre presentación y dominio; la separación entre la vista y el controlador es menos clara. Aunque se pueden encontrar diferentes implementaciones de MVC, el flujo que sigue el control generalmente es el siguiente:

El usuario interactúa con la interfaz de usuario de alguna forma (por ejemplo, el usuario pulsa un botón, enlace)

MVC no menciona específicamente esta capa de acceso a datos. Es común pensar que una aplicación tiene tres capas principales: presentación (IU), negocio y acceso a datos. En MVC, la capa de presentación está partida en controlador y vista. La principal separación es entre presentación y negocio; la separación entre MVC es menos clara. Aunque se pueden encontrar diferentes implementaciones de MVC, el flujo que sigue el control generalmente es el siguiente:

1. El usuario interactúa con la interfaz de usuario de alguna forma (por ejemplo, el usuario pulsa un botón, enlace)
2. El controlador recibe (por parte de los objetos de la interfaz-vista) la notificación de la acción solicitada por el usuario. El controlador gestiona el evento que llega, frecuentemente a través de un gestor de eventos (handler) o callback.

3. El controlador accede al modelo, actualizándolo, posiblemente modificándolo de forma adecuada a la acción solicitada por el usuario (por ejemplo, el controlador actualiza el carro de la compra del usuario). Los controladores complejos están a menudo estructurados usando un patrón de comando que encapsula las acciones y simplifica su extensión.
4. El controlador delega a los objetos de la vista la tarea de desplegar la interfaz de usuario. La vista obtiene sus datos del modelo para generar la interfaz apropiada para el usuario donde se refleja los cambios en el modelo (por ejemplo, produce un listado del contenido del carro de la compra). El modelo no debe tener conocimiento directo sobre la vista. Sin embargo, el patrón de observador puede ser utilizado para proveer cierta indirección entre el modelo y la vista, permitiendo al modelo notificar a los interesados de cualquier cambio. Un objeto vista puede registrarse con el modelo y esperar a los cambios, pero aun así el modelo en sí mismo sigue sin saber nada de la vista. El controlador no pasa objetos de dominio (el modelo) a la vista aunque puede dar la orden a la vista para que se actualice. Nota: En algunas implementaciones la vista no tiene acceso directo al modelo, dejando que el controlador envíe los datos del modelo a la vista.
5. La interfaz de usuario espera nuevas interacciones del usuario, comenzando el ciclo nuevamente.

El principio más importante de la arquitectura MVC es la separación del código del programa en tres capas, dependiendo de su naturaleza. La lógica relacionada con los datos se incluye en el modelo, el código de la presentación en la vista y la lógica de la aplicación en el controlador.

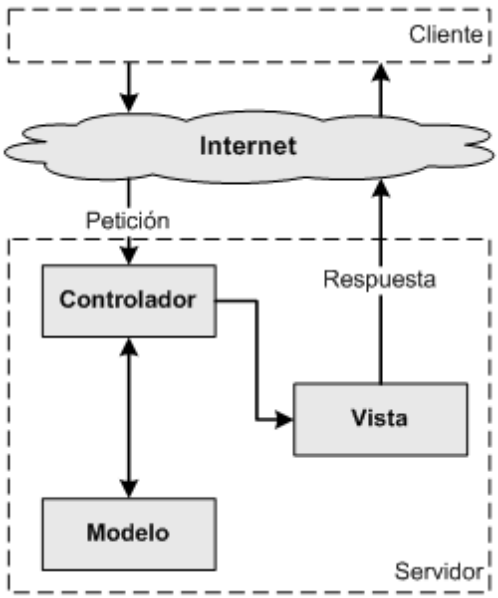

Fig. # 6: Separación de las capas lógicas del MVC.

La programación se puede simplificar si se utilizan otros patrones de diseño. De esta forma, las capas del modelo, la vista y el controlador se pueden subdividir en más capas.

2.4.2 Diseño

Algunos de los patrones de Diseño utilizados en la implementación de la solución arquitectónica son los descritos a continuación:

FACADE

Problema: ¿Como acceder a diferentes clases a través de una única clase?

Propósito: Simplificar el acceso a un conjunto de clases o interfaces. Proporcionar una interfaz unificada de alto nivel que, representando a todo un subsistema, facilite su uso. La "fachada" satisface a la mayoría de los clientes, sin ocultar las funciones de menor nivel a aquellos que necesiten acceder a ellas.

Solución: Symfony mediante el sistema de configuración de archivos YML implementa este patrón permitiendo acceder a diferentes configuraciones del Framework desde un único lugar. Además de que el acceso a la aplicación es a través del controlador frontal que implementa este patrón.

ADAPTER

Problema: ¿Cómo tener una única interfaz de acceso a múltiples bases de datos?

Propósito: Convertir la interfaz de una clase para que se adapte a lo que el cliente que la usa necesita, permitiendo así que trabajen juntas clases cuyas interfaces son incompatibles.

Solución: Symfony da la posibilidad de cambiar a otro sistema de base de datos completamente diferente a mitad de desarrollo, solo basta con configurar un archivo YAML. La capa de abstracción utilizada encapsula toda la lógica de los datos. El resto de la aplicación no tiene que preocuparse por las consultas SQL y el código SQL que se encarga del acceso a la base de datos es fácil de encontrar.

DECORATOR

Problema: ¿Cómo añadirle una nueva característica a una clase, pudiendo así obtener gran funcionalidad combinando piezas sencillas?

Propósito: Añadir responsabilidades adicionales a un objeto dinámicamente, proporcionando una alternativa flexible a la especialización mediante herencia, cuando se trata de añadir funcionalidades.

Solución: La plantilla no es un documento XHTML válido. Le faltan la definición del `DOCTYPE` y las etiquetas `<html>` y `<body>`. El motivo es que estos elementos se encuentran en otro lugar de la aplicación, un archivo llamado `Layout.php` que contiene el Layout de la página. Este archivo, que también se denomina plantilla global, almacena el código HTML que es común a todas las páginas de la aplicación, para no tener que repetirlo en cada página. El contenido de la plantilla se integra en el Layout, o si se mira desde el otro punto de vista, el Layout *decora* la plantilla.

Ejemplo:

Fig. # 7: Ejemplo de Implementación del patrón Decorator en Symfony

SINGLETON

Problema: Obtener un punto de acceso global a una clase.

Propósito: Asegurar que solo exista una instancia de una clase especifica en un sistema a desarrollar.

Solución: El controlador frontal proporciona un punto de acceso global a la aplicación. En una acción, el método `getContext`() devuelve el mismo *singleton*. Se trata de un objeto muy útil que guarda una referencia a todos los objetos del núcleo de Symfony relacionados con una petición dada, y ofrece un método de acceso para cada uno de ellos:

- `sfController`: El objeto controlador (->`getController`())
- `sfRequest`: El objeto de la petición (->`getRequest`())
- `sfResponse`: El objeto de la respuesta (->`getResponse`())
- `sfUser`: El objeto de la sesión del usuario (->`getUser`())
- `sfDatabaseConnection`: La conexión a la base de datos (->`getDatabaseConnection`())
- `sfLogger`: El objeto para los logs (->`getLogger`())
- `sfI18N`: El objeto de internacionalización (->`getI18N`())

Se puede llamar al singleton `sfContext`::`getInstance`() desde cualquier parte del código.

TEMPLATE METHOD

Problema: ¿Cómo definir la estructura de un algoritmo?

Propósito: Definir el esqueleto básico de un algoritmo. Se trata de una clase que define parte de un algoritmo mediante clases abstractas. Posteriormente, las subclases modifican los métodos abstractos para implementar las acciones concretas. Factorizar el comportamiento común de varias subclases Implementar las partes fijas de un algoritmo una sola vez y dejar que las subclases implementen las partes variables.

Solución: Symfony tiene una estructura específica a la hora de implementar un método, esto depende de la clase en que se implemente la funcionalidad.

BAJO ACOPLAMIENTO

Problema: ¿Cómo dar soporte a una dependencia escasa y a un aumento de la reutilización?

Propósito: Aumentar la reutilización y eliminar las dependencias entre las clases para propiciar un fácil mantenimiento y entendimiento.

Solución: Symfony asigna a cada clase una responsabilidad para mantener pocas dependencias entre las mismas.

ALTA COHESION

Problema: ¿Cómo mantener la complejidad dentro de límites manejables?

Propósito: Cada elemento dentro del diseño debe realizar una labor única dentro del sistema, no desempeñada por el resto de los elementos y auto-identificable.

Solución: Symfony agrupa las clases por funcionalidades que son fácilmente reutilizables, bien por su uso directo o por herencia.

CONTROLADOR

Problema: ¿Quién debería encargarse de un evento del sistema?

Propósito: Facilitar la centralización de actividades, delegar las actividades en otras clases con las que mantiene un modelo de alta cohesión.

Solución: Symfony implementa para cada vista de la aplicación una clase controladora que se encarga de atender el evento generado por la vista.

CONTROLADOR FRONTAL

Problema: ¿Quién debería encargarse de manejar las peticiones de los usuarios?

Propósito: Centralizar todas las peticiones de los usuarios y proveerlos de un punto único de entrada a la aplicación.

Solución: Todas las peticiones Web son manejadas por un solo controlador frontal, que es el punto de entrada único de toda la aplicación en un entorno determinado. Cuando el controlador frontal recibe una petición, utiliza el sistema de enrutamiento para enviar la acción al controlador responsable de la solución. Además de esto el controlador frontal se encarga de manejar la seguridad y la ejecución de los filtros en Symfony.

2.5 Comunicación

La comunicación entre los distintos componentes que componen una arquitectura es otro aspecto a señalar en la definición arquitectónica descrita en el Capítulo 1. A continuación se describirán los distintos elementos de comunicación a tener en cuenta en la solución planteada.

2.5.1 Comunicación entre capas.

La comunicación entre las capas en las que se divide la arquitectura está dada por mantener los paradigmas de la Programación Orientada a Objetos y la posibilidad de realizar sistemas con mayor portabilidad se utiliza entre el cliente y el Controlador Frontal, y viceversa, tecnología JSON, con esta se serializan los datos pasados por el usuario en Objetos PHP. Esto permite utilizar una interfaz de cualquier tipo para interactuar con el servidor de aplicación además de la seguridad que brinda este tipo de comunicación en materia de desarrollo, ya que los implementadores de interfaz de usuario no tienen acceso al comportamiento de los objetos enviados desde la capa controladora.

Siempre que esto sea posible la comunicación será utilizada en combinación con AJAX, para permitir mayor velocidad y dinamismo en la interacción de los sistemas por vía Web. La utilización de AJAX con comunicación basada en XML, se utilizará para los casos en los que sea necesario por no ser factible utilizar JSON.

La comunicación entre las capas del Controlador y el Modelo, gracias a la implementación del ORM, está dada por el envío de objetos que contienen la información necesaria para manejar las peticiones del usuario o las acciones a realizar. En caso de que los arreglos de datos a utilizar contengan múltiples registros serán manejados mediante arreglos que contienen los Objetos referentes a las clases del Modelo.

Para manejar la comunicación con la Base de Datos se utiliza Creole, librería que trae el Propel 1.2 para la abstracción de la BD. La utilización de esta librería es más lenta que utilizar comunicación mediante la librería PDO, pero esta no permite conectarse a Oracle 8i, pues las DLL compiladas para PHP 5.x no soportan la comunicación con esta versión de Oracle por ser incompatibles con esta versión.

2.5.2 Comunicación Interna.

La comunicación entre los distintos componentes internos de la aplicación está dada por los elementos del núcleo de Symfony para manejar los flujos de datos que serán explicados con más detalle en el próximo capítulo. Estos elementos son sfRequest, sfUser, sfResponse, Flash. Estas clases permiten el acceso a la información desde los distintos puntos del código de la aplicación, y tienen distintas formas de acceso desde cada uno de ellos para evitar violaciones del acceso al código.

2.6 Conclusiones

En este capítulo se expuso la solución arquitectónica propuesta para satisfacer los requisitos no funcionales del cliente. Su composición separada por componentes y la explicación de los paradigmas de la programación a utilizar para optimizar los resultados de los productos obtenidos. Los aspectos más importantes de esta arquitectura como son los Patrones Arquitectónicos y de Diseño más importantes a utilizar, así como la comunicación entre las capas e interna de las aplicaciones desarrolladas.

Capítulo 3: DESCRIPCIÓN ARQUITECTÓNICA

3.1 Introducción

En este capítulo se explica el funcionamiento interno del Framework Symfony y la forma en que esto afecta el desarrollo de aplicaciones Web. Se considera este un aspecto importante para lograr una cabal comprensión del marco de desarrollo para la arquitectura propuesta. También se referencia los aspectos arquitectónicamente significativos, el estándar de codificación en las aplicaciones, los distintos mecanismos de seguridad que se implementan, los modelos que permiten extender las funcionalidades mediante plugins y los sistemas de mantención y optimización de las aplicaciones desarrolladas mediante la utilización de Logs.

3.2 MVC según Symfony

Para realizar las más sencillas de las páginas en Symfony se necesitan al menos siete scripts diferentes:

- La capa del Modelo
  - Abstracción de la base de datos
  - Acceso a los datos
- La capa de la Vista
  - Vista
  - Plantilla
  - Layout
- La capa del Controlador
  - Controlador frontal
  - Acción

Por lo que parecen muchos archivos para abrir y modificar cada vez que se crea una página. Pero Symfony simplifica este proceso. Toma lo mejor de la arquitectura MVC y la implementa de forma que el desarrollo de aplicaciones sea rápido y sencillo.

En primer lugar, el controlador frontal y el Layout son comunes para todas las acciones de la aplicación. Se pueden tener varios controladores y varios Layout, pero solamente es obligatorio tener uno de cada. El controlador frontal es un componente que sólo tiene código relativo al MVC, por lo que no es necesario crear uno, ya que Symfony lo genera de forma automática.

Además las clases de la capa del modelo también se generan automáticamente, en función de la estructura de datos de la aplicación. La librería Propel se encarga de esta generación automática, ya que crea el esqueleto o estructura básica de las clases y genera automáticamente el código necesario. Cuando Propel encuentra restricciones de claves foráneas (o externas) o cuando encuentra datos de tipo fecha, crea métodos especiales para acceder y modificar esos datos, por lo que la manipulación de datos es algo muy sencillo. La abstracción de la base de datos es completamente invisible al programador, ya que la realiza otro componente específico llamado Creole. Así, si se cambia el sistema Gestor de Bases de Datos en cualquier momento, no se debe reescribir ni una línea de código, ya que tan sólo es necesario modificar un parámetro en un archivo de configuración.

Por último, la lógica de la vista se puede transformar en un archivo de configuración sencillo, sin necesidad de programarla.

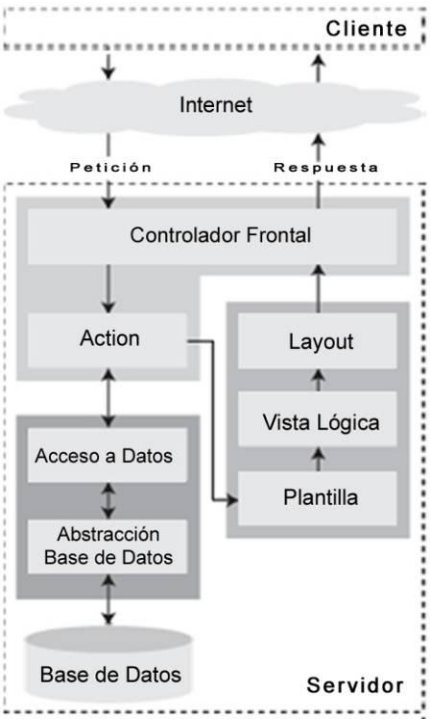

Fig. # 8: Implementación del Patrón MVC por Symfony

Al final, solo es necesario crear dos scripts de forma muy sencilla, debido a que los demás son creados de manera automática por el Framework. Al considerar el número de líneas de código, de un ejemplo de página creado según la arquitectura MVC en Symfony no requiere más líneas ni más tiempo de programación que un script simple.

Abstracción de la base de datos

La capa del modelo se puede dividir en la capa de acceso a los datos y en la capa de abstracción de la base de datos. De esta forma, las funciones que acceden a los datos no utilizan sentencias ni consultas que dependen de una base de datos, sino que utilizan otras funciones para realizar las consultas. Así, si se cambia de sistema gestor de bases de datos, solamente es necesario actualizar la capa de abstracción de la base de datos.

Los elementos de la vista

La capa de la vista también puede aprovechar la separación de código. Las páginas web suelen contener elementos que se muestran de forma idéntica a lo largo de toda la aplicación: cabeceras de la página, el Layout genérico, el pie de página y la navegación global. Normalmente sólo cambia el interior de la página. Por este motivo, la vista se separa en un Layout y en una plantilla. Casi siempre el Layout es global en toda la aplicación o al menos en un grupo de páginas. La plantilla sólo se encarga de visualizar las variables definidas en el controlador. Para que estos componentes interaccionen entre sí correctamente, es necesario añadir cierto código: *plantilla-vista-Layout*.

- *Plantilla*: Es la encargada de contener el cuerpo de la página, las variables que el controlador envía como respuesta a la petición o la consulta realizada al modelo.
- *Vista*: Contiene las variables de cabeceras y titulo de la página, además de una variable con el contenido de la plantilla, que sería el cuerpo de la página.
- *Layout*: Es la que contiene la mayor cantidad de código HTML, y la disposición de las variables pasadas por la vista en la página, lista para mostrarla al usuario.

Acciones y controlador frontal

En muchos casos el controlador de las aplicaciones web tiene responsabilidades en común con otros controladores. Algunas de estas tareas son, el manejo de las peticiones del usuario, el manejo de la seguridad, cargar la configuración de la aplicación y otras tareas similares. Por este motivo, el controlador normalmente se divide en un controlador frontal, que es único para cada aplicación, y las acciones, que incluyen el código específico del controlador de cada página. Una de las principales ventajas de utilizar un controlador frontal es que ofrece un punto de entrada único para toda la aplicación. Así, en caso de que sea necesario impedir el acceso a la aplicación, solamente es necesario editar el script correspondiente al controlador frontal. Si la aplicación no dispone de controlador frontal, se debería modificar cada uno de los controladores.

3.2.1 El Controlador

En Symfony el Controlador está dividido en varios componentes que se encargan de realizar distintas tareas y responsabilidades de la aplicación, estos son:

- *El controlador frontal* es el único punto de entrada a la aplicación. Carga la configuración y determina la acción a ejecutarse.
- *Las acciones* contienen la lógica de la aplicación. Verifican la integridad de las peticiones y preparan los datos requeridos por la capa de presentación.
- *Los objetos sfRequest, sfResponse y sfUser* dan acceso a los parámetros de la petición, las cabeceras de las respuestas y a los datos persistentes del usuario. Se utilizan muy a menudo en la capa del controlador.
- *Los filtros* son trozos de código ejecutados para cada petición, antes o después de una acción. Por ejemplo, los filtros de seguridad y validación son comúnmente utilizados en aplicaciones web. Puedes extender el Framework creando tus propios filtros.

Controlador Frontal

Es la única entrada a la aplicación, esto brinda una gran facilidad y seguridad a la aplicación, pues este es el que se encarga de recibir todas las peticiones del cliente y procesarlas, brindar los accesos a las acciones, desencadenar los filtros, etc.

El controlador frontal se encarga de despachar las peticiones, lo que implica algo más que detectar la acción que se ejecuta. De hecho, ejecuta el código común a todas las acciones, incluyendo:

1. Define las constantes del núcleo.
2. Localiza la librería de Symfony.
3. Carga e inicializa las clases del núcleo del Framework.
4. Carga la configuración.
5. Decodifica la URL de la petición para determinar la acción a ejecutar y los parámetros de la petición.
6. Si la acción no existe, re-direcciona a la acción del error 404.
7. Activa los filtros.
8. Ejecuta la acción y produce la vista.
9. Muestra la respuesta.

Las constantes del núcleo son definidas en primer lugar en el archivo que contiene el controlador frontal. Este archivo es el que inicializa las primeras constantes para el uso de la aplicación. Este archivo se encuentra en la raíz del directorio del servidor Web.

En estas constantes son utilizadas en el mismo archivo para acceder al archivo de configuración de la aplicación, donde se encuentra las clases y librerías de la aplicación y del núcleo de Symfony. Además se decodifica la petición del cliente para mediante un sistema de enrutamiento acceder a la acción deseada y enviar los parámetros que servirán de datos para el trabajo de la clase.

En caso de que la acción accedida no exista, el controlador frontal se encarga de re-direccionar a una página de Error404 –página no encontrada- o en caso de que por la activación de los filtros correspondientes con la acción solicitada se requiera de ciertos permisos para acceder a esta que el usuario no posea, la aplicación tomará la decisión más conveniente por el programador, incluyendo la salida de mensajes de Error404 u otro. En la activación de los Filtros se pueden realizar otras acciones que permitan optimizar la seguridad del sistema y la eficiencia, como la validación de los datos de los formularios enviados, etc.

Luego de que el controlador de la lógica del negocio realice la acción correspondiente y se produzca la vista correspondiente para mostrar el resultado de la acción solicitada, el controlador frontal se encarga de mostrar esta vista al cliente, repitiendo una vez más el proceso de recibir la solicitud de una nueva acción para enviar una nueva respuesta.

Symfony, trae un controlador frontal por defecto que es capaz de realizar todas estas acciones para una aplicación determinada, pero si se desea utilizar otro controlador para propósitos más específicos como el de administración y configuración de la aplicación, este permite la configuración de un nuevo controlador, incluso de que ambos trabajen en paralelo, como en el caso de un controlador para la parte productiva y el de uno para la parte de desarrollo, que también viene incluido en la implementación del Framework.

La diferencia entre uno y el otro es que para los desarrolladores brinda muchas facilidades como la posibilidad de realizar debugger desde la misma página de la aplicación, acceder a las variables enviadas por métodos POST y GET, los objetos del Framework `sfRequest` y `sfResponse`, las variables de Sesión y de Globales de PHP, la configuración, los sistemas de Logs y Mensajes de la aplicación en tiempo de ejecución, los tiempos de respuesta de la aplicación. Estos controladores tiene el nombre de la aplicación terminado en "`_dev`".

Acciones

Las acciones son el corazón de la aplicación, puesto que contienen toda la lógica de la aplicación. Las acciones utilizan el modelo y definen variables para la vista. Cuando se realiza una petición web en una aplicación Symfony, la URL define una acción y los parámetros de la petición.

Las acciones están agrupadas en clases que son llamadas `action` y se encuentran generalmente en los archivos `action.class.php`. Estás se encargan de contener las acciones como métodos que son llamados `executeMyAccion`. Esto le da a entender que el método que comience con la palabra `execute` corresponde a una acción que será llamada desde el exterior de la aplicación por un cliente determinado. Las clases además pueden contener otras funciones que no cumplan con esta regla de nomenclatura para las acciones internas de las clases o de apoyo a las otras funciones. Estas funciones deben ser privadas o protegidas –*private* o *protected*-.

Además se pueden crear clases que correspondan a acciones en específico. Estas clases tendrán el nombre de `myaccionAction`, estas se ubican en los archivos `myaccionAction.class.php` correspondiente para cada acción y dentro tendrán una función `execute` que será la correspondiente a la llamada de la acción. Las demás funciones son de apoyo a la acción, esto se utiliza para acciones

muy complicadas. La principal atención a tener en cuenta a la hora de utilizar estas clases es que no se podrán utilizar funciones implementadas en otras clases de acciones o en la clase `action`.

Todas estas clases heredan de las clases de Symfony `sfActions` y `sfAction` respectivamente. Estás clases del núcleo del Framework están preparadas para atender las peticiones de los clientes y contienen métodos que permiten hacer el trabajo de los programadores más sencillo y flexible. Entre estas se encuentra la posibilidad de manejar la salida de las acciones de distintas maneras para que el Framework entienda lo que debe hacer, utilizando la salida de mensajes como Error404, Mensaje de Texto enviado desde la acción, cabeceras HTTP o JSON, re-direccionar a otra acción, aplicación o URL externa a la aplicación, entre otros casos.

Para los casos en los que es necesario realizar un grupo de procedimientos antes o después de la llamada de todas las acciones, las clases `sfActions` y `sfAction` permiten el uso de los métodos `preExecute` y `postExecute` respectivamente. Esto ayuda a la reutilización del código que sea común para un gran grupo de acciones en el caso de las clases que hereden de `sfActions` y para el caso de las acciones que hereden de `sfAction` el código que se desee sea ejecutado justo antes o después de la llamada de la acción.

sfRequest

Para acceder a las peticiones realizadas por los clientes de la aplicación Symfony utiliza un objeto llamado sfRequest, esto objeto se puede acceder desde cualquier punto del código de la aplicación y contiene toda la información pasada por el usuario ya sea por POST, GET, JSON o AJAX, además de conocer alguna información del cliente que está accediendo a la aplicación, el servidor, la URL, etc.

Otras facilidades del objeto `sfRequest` es el trabajo con los archivos subidos al servidor. Esto se realiza mediante el objeto `sfWebRequest`. Esta clase tiene todas las características para poder trabajar con los archivos al servidor y manejarlos de la manera más adecuada.

sfResponse

Aunque el objeto response (objeto respuesta) es parte de la vista, normalmente se modifica en la acción. Las acciones acceden al objeto respuesta creado por Symfony, y llamado `sfResponse`,

mediante el método `getResponse()`. Esta clase contiene un conjunto de métodos `setters` y `getters` que le permiten al programador acceder a los datos que contiene, ya que funciona como un contenedor de atributos propiamente dicho.

Esta clase se utiliza para enviar toda la información necesaria al cliente y para construir la página resultante que el cliente espera a partir de los datos generados por el controlador.

sfUser

Symfony maneja automáticamente las sesiones del usuario y es capaz de almacenar datos de forma persistente entre peticiones. Utiliza el mecanismo de manejo de sesiones incluido en PHP y lo mejora para hacerlo más configurable y más fácil de usar.

El objeto `sfUser` es el encargado de manejar la información referente al usuario de manera que se maneja con las variables de sesión en PHP. Los datos guardados en este objeto estarán persistentes mientras dure la sesión en el servidor. Los atributos de usuarios pueden guardar cualquier tipo de información (cadenas de texto, arreglos y arreglos asociativos). Se pueden utilizar para cualquier usuario, incluso si ese usuario no se ha identificado.

Si se necesita guardar información entre peticiones pero solo de forma temporal, Symfony brinda la posibilidad de utilizar la clase `sfFlash`. Estos tipos de atributos son persistentes solo entre una petición y la siguiente, ya que después serán eliminados, es una manera eficiente de utilizar los datos temporales, ya que el programador no debe preocuparse por tener que eliminar la variable después de utilizarla.

El manejo de las sesiones en Symfony se trata por defecto tanto en el lado del cliente como del servidor (COOKIE y SESSION), aunque se puede configurar la para que la aplicación trate las sesiones solo de un lado o del otro, incluso las sesiones del lado servidor se pueden guardar en una tabla en una base de datos determinada.

Filtros

Los filtros son utilizados en Symfony, para atender las validaciones, y los procesos de seguridad entre otras cosas. Estos desencadenan validaciones a nivel de aplicación, o de módulo que son necesarias que se cumplan para el correcto funcionamiento de cada una de las acciones. Al igual que los métodos `preExecute y postExecute` estos son capaces de ejecutarse antes, o después de la llamada a una acción determinada, pero a un nivel más alto, ya que se pueden definir filtros a nivel de aplicación, que son comunes para todas las clases `action` de un proyecto.

Los filtros son capaces de utilizar todas las variables del Framework, tanto como los objetos `sfRequest`, `sfResponse` y `sfUser`. Datos muy necesarios para realizar las validaciones de las acciones. En los archivos de configuración filters.yml se configuran los filtros necesarios que deben utilizar cada una de las acciones o módulos de cada proyecto.

3.2.2 La Vista

La vista se encarga de producir las páginas que se muestran como resultado de las acciones. La vista en Symfony está compuesta por diversas partes, estando cada una de ellas especialmente preparada para que pueda ser fácilmente modificable por la persona que normalmente trabaja con cada aspecto del diseño de las aplicaciones.

- Los diseñadores web normalmente trabajan con las plantillas (que son la presentación de los datos de la acción que se está ejecutando) y con el Layout (que contiene el código HTML común a todas las páginas). Estas partes están formadas por código HTML que contiene pequeños trozos de código PHP, que normalmente son llamadas a los diversos HELPERS disponibles.
- Para mejorar la reutilización de código, los programadores suelen extraer trozos de las plantillas y los transforman en componentes y elementos parciales. De esta forma, el Layout se modifica para definir zonas en las que se insertan componentes externos. Los diseñadores web también pueden trabajar fácilmente con estos trozos de plantillas.
- Los programadores normalmente centran su trabajo relativo a la vista en los archivos de configuración YAML (que permiten establecer opciones para las propiedades de la respuesta y para otros elementos de la interfaz) y en el objeto respuesta. Cuando se trabaja con variables

en las plantillas, deben considerarse los posibles riesgos de seguridad de XSS (cross-site scripting) por lo que es necesario conocer las técnicas de escape de los caracteres introducidos por los usuarios.

La vista en Symfony se divide en tres partes fundamentales:

- Plantilla
- Vista Lógica
- Layout

Además el Framework brinda otras facilidades para ayudar a facilitar el entendimiento del código y la reutilización del mismo, algunos de los más importantes son:

- HELPERs
- Parciales
- Componentes
- Slots
- Slots de Componentes

Plantillas

Como se explica anteriormente el objetivo de la plantilla es el de contener el código HTML correspondiente a la salida de una acción determinada. El uso de código PHP debe ser el mínimo posible. Solo llamadas a las variables definidas en la acción o en los objetos `sfRequest`, `sfResponse`, `sfContext`, `sfUser` y `sfFlash` o la llamada a los HELPERS.

Las plantillas dan respuestas a las acciones, y es el primer eslabón en construir la respuesta que verá el cliente después que se ejecute la acción correspondiente.

Vistas

La vista se compone de dos partes principales:

1. El código HTML, correspondiente a la plantilla, el Layout o los fragmentos de plantillas.

2. El resto, que incluye entre otros los siguientes elementos:

- Declaraciones <meta>: palabras clave (keywords), descripción (description), duración de la cache, etc.
- El título de la página: no solo es útil para los usuarios que tienen abiertas varias ventanas del navegador, sino que también es muy importante para que los buscadores indexen bien la página.
- Inclusión de archivos: de JavaScript y de hojas de estilos.
- Layout: algunas acciones necesitan un Layout personalizado (ventanas emergentes, anuncios, etc.) o puede que no necesiten cargar ningún Layout (por ejemplo en las acciones relacionadas con AJAX).

El segundo aspecto es considerado la configuración y Symfony recoge dos formas de atender estas configuraciones. Una manera, y la más normal es mediante el archivo `view.yml` que es el que comprende la vista lógica. La vista lógica (`view.yml`) se encarga de todos los aspectos de la configuración, se utiliza cuando los valores de configuración no dependen del contexto o de alguna consulta a la base de datos. Cuando se trabaja con valores dinámicos que cambian con cada acción, se recurre al segundo método para establecer la configuración de la vista: añadir los atributos directamente en el objeto `sfResponse` durante la acción.

Este objeto es capaz de trabajar con los valores de las cabeceras HTTP, las COOKIES, los METAS y las cabeceras de las páginas mediante diferentes funciones que contiene la clase a las que se puede acceder desde las acciones, las plantillas o el Layout.

Layout

El Layout es el encargado de encapsular el código de la plantilla dentro de una página XHTML, esta contiene toda la información común para las plantillas o un grupo de ellas. En otras palabras el Layout se encarga de decorar la plantilla.

Los archivos del Layout se encuentran en el directorio `../templates/` de cada aplicación, en esta se pueden crear tantos Layout sean necesarios o redefinir el Layout por defecto que crea Symfony para cada aplicación.

HELPERs

Los HELPERS son métodos especiales que devuelven un fragmento de código HTML (o JavaScript) para etiquetas de formularios, hipervínculos, campos de texto, botones, etc. Los HELPERS facilitan la creación de las plantillas y producen el mejor código HTML posible en lo que se refiere al rendimiento y a la accesibilidad. Aunque se puede usar HTML normal y corriente, los HELPERS normalmente son más rápidos de escribir.

Los archivos que contienen las declaraciones de los HELPERS no son cargados automáticamente, ya que estos no son clases, sino archivos que contienen funciones. No obstante se pueden realizar las llamadas a los HELPERS de dos maneras. Una es utilizando la función `use_HELPER`(). Los HELPERS se agrupan en archivos de distintos tipos, por lo que si se desea utilizar los de texto, se le pasa por parámetro a la función `use_HELPER('text')`. O si se desea utilizar los HELPERS de varios tipos los parámetros pueden ser pasados separados por comas. La otra forma de cargar los HELPERS a utilizar es definiéndolos en el archivo settings.yml, los que sean definidos aquí, serán cargados automáticamente por el Framework al ejecutar cada una de las plantillas. Además Symfony incluye algunos HELPERS por defecto. Estos son:

- HELPER: se necesita para incluir otros HELPERs (de hecho, la función `use_HELPER`() también es un HELPER)
- Tag: HELPER básico para etiquetas y que utilizan casi todos los HELPERs
- Url: HELPERs para la gestión de enlaces y URL
- Asset: HELPERs que añaden elementos a la sección <head> del código HTML y que proporcionan enlaces sencillos a elementos externos (imágenes, archivos JavaScript, hojas de estilo, etc.)
- Partial: HELPERs que permiten incluir trozos de plantillas
- Cache: manipulación de los trozos de código que se han añadido a la cache
- Form: HELPERs para los formularios

De estos, los tres últimos de la lista (Partial, Cache y Form) se pueden obviar en el archivo `settings.yml`, esto puede incrementar ligeramente el rendimiento de la aplicación.

Si se quiere utilizar un HELPER fuera de una plantilla, se puede cargar un grupo de HELPERS desde cualquier punto de la aplicación mediante la función `sfLoader::loadHELPERs($HELPERs)`, donde

la variable `$HELPERs` es el nombre de un grupo de HELPERS o un arreglo con los nombres de varios grupos de HELPERS. Por tanto, si se quiere utilizar `auto_link_text()` dentro de una acción, es necesario llamar primero a `sfLoader::loadHELPERs('Text').`

A pesar de la gran variedad de HELPERS que contiene Symfony, si no se encuentra lo que se desee utilizar se pueden crear otros nuevos. Esto es tan sencillo como crear un archivo `NombreHELPER.php` en cualquiera de las carpetas `../lib/HELPER/` que se encuentran dentro de la aplicación. Así podrán ser llamados mediante el `Nombre` que tenga el fichero, que es el que define el grupo al que pertenece. Incluso redefinir los ya existentes para un módulo o aplicación específicos.

Parciales

Los elementos definidos como `partial` son pedazos de códigos que son reutilizados en las llamadas a distintas acciones. Estos fragmentos de código no requieren mucha lógica y pueden utilizar algunas variables, pero no realizan accesos al modelo. Son sencillamente segmentos que deben aparecer en distintos lugares del Layout o son utilizados por diferentes plantillas para completar distintas acciones.

Los elementos parciales son ubicados dentro del directorio `../template`/ de cada módulo o de la aplicación de forma global. Su nombre comienza con un guión bajo (`_nombreParcial.php`) para identificarlos de los demás elementos del directorio como plantillas y Layouts.

La llamada a los mismos se realiza desde las plantillas que los van a utilizar mediante la utilización de HELPERS específicos para ello. Estos reciben por parámetros el módulo donde se encuentra el parcial, el nombre del parcial y un arreglo con los parámetros que este va a utilizar para trabajar.

Componentes

Los Componentes son el equivalente a acciones pero que se ejecutan de manera más rápida. Estos son reutilizados en distintas plantillas para mostrar información y complementar las acciones. Estos son utilizados para realizar acciones dentro de las acciones, cuando en una plantilla es necesario mostrar información la cual requiere una lógica adicional, incluyendo el acceso al modelo, se recomienda el uso de los componentes.

Los componentes se agrupan en un archivo llamado `components.class.php` dentro del directorio `../action/` y la dicha clase hereda de `sfComponents`. Al igual que el fichero `action.class.php` este contiene la clase componentes, que agrupa las distintas acciones de los componentes, cada una de estas acciones comienza con la preposición `execute` al igual que las acciones. Las plantillas correspondientes a los componentes tienen la nomenclatura que las plantillas de las acciones, pero comenzando por un guión bajo. A continuación una pequeña comparación entre las clases `components` y `action`.

Como se observa en el Anexo III , las acciones de los componentes contienen su propia plantilla, por lo que pueden acceder a los mismos HELPERS a los que acceden las plantillas de las acciones.

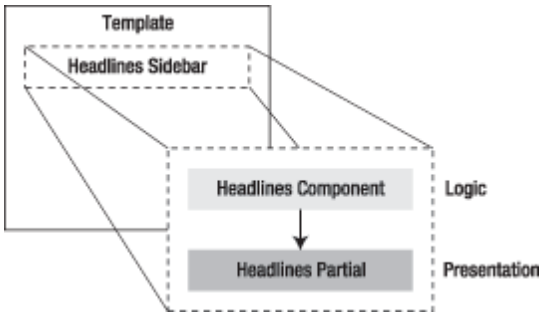

Fig. # 9: Ejemplo de utilización de componente en una plantilla.

Para la llamada de los componentes en las plantillas, se realiza al igual que con los parciales mediante la utilización de HELPERS destinados con este fin. En la llamada al HELPER se incluye el módulo y el nombre del componente, por lo que se pueden incluir componentes de otros módulos y aumentar la reutilización del código.

Se pueden incluir componentes dentro de otros componentes y también en el Layout global como si fuera una plantilla normal. Al igual que en las acciones, los métodos `execute` de los componentes pueden pasar variables a sus elementos parciales relacionados y pueden tener acceso a los mismos atajos. Pero las similitudes se quedan solo en eso. Los componentes no pueden manejar la seguridad ni la validación, no pueden ser llamados desde Internet (solo desde la propia aplicación) y no tienen distintas posibilidades para devolver sus resultados. Por este motivo, los componentes son más rápidos que las acciones.

Slots

Los componentes y los parciales son utilizados para la reutilización de código, pero en ocasiones se desea rellenar ciertas partes del Layout con pedazos de código que tengan un comportamiento distinto según la acción que se ejecute, o que se creen de forma dinámica según el resultado de una acción determinada. Para estos casos es recomendable el uso de los Slots.

Un slot es una zona que se puede definir en cualquier elemento de la vista (Layout, plantilla o elemento parcial). La forma de rellenar esa zona es similar a establecer el valor de una variable. El código de relleno se almacena de forma global en la respuesta, por lo que se puede definir en cualquier sitio (Layout, plantilla o elemento parcial). Se debe definir un slot antes de utilizarlo y también hay que tener en cuenta que el Layout se ejecuta después de la plantilla y que los elementos parciales se ejecutan cuando los llama una plantilla.

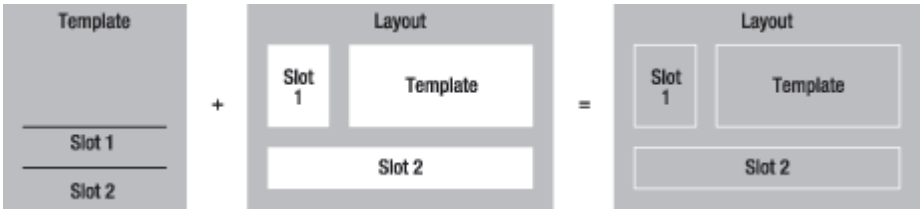

Fig. # 10: Decoración de una plantilla mediante Slots.

Durante el proceso de creación de la plantilla, se definen los slots con los valores correspondientes que tendrán a la hora de ejecutarse la lectura del Layout. Por lo que estas variables serán leídas por el mismo en el momento de construir la vista.

Las llamadas a los Slots se realizan mediante la utilización de HELPERS, estas llamadas se realizan en el Layout, ya que la plantilla se encarga de la definición de los mismos. Por su parte, en la plantilla se definen los Slots mediante los métodos `slot()` y `end_slot()`, dentro se pone el código correspondiente al mismo. Cada slot se define con un nombre, el cual se utiliza para ser llamado por los HELPERS. También se pueden realizar comprobaciones en el Layout mediante `has_slot('nombre')` que permite comprobar si un slot está definido para la acción correspondiente.

Slots de Componentes

Al igual que los slots, los slots de componentes son zonas que se pueden definir en los elementos de la vista. La principal diferencia reside en la forma en la que se decide qué código rellena esas zonas. En un slot normal, el código se establece en otro elemento de la vista; en un slot de componentes, el código es el resultado de la ejecución de un componente, y el nombre de ese componente se obtiene de la configuración de la vista lógica (`view.yml`).

Los slots de componentes se pueden utilizar para añadir en las páginas web las "migas de pan", los menús de navegación que dependen de cada página y cualquier otro contenido que se deba insertar de forma dinámica. Como componentes que son, se pueden utilizar en el Layout global y en cualquier plantilla, e incluso en otros componentes. La configuración que indica el componente de un slot siempre se extrae de la configuración de la última acción que se ejecuta.

3.2.3 El Modelo

La lógica de negocio de las aplicaciones web depende casi siempre en su modelo de datos. El componente que se encarga por defecto de gestionar el modelo en Symfony es una capa de tipo ORM (`object-relational mapping`) realizada mediante el proyecto Propel. En las aplicaciones realizadas con Symfony, el acceso y la modificación de los datos almacenados en la base de datos se realiza mediante objetos; de esta forma nunca se accede de forma explícita a la base de datos. Este comportamiento permite un alto nivel de abstracción y permite una fácil portabilidad.

El modelo que se implementa con la utilización de Symfony está basado en los siguientes componentes:

- Abstracción del Acceso a Datos.
- Schema.
- ORM.
    - Objetos.
    - Peer.
    - Criteria.

Abstracción de la Base de Datos

Existe otra consideración importante que hay que tener en cuenta cuando se crean elementos de acceso a los datos: las empresas que crean las bases de datos utilizan variantes diferentes del lenguaje SQL. Si se cambia a otro sistema gestor de bases de datos, es necesario reescribir parte de las consultas SQL que se definieron para el sistema anterior. Si se crean las consultas mediante una sintaxis independiente de la base de datos y un componente externo se encarga de traducirlas al lenguaje SQL concreto de la base de datos, se puede cambiar fácilmente de una base de datos a otra. Este es precisamente el objetivo de las capas de abstracción de bases de datos. Esta capa obliga a utilizar una sintaxis específica para las consultas y a cambio realiza el trabajo sucio de optimizar y adaptar el lenguaje SQL a la base de datos concreta que se utiliza.

La principal ventaja de la capa de abstracción es la portabilidad, porque hace posible el cambiar la aplicación a otra base de datos, incluso en mitad del desarrollo de un proyecto. Si se debe desarrollar rápidamente un prototipo de una aplicación y el cliente no ha decidido todavía la base de datos que mejor se ajusta a sus necesidades. Symfony utiliza Propel como ORM y Propel utiliza Creole como capa de abstracción de bases de datos. Estos 2 componentes externos han sido desarrollados por el equipo de Propel, y están completamente integrados en Symfony, por lo que se pueden considerar una parte más del Framework.

Schema

El esquema (`schema`) es lo que le permite a Symfony mapear la base de datos a el ORM. Se trata de un archivo de configuración (.YML o .XML) que contiene la información de la Base de Datos referente a las tablas y relaciones. Lo que permite hacer que el Framework realice la producción de las clases del modelo de manera automatizada. El esquema, se crea cada vez que se va a generar el modelo, y mediante Symfony se puede generar automáticamente a partir de la lectura de la Base de Datos.

ORM

Las bases de datos son relacionales y las aplicaciones desarrolladas a partir de PHP 5 y Symfony están orientadas a objetos. Para acceder de forma efectiva a la base de datos desde un contexto orientado a objetos, es necesaria una interfaz que traduzca la lógica de los objetos a la lógica

relacional. Esta interfaz se llama ORM o “mapeo de objetos a bases de datos”, y está formada por objetos que permiten acceder a los datos y que contienen en sí mismos el código necesario para hacerlo.

La principal ventaja que aporta el ORM es la reutilización, permitiendo llamar a los métodos de un objeto de datos desde varias partes de la aplicación e incluso desde diferentes aplicaciones. La capa ORM también encapsula la lógica de los datos permitiendo abstraer de la Base de Datos la lógica del negocio, así se encapsulan los métodos que sean necesarios para el negocio dentro de las clases. Todo el código repetitivo de acceso a los datos y toda la lógica de negocio de los propios datos se puede almacenar en esos objetos.

Las clases generadas por el ORM se dividen en dos partes fundamentales:

- Las clases de los Objetos
- Las clases Peer

Estos son el resultado de dos patrones de Acceso a Datos, `Tabla Data Gateway` y `Row Data Gateway`, estos se encargan de manejar las tablas y los registros de las base de datos respectivamente.

Clases Objeto y Peer

Las instancias de las clases `Objetos` que se encuentran en los archivos O`bjeto.php` y `BaseObjeto.php` representan registros de las tablas correspondientes. Se utilizan para manejar los datos de una instancia de las tablas, hacer cambios en el registro y guardarlos.

Las clases `Peer` se encuentran en los ficheros `ObjetoPeer.php` y `BaseObjetoPeer.php` y se encargan de manejar las funciones que controlan las distintas instancias de las clases Objeto. Clases que tienen métodos estáticos para trabajar con las tablas de la base de datos. Proporcionan los medios necesarios para obtener los registros de las tablas. Sus métodos devuelven normalmente un objeto o una colección de objetos de la clase `Objeto` relacionada.

La creación de cuatro clases para cada tabla, es debido a que a la hora de personalizar las clases, se debe hacer en las clases que no tienen el nombre `Base`, ya que estas no se modifican a la hora de volver a generar el modelo de datos, en caso de que existan cambios en el esquema. Por lo que todas las personalizaciones necesarias se realizarán en las clases O`bjeto` y `ObjetoPeer`. Estas heredan de las clases `BaseObjeto y BaseObjetoPeer` y generalmente están vacías en el momento de ser generadas, pero en caso de cambios no son afectadas, a diferencia de las clases `Base`.

Para el caso de las clases `ObjetoPeer` se deben crear funciones con las consultas que se van a utilizar para cada objeto. Esto facilita la implementación del controlador, pues evita tener que implementar las mismas consultas varias veces.

Además de agregar nuevas funcionalidades a las clases `Objeto` y `ObjetoPeer` gracias a la herencia que estos tienen de las clases `Base`, esto permite también redefinir los métodos de estas clases para adaptarlos mejor a las necesidades de cada clase.

Las clases `Peer` tienen funciones que permiten realizar SELECT, UPDATE, DELETE tanto de un objeto como de varios registros, todos estos métodos reciben como parámetro un objeto de la clase `Critetia`, que es la encargada de realizar las consultas SQL.

Criteria

Realizar consultas a una tabla no es necesario para realizar un modelo eficiente, si se quiere abstraer la aplicación de la Base de Datos completamente se debe evitar por completo el uso de código SQL en la mayor medida posible, sobre todo a la hora de realizar consultas un poco más complejas. Para solucionar este problema Propel cuenta con la clase Criteria que permite realizar las consultas de manera independiente del código SQL y completamente orientado a los paradigmas de la POO.

El Anexo IV muestra una tabla de equivalencia entre el lenguaje SQL y la clase Criteria.

3.3 Organización del código

Symfony organiza el código fuente en una estructura de tipo proyecto y almacena los archivos del proyecto en una estructura estandarizada de tipo árbol.

3.3.1 Proyecto

Symfony considera un proyecto como "un conjunto de servicios y operaciones disponibles bajo un determinado nombre de dominio y que comparten el mismo modelo de objetos". Dentro de un proyecto, las operaciones se agrupan de forma lógica en aplicaciones.

3.3.2 Aplicación

Una aplicación se ejecuta de forma independiente respecto de otras aplicaciones del mismo proyecto. Lo habitual es que un proyecto contenga dos aplicaciones: una para la parte pública y otra para la parte de gestión, compartiendo ambas la misma base de datos. También es posible definir proyectos que estén formados por varios sitios web pequeños, cada uno de ellos considerado como una aplicación. En este caso, es importante tener en cuenta que los enlaces entre aplicaciones se deben indicar de forma absoluta.

3.3.3 Módulo

Cada aplicación está formada por uno o más módulos. Un módulo normalmente representa a una página web o a un grupo de páginas con un propósito relacionado. Por ejemplo, una aplicación podría tener módulos como *home, artículos, ayuda, carritoCompra, cuenta*, etc.

3.3.4 Acción

Los módulos almacenan las acciones, que representan cada una de las operaciones que se puede realizar en un módulo. Por ejemplo el módulo carritoCompra puede definir acciones como añadir, mostrar y actualizar. Normalmente las acciones se describen mediante verbos. Trabajar con acciones

es muy similar a trabajar con las páginas de una aplicación web tradicional, aunque en este caso dos acciones diferentes pueden acabar mostrando la misma página (como por ejemplo la acción de añadir un comentario a una entrada de un blog, que acaba volviendo a mostrar la página de la entrada con el nuevo comentario).

Para las aplicaciones desarrolladas por el PSTA la estructura de la organización del código sería de la siguiente manera, la estructura del polo estará centralizada en el proyecto y por tanto en la raíz principal del proyecto. Este a su vez contiene las distintas aplicaciones que se desarrollen, y cada una de estas aplicaciones con sus módulos correspondientes.

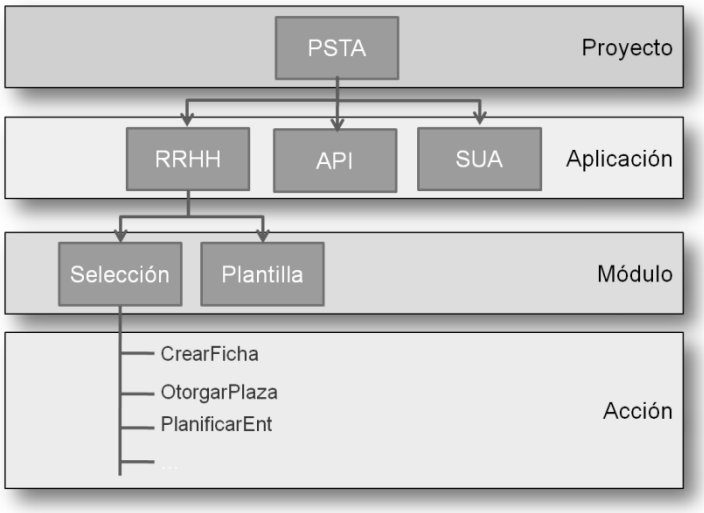

Fig. # 11: Ejemplo de la Estructura de las Aplicaciones con Symfony en el PSTA

3.3.5 Estructura del Directorio

Debido a que todas las aplicaciones Web dentro de un proyecto utilizan componentes en común o comparten ciertos contenidos como son:

- Una base de datos.
- Archivo estáticos (HTML, imágenes, archivos de JavaScript, hojas de estilos, etc.).
- Archivos subidos al sitio web por parte de los usuarios o los administradores.
- Clases y librerías PHP.
- Librerías externas (scripts desarrollados por terceros).

- Archivos que se ejecutan por lotes (batch files) que normalmente son scripts que se ejecutan vía línea de comandos o mediante CRON.
- Archivos de log (las trazas que generan las aplicaciones y/o el servidor).
- Archivos de configuración.

Symfony proporciona una estructura en forma de árbol de archivos para organizar de forma lógica todos esos contenidos, además de ser consistente con la arquitectura MVC utilizada y con la agrupación proyecto / aplicación / módulo. Cada vez que se crea un nuevo proyecto, aplicación o módulo, se genera de forma automática la parte correspondiente de esa estructura. Además, la estructura se puede personalizar completamente, para reorganizar los archivos y directorios o para cumplir con las exigencias de organización de un cliente.

De forma general la organización física es la siguiente:

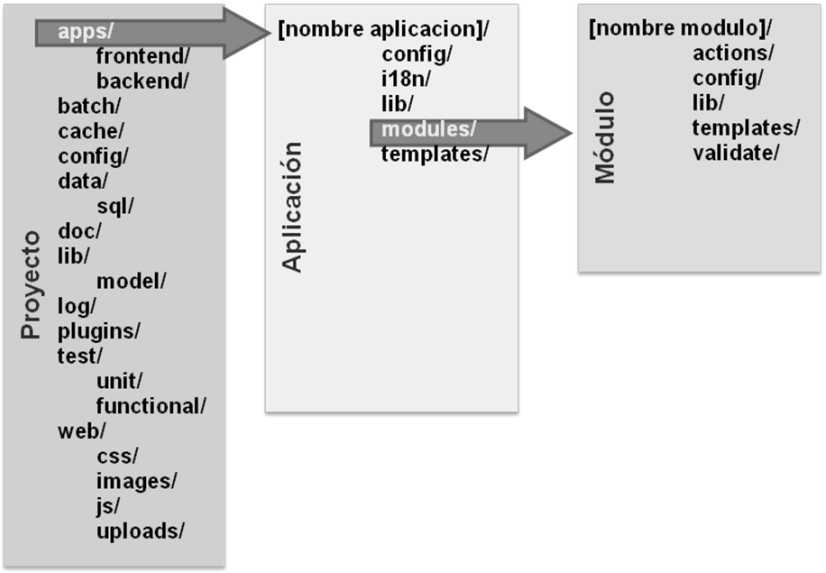

Fig. # 12: Organización física para las aplicaciones con Symfony

En el Anexo V se encuentra una breve descripción de cada uno de estos directorios.

3.4 Configuración

3.4.1 Configuración en PHP

La utilización de PHP como lenguaje de desarrollo trae la particularidad de tener que configurar el núcleo del lenguaje para obtener un mejor resultado a la hora de su utilización. La configuración de PHP se realiza mediante la modificación del archivo PHP.ini o mediante la modificación de dichas opciones en tiempo de ejecución mediante la llamada a opciones del lenguaje.

Por medidas de seguridad es recomendable comprobar la activación de las opciones de configuración antes de ejecutar las aplicaciones, para que en caso contrario poder activar o desactivar las opciones deseadas sin correr el riesgo de que uno de los sistemas esté ejecutándose en un entorno que no sea seguro.

3.4.2 Configuración en Symfony

Symfony es un Framework muy flexible, y su flexibilidad se basa principalmente en su poder de ser configurado en un alto grado y detalle. El Framework permite realizar una configuración en cascada lo que significa que las configuraciones más generales son válidas para todas las dependencias, pero además se pueden redefinir en cada una de las aplicaciones o módulos.

Los archivos de configuración de Symfony se basan principalmente en archivos YAML (.YML), pero la configuración también se puede encontrar en archivos PHP directamente, ya que al final los archivos YAML son convertidos al código PHP que es almacenado en la caché del proyecto. Por esta razón cada vez que un archivo de configuración YAML es modificado se debe limpiar la caché del proyecto para que esta se vuelva a generar en la próxima petición a la aplicación. En el Anexo VI se muestra un listado de los archivos utilizados por Symfony para su configuración.

Configuración en Cascada

Para las opciones de configuración que deben tener valores diferentes para distintos archivos Symfony precisa las opciones de configuración en cascada. Para hacer esto posible se definen los distintos niveles de configuración:

- Niveles de granularidad:
  - Configuración por defecto establecida por el Framework
  - Configuración global del proyecto (`miproyecto/config`/)
  - Configuración local de cada aplicación (/`apps/miaplicacion/config`/)
  - Configuración local de cada módulo (/`modules/mimodulo/config`/)
- Niveles de entornos de ejecución:
  - Específico para un solo entorno
  - Para todos los entornos

Muchas de las opciones que se pueden establecer dependen del entorno de ejecución. Por este motivo, los archivos de configuración YAML están divididos por entornos, además de incluir una sección que se aplica a todos los entornos.

3.4.3 Otras configuraciones

Además de configurar el PHP y Symfony se debe tener una correcta configuración de los archivos del Servidor Web Apache, así como el servidor de Base de Datos, ya sea para Oracle 8+, PostgreSQL u otro sistema de Base de Datos que se utilice. Esto aumenta la seguridad y el rendimiento de las aplicaciones, además de que ciertos requerimientos son necesarios para el correcto funcionamiento de los sistemas.

En el Anexo VII , se muestran algunos requerimientos a tener en cuenta en las configuraciones generales de los archivos utilizados por el servidor de aplicación.

3.5 Plugins

Los plugins son librerías de archivos que cumplen un propósito específico. Pueden ser considerados componentes que se le agregan a las aplicaciones para aumentar su rendimiento y extender sus funcionalidades. Además de que brinda la posibilidad de reutilizar una porción de código en determinadas partes de la aplicación o entre distintas aplicaciones, sobre todo cuando este código comprende algo más que una clase.

Los plugins permiten agrupar todo el código diseminado por diferentes archivos y reutilizar este código en otros proyectos. Los plugins permiten encapsular clases, filtros, *mixins*, *HELPERS*, archivos de configuración, tareas, módulos, esquemas y extensiones para el modelo, *fixtures*, archivos estáticos, etc. Los plugins son fáciles de instalar, de actualizar y de desinstalar. Se pueden distribuir en forma de archivo comprimido `.tgz`, un paquete PEAR o directamente desde el repositorio de código. La ventaja de los paquetes PEAR es que pueden controlar las dependencias, lo que simplifica su actualización. La forma en la que Symfony carga los plugins permite que los proyectos puedan utilizarlos como si fueran parte del propio Framework.

Otra efectividad que se aprovecha de los plugins es la fortaleza que estos brindan a la seguridad de las aplicaciones. Ya que permite la creación de componentes que permitan realizar comprobaciones de seguridad verticalmente sobre la aplicación en cualquier lugar que esta la requiera.

Se selecciona crear un Plugins cuando el código que se desea incluir para un propósito específico requiere de más de un archivo. En caso contrario se crea en un archivo dentro del directorio `/lib` y el `autoload` de las aplicaciones se encargará de incluir el contenido del archivo para ser utilizado desde el punto de la aplicación necesario. Los Plugins son componentes fáciles de instalar, actualizar y desinstalar; además de ser una manera sencilla de distribuir el código a terceros.

En el repositorio de componentes de Symfony existen diversos Plugins creados por el grupo de desarrollo del proyecto y por otros usuarios de Symfony que han hecho sus contribuciones. Estos brindan diversas funcionalidades, además es posible crear plugins para funcionales específicas para las aplicaciones propias.

3.5.1 Instalación

El proceso de instalación de los plugins difiere del modo en que ellos vengan empaquetados. Para esto se debe leer el archivo README del plugins. No obstante siempre que se instale un plugins se debe limpiar la caché de la aplicación de Symfony para el correcto funcionamiento del mismo.

Los plugins pueden ser instalados a nivel de aplicación o de proyecto, los métodos de instalación descritos a continuación colocan los archivos de la instalación en el directorio `miproyecto/plugins/nombrePlugins`.

La manera más común de instalar un Plugins es mediante el método PEAR. Este tiene tres formas básicas. Desde el sitio de Symfony, desde un canal de PEAR y desde el disco local. Las tres formas utilizan el siguiente formato respectivamente:

```
> cd miproyecto
> php symfony plugin-install http://plugins.symfony-project.com/nombrePlugins
> php symfony cc
```

Fig. # 13: Instalación de un Plugins desde SVN de Symfony

```
> cd miproyecto
> php symfony plugin-install nombreCanal/nombrePlugins
> php symfony cc
```

Fig. # 14: Instalación de un Plugins por un canal de PEAR

```
> cd miproyecto
> php symfony plugin-install camino/del/plugins/nombrePlugins.tgz
> php symfony cc
```

Fig. # 15: Instalación de un Plugins desde un Archivo en el Disco Duro.

La otra manera de realizar la instalación desde un archivo desde el disco local es de forma manual, esta requiere descomprimir el archivo del plugins en una carpeta `miproyecto/plugins/nombrePlugins`, y en caso de que el plugins contenga dentro alguna carpeta llamada `web`, se copia para `miproyecto/web/nombrePlugins`.

Hay que tener en cuenta que a la hora de instalar un Plugins hay algunas tareas que el comando `plugin-install` no realiza, por lo que hay que realizarlas a mano.

- El código de los plugins puede hacer uso de una configuración propia, pero no se pueden indicar los valores por defecto en un archivo de configuración `app.yml` dentro del directorio `config/` del plugin. Para trabajar con valores por defecto, se utilizan los segundos argumentos opcionales en las llamadas a los métodos `sfConfig::get()`. Las opciones de configuración se pueden redefinir en el nivel de la aplicación.
- Las reglas de enrutamiento propias se deben añadir manualmente en el archivo `routing.yml`.
- Los filtros propios también se deben añadir manualmente al archivo `filters.yml` de la aplicación.
- Las factorías propias se deben añadir manualmente al archivo `factories.yml` de la aplicación.

3.5.2 Activación

La activación de los plugins se realiza a nivel de aplicación. Para cada una de las aplicaciones que requieran de la utilización de un plugins se debe configurar en el archivo `miaplicacion/config/settings.yml`:

```
all:
  .settings:
    enabled_modules:  [default, sfMiPlugin]
```

Fig. # 16: Activación de un Plugins en el archivo settings.yml.

3.5.3 Desinstalación

Para desinstalar un plugins se utiliza el comando `plugin-uninstall` seguido del canal del plugins y el nombre. Por ejemplo:

```
> cd miproyecto
> php symfony plugin-uninstall pear.symfony-project.com/sfMiPlugins
> php symfony cc
```

Fig. # 17: Desinstalación de un Plugins en Symfony mediante la consola.

Para conocer los datos de los plugins instalados en la aplicación junto con sus canales correspondientes, se utiliza el comando `plugin-list`.

3.5.4 Estructura de un Plugins

Para que las aplicaciones en Symfony, puedan interpretar los plugins, estos deben tener una estructura específica de directorios y archivos. La estructura de directorios de un Plugins es muy semejante a la de una aplicación en Symfony. Ver Anexo VIII

Esta estructura permite que la aplicación interprete el plugins como parte del Framework, esto se pone de manifiesto de las siguientes maneras:

- Las tareas de Propel se ejecutan tanto para las aplicaciones como para los plugins que requieran de un modelo. Para esto los esquemas de los plugins siempre deben tener un atributo `package` que siga la notación `plugins.nombrePlugin.lib.model`, como se muestra en el ejemplo:

```
propel:
  _attributes: { package: plugins.miPlugin.lib.model }
```

Fig. # 18: Atributo package en el archivo miPlugin/config/schema.yml

- La ejecución de la configuración del plugins descrita en `config.php` se realiza después de que se corre la ejecución del proyecto y la aplicación, por lo que las modificaciones que incluyen la utilización del plugins sobre escriben la configuración de la aplicación.
- Los archivos de datos o *fixtures* del directorio `data/fixtures/` del plugins se procesan mediante la tarea `propel-load-data`.
- Las tareas del plugins están disponibles en la línea de comandos de Symfony tan pronto como se instala el plugins.
- Las clases propias se cargan automáticamente de la misma forma que las clases que se guardan en las carpetas `lib/` del proyecto.
- Cuando se realiza una llamada a `use_HELPER()` en las plantillas, se cargan automáticamente los HELPERS de los plugins. Estos HELPERS deben encontrarse en un subdirectorio llamado `HELPER/` dentro de cualquier directorio `lib/` del plugin.
- Los módulos proporcionan nuevas acciones, siempre que se declaren en la opción `enabled_modules` de la aplicación.

En general, todas las configuraciones que deben realizarse sobre los archivos de configuración de las aplicaciones, se tienen que añadir manualmente. Los plugins que requieran esta instalación manual, deberían indicarlo en el archivo README incluido.

3.5.5 Plugins Utilizados

Debido a la gran cantidad de Plugins disponibles en los repositorios de Symfony y a la cantidad de diversas soluciones que estos brindan, es imposible enumerar los plugins que se utilizarán para las aplicaciones desarrolladas por el PSTA. Se dará una breve descripción de los conjuntos en que se agrupan estos:

Generación Dinámica

Este grupo de plugins permiten la generación dinámica de algunos componentes Web como son Barras de Navegación, Tabs basados en CSS, Tablas generadas a partir de una lista de Objetos entre otras cosas.

Javascript

Los plugins de Javascript se componen en dos grupos, desasociados a Frameworks de este tipo y los asociados a algún Framework determinado. Su función principal es la de facilitar HELPERs específicos para facilitar el trabajo con componentes Javascript. En el caso de los asociados, ayudan la interacción de Symfony con algunos de los Frameworks para Javascript como Dojo, Ext, YUI, jQuery, Prototype, Scriptaculous, etc.

Medios de Comunicación

Permiten manejar la subida y gestión de archivos de información como imágenes, videos, flash, sonido y documentos PDF.

Extensiones de Propel

Los plugins que permiten ampliar las facilidades de utilizar Propel se separan de forma General y en los de comportamiento. Estos plugins permiten optimizar la utilización de Propel y hacer de esto una grata experiencia para los desarrolladores. Los Plugins de comportamiento se especializan en agregar funcionalidades a las clases del modelo mediante determinados comportamientos que brindan un valor agregado a dichas clases. Se basan en funciones elegantes que se deben incluir en el diseño e implementación para dar solución a algunos problemas comunes que se pueden presentar en la implementación de una solución.

Remplazar el Modelo

Plugins que permiten el trabajo con otros ORMs que el utilizado por Symfony por defecto (Propel 1.2) como son el caso de Doctrine y Propel 1.3

Remplazar la Vista

Permite integrar la capa de la vista utilizada por Symfony con otras tecnologías como Smarty, Haml, XSL u OPT.

Correctores gramaticales (Parsers)

Permiten realizar correcciones gramaticales al código PHP mediante la integración con aplicaciones de este tipo.

Rendimiento

Los plugins de rendimiento, como bien su nombre indica aumentan considerablemente la velocidad y estabilidad de las aplicaciones realizadas con el Framework, utilizando la caché de las aplicaciones en muchos casos para optimizar las funciones implementadas.

Administración de Proyecto

Los plugins de administración permiten mejorar las tareas de este tipo que trae Symfony incorporadas además de incorporar otras nuevas.

Seguridad

Optimizan la seguridad de las aplicaciones implementados sobre Symfony ampliando las posibilidades de configuración y los mecanismos de seguridad.

Anti-Spam

Aumenta la seguridad de los proyectos previniendo el registro automático de usuarios, así como controlar el flujo de acciones que realiza un usuario determinado de forma simultánea o denegar el acceso a las aplicaciones vía IP, entre otras facilidades.

Administración de Usuarios

Amplía la gestión de información referente a los usuarios conectados a la aplicación.

i18n y l10n

Mejora la experiencia de los desarrolladores y los usuarios en cuanto a la internacionalización y localización del contenido de las aplicaciones.

Soluciones para el pago con tarjetas de crédito

Automatiza las tereas de validación y gestión de tarjetas de créditos y transacciones bancarias de los usuarios.

Correo electrónico

Permiten realizar validaciones sobre direcciones de correo electrónicos y evitar entradas Spam de este tipo.

Misceláneas

En este grupo se encuentran otros tipos de plugins que enriquecen las aplicaciones realizadas con el Framework brindándoles diversas funcionalidades. Por ser un grupo muy diverso no deja de ser importante pues en él se encuentran varios de los plugins que brindan algunas de las funciones más importantes de Symfony, algunos ejemplos son:

- ckWebServicePlugin: Permite publicar las acciones de las aplicaciones en Symfony como Servicios Web.
- sfAjaxWebDebugPlugin: Habilita una barra de herramientas para depurar las aplicaciones con AJAX
- sfDateTimePlugin: Permite manipular las fechas en las aplicaciones de manera casi “mágica”.
- sfEventCalendarPlugin: Crear de manera sencilla un calendario de eventos. Retornar de manera fácil los datos de un día, semana, mes o año.
- sfMenuGeneratorPlugin: Permite la generación dinámica de Menús. Estos son configurables a partir de los archivos app.yml y module.yml
- sfPhpExcelPlugin: Permite la generación de ficheros MS Office Excel 5 o 2007.
- sfPrinterFriendlyPlugin: Implementa una interfaz amigable para la impresión desde cualquier página de una aplicación.
- sfSpyPlugin: Permite registrar todas las acciones de los usuarios en una Base de Datos, muy útil a la hora de realizar una auditoría de la aplicación.
- sfSugarCRMPlugin: Permite manejar distintas llamadas SOAP de manera simultánea, es decir maneja varios WSDL como si fueran uno solo.

Así además se pueden encontrar en el sitio oficial de Symfony numerosos plugins que son añadidos y mejorados constantemente.

3.6 Seguridad

3.6.1 Seguridad del Sistema de Directorios

El sistema de directorios que ofrece la arquitectura de Symfony permite separar los scripts donde se encuentran recogidos todos los segmentos de código de la aplicación del lado del servidor, dígase archivos PHP, YML y XML(de configuración), de los archivos asequibles por el cliente mediante su navegador. Esto se debe a que todos estos archivos no se encuentran en una carpeta compartida por el Servidor Web, por lo que no se puede acceder a ella mediante ninguna vía.

Todos los archivos *publicados* (CSS, JS, Imágenes y los Controladores Frontales) se encuentran en una carpeta que es la carpeta donde el usuario navegará. Se hace necesario configurar el Servidor Web Apache para las aplicaciones que se hospeden en un Host compartido con otras aplicaciones de otra índole como el que se muestra a continuación:

```
<VirtualHost *:80>
  ServerName miaplicacion.dominio.cu
  DocumentRoot "/var/htdocs/miproyecto/web"
  DirectoryIndex index.php
  <Directory "/var/htdocs/miproyecto/web">
    AllowOverride All
    Allow from All
  </Directory>
```

Fig. # 19: Configuración del Servidor Web para permitir el acceso solo al directorio "web".

Además de esta configuración Symfony permite guardar los directorios que contienen las librerías del Framework y de la aplicación en cualquier lugar del servidor. Para que el controlador frontal reconozca donde se guarda este directorio se carga en un grupo de constantes que se define al comienzo de cada controlador, esto ayuda además en la utilización de varios controladores frontales para la misma aplicación, ayudando a manejar las dos partes de una aplicación como la parte de gestión y la de administración.

3.6.2 Seguridad de la Acción

La posibilidad de ejecutar una acción puede ser restringida a usuarios con ciertos privilegios. Las herramientas proporcionadas por Symfony para este propósito permiten la creación de aplicaciones seguras, en las que los usuarios necesitan estar autenticados antes de acceder a alguna característica o a partes de la aplicación. Añadir esta seguridad a una aplicación requiere dos pasos: declarar los requerimientos de seguridad para cada acción y autenticar a los usuarios con privilegios para que puedan acceder estas acciones seguras.

Antes de ser ejecutada, cada acción pasa por un filtro especial que verifica si el usuario actual tiene privilegios de acceder a la acción requerida. En Symfony, los privilegios están compuestos por dos partes:

- Las acciones seguras requieren que los usuarios estén autenticados.
- Las credenciales son privilegios de seguridad agrupados bajo un nombre y que permiten organizar la seguridad en grupos.

Para restringir el acceso a una acción se crea y se edita un archivo de configuración YAML llamado `security.yml` en el directorio `config/` del módulo. En este archivo, se pueden especificar los requerimientos de seguridad que los usuarios deberán satisfacer para cada acción o para todas (`all`) las acciones.

La sintaxis YAML utilizada en el archivo security.yml permite restringir el acceso a usuarios que tienen una combinación de credenciales, usando asociaciones de tipo AND y OR. Con estas combinaciones, se pueden definir flujos de trabajo y sistemas de manejo de privilegios muy complejos

3.6.3 Métodos de Validación y Manejo de Errores

La clase `Action` presenta métodos mágicos para el manejo de errores y la validación de los datos para cada una de las acciones que posee o para todas en general. Estas funciones se nombran `validateMiAccion()` y `handleErrorMiAccion()` para el caso de una acción en específico y en caso de no encontrar `handleErrorMiAccion()` se remite a la función de la clase `handleError()`,

este método es genérico para manejar todos los errores de la clase. En caso contrario Symfony retorna `sfView::ERROR` que lanza la plantilla `miAccionError.php`.

El flujo de trabajo en la validación de los datos y el manejo de los errores se muestra en el Anexo IX .

Las validaciones de los datos enviados por el cliente en un formulario se realizan mediante archivos YML, esto permite que el código de validación sea menos repetitivo en los métodos de validación y la ejecución de los mismos se realice de forma más segura. Estas validaciones se realizan en el archivo `acción.yml` que se encuentra en el directorio `modules/modulo/acción/valídate/` y se refiere a el nombre de la acción de dicho módulo. La ejecución de las validaciones se realizan antes de la llamada a la función `validateMyAccion`, así sea positivo o negativo su resultado. Después del resultado de la validación de un formulario es posible mostrar los datos incorrectos al usuario en los mismos campos del mismo junto con mensajes de error que muestran donde están los datos incorrectos.

Las validaciones del lado del cliente se manejan desde el Framework EXTJS, seleccionado por la arquitectura para crear la interfaz de las aplicaciones. Este provee un grupo de validadores para los distintos tipos de componentes de los formularios.

3.6.4 Filtros

Symfony cuenta con un sistema de validación muy efectivo basándose en filtros, estás validaciones se encargan de los procesos de seguridad de las acciones entre otras tareas. Symfony de hecho procesa cada petición como una cadena de filtros ejecutados de forma sucesiva. Cuando el Framework recibe una petición, se ejecuta el primer filtro. En algún punto, llama al siguiente filtro en la cadena, luego el siguiente, y así sucesivamente. Cuando se ejecuta el último filtro, los filtros anteriores pueden finalizar, y así hasta el filtro de sfRenderingFilter que siempre es el primero por definición del núcleo de Symfony.

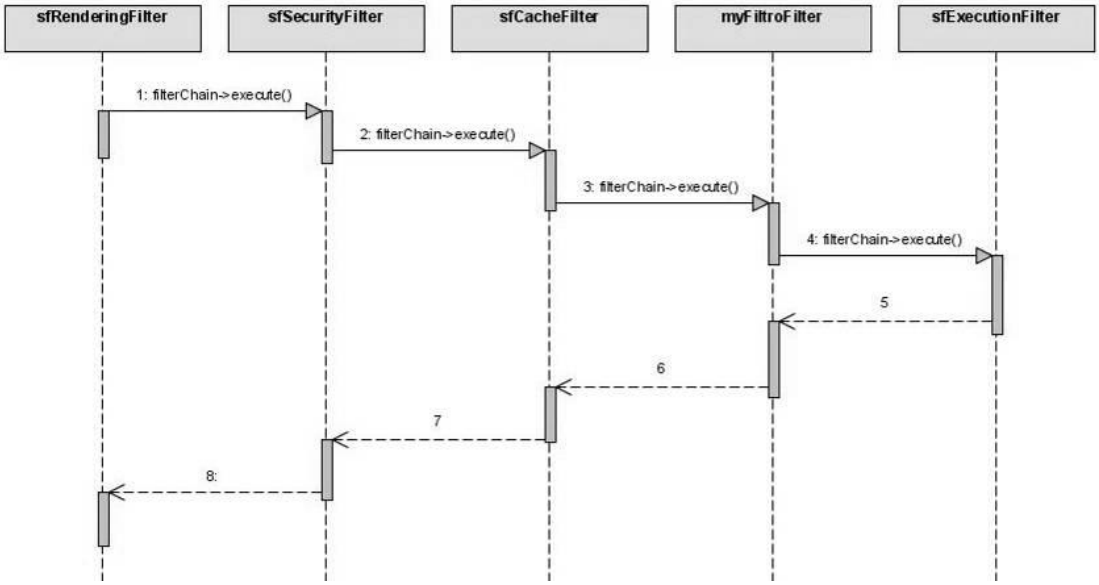

Fig. # 20: Secuencia de ejecución de los Filtros en Symfony

Los filtros declarados por los desarrolladores son ubicados en la cadena de filtros que ejecuta Symfony, por lo que cada filtro se divide en tres partes, el código que se ejecuta antes de la llamada al método `$filterChain->execute()` y el código que se ejecuta después de la llamada a la misma función.

Los filtros no son utilizados solo con fines de seguridad para las aplicaciones. También tienen otras utilidades como la reutilización de código, ya que son pedazos de código que se ejecutan siempre antes de la llamada a las acciones de un módulo o una aplicación.

3.6.5 Mecanismo de escape

Para evitar XSS (cross-site scripting) en los datos ingresados en las plantillas de forma dinámica por usuarios maliciosos es necesario tomar ciertas medidas en las aplicaciones, una de estas técnicas son los mecanismos de escape. Symfony provee de ciertos componentes para evitar que este proceso sea un método repetitivo, mediante la utilización de los parámetros `escaping_strategy` y `escaping_method` en el archivo de configuración `miaplicacion/config/settings.yml`.

La utilización de estos elementos que permiten el escape en Symfony lo que hace es utilizar la función `htmlentities()` de PHP en todas las variables mostradas en la plantilla y el Layout. Los mecanismos de escape incluyen además las salidas de los arreglos y objetos. Ya que todos los valores

de estos tipos de datos utilizados en los elementos de la vista son decorados con el tipo de dato `sfOutputEscaperArrayDecorator` y `sfOutputEscaperObjetctDecorator` respectivamente para facilitar su manipulación en los mecanismos de escape. Los métodos de estas clases contienen un parámetro adicional que permite modificar el valor de `escaping_method` de forma dinámica para una salida en específico.

3.6.6 SQL Injection. Uso de Criteria

Otros de los problemas de seguridad más clásicos son los ataques de Inyección SQL. Symfony también promueve un mecanismo para escapar de esta amenaza mediante el uso de las clases `Criteria` que están disponibles en el modelo.

A pesar de que el modelo en Symfony permite utilizar consultas SQL de manera pura se debe construir las consultas utilizando la clase `Criteria`. Ya que las utilidades de esta clase permite crear las consultas tan complejas como se necesite y todos los valores pasados son escapados para evitar el ataque de Inyección SQL. El uso de `Criteria` trae otras ventajas como la optimización del código SQL para el gestor de Base de Datos que se esté utilizando y la obtención de los resultados en un arreglo de Objetos del tipo seleccionado.

3.6.7 Sistema de Enrutamiento

Los sistemas de Enrutamiento utilizados por Symfony, brindan ciertas garantías de seguridad en cuanto a ocultar el sistema de directorios de la aplicación. Symfony maneja un sistemas de URL “limpias”, es decir, que estas están asociadas a la acción que ejecutan. La utilización de este sistema tiene varias ventajas desde el punto de vista de la seguridad como las que se muestran a continuación:

- La seguridad se implementa a nivel de acciones, no a nivel de páginas. Es decir, se evita la necesidad de incluir al comienzo de cada página un script que valide si el usuario tiene permisos para acceder a la página, ya que esto puede ser violado. La única manera de acceder a los scripts es mediante el controlador frontal, que este a su vez se encarga de validar si la acción solicitada por el cliente es accesible por el mismo mediante la utilización de filtros. Al no

existir otras vías de entrada a la aplicación, se garantiza que el eslabón más débil de la aplicación tenga toda la seguridad orientada hacia él.

- Al utilizar el sistema de enrutamiento de Symfony se evita mostrar la estructura interna del directorio de la aplicación. Otro aspecto importante en la seguridad de una aplicación web.
- Los parámetros pasados por GET como parte de la dirección de una petición no contienen información trascendental para la integridad de la aplicación como los nombres de los las variables que se están enviando, ya que solamente se muestra el resultado de las mismas, gracias a la configuración del archivo *routing.yml* que contiene el formato de las variables que se pasarán a la petición. Esto permite hacer las URL más seguras para la aplicación y personalizables para el usuario.
- El uso de este tipo de enrutamiento permite que cualquier URL no reconocida se redirige a una página especificada por el programador y los usuarios no pueden navegar por el directorio raíz del servidor mediante la prueba de diferentes URL. La razón es que no se visualiza el nombre del script utilizado o el de sus parámetros.

Otras ventajas de la utilización del enrutamiento de URL limpias son:

- Las URL tienen significado y ayudan a los usuarios a decidir si la página que se cargará al pulsar sobre un enlace contiene lo que esperan. Un enlace puede contener detalles adicionales sobre el recurso que enlaza.
- Las URL que aparecen en los documentos impresos son más fáciles de escribir y de recordar.
- La URL se puede convertir en una especie de línea de comandos, que permita realizar acciones u obtener información de forma intuitiva. Este tipo de aplicaciones son las que más rápidamente utilizan los usuarios más avanzados.
- Se puede modificar el aspecto de la URL y el del nombre de la acción o de los parámetros de forma independiente y con una sola modificación. En otras palabras, es posible empezar a programar la aplicación y después modificar el aspecto de las URL sin estropear completamente la aplicación.
- Aunque se modifique la estructura interna de la aplicación, las URL pueden mantener su mismo aspecto hacia el exterior. De esta forma, las URL se convierten en persistentes y pueden ser añadidas a los marcadores o favoritos.
- Cuando los motores de búsqueda indexan un sitio web, suelen tratar de forma diferente (incluso saltándoselas) a las páginas dinámicas (las que acaban en .php, .asp, etc.) Así que si se formatean las URL de esta forma, los buscadores creen que están indexando contenidos

estáticos, por lo que generalmente se obtiene una mejor indexación de las páginas de la aplicación.

Internamente el Framework interpreta las URI normalmente, pero estás son interpretadas para el usuario como las URL limpias, una manera de realizar convertir las URI a URL es mediante la utilización del HELPER `url_for()`. Para imprimir links de manera dinámica en las aplicaciones se utiliza este HELPER que trasforma las URI a URL para los usuarios.

3.7 Logs

Satisfacer los requerimientos no funcionales de un sistema es un proceso bastante complicado, pues va más allá del propio desarrollo de software. Para lograr que las aplicaciones tengan un alto grado de usabilidad y optimización es necesario mantener las aplicaciones bajo una auditoría constante que permita conocer el comportamiento de los usuarios con el mismo. Pues aunque los sistemas deben ser desarrollados para prever cualquier tipo de conducta por parte de los usuarios, este requerimiento es muy difícil de cumplir. Para facilitar este trabajo se utilizan herramientas que permitan monitorear el comportamiento de la aplicación bajo distintas condiciones, que durante el proceso de desarrollo y prueba es muy difícil de simular, como son los altos niveles de conexiones, concurrencia de los datos, flujos de datos, por solo citar algunos. De esta manera lograr que mediante este mecanismo de auditoría sobre estos componentes las versiones posteriores de los productos desarrolladas por el PSTA puedan mejorar en rendimiento, usabilidad y eficiencia. Una de las herramientas utilizadas para el proceso de optimización de las aplicaciones es la utilización de los Logs (o trazas) que dejan las aplicaciones durante su ejecución.

La importancia de los Logs va más allá del proceso de desarrollo de las aplicaciones. Es necesario conocer como está enmarcado el comportamiento de la aplicación incluso luego de haber logrado una aplicación estable y aprobada por los procesos de prueba realizados ya en el entorno de instalación. Pues se hace necesario que el administrador del sistema disponga de las herramientas necesarias para poder manejar los posibles errores que la aplicación pueda lanzar o algún tipo de comportamiento inesperado, así como que los desarrolladores puedan controlar lo que va mal en la aplicación en el momento de auditar su comportamiento y lograr mejores versiones posteriores.

3.7.1 Logs de PHP

La configuración del archivo PHP.ini brinda la posibilidad de mantener un histórico de los diferentes tipos de errores y mensajes ocurren durante la ejecución del código. Esto brinda la posibilidad de conocer en qué momento ha ocurrido un error en el sistema desde el núcleo de el código PHP, y las posibles causas que lo ha originado, además de información sobre el script donde se ha lanzado la excepción. Estos mensajes de error se pueden mostrar tanto en pantalla como en los archivos de Logs de PHP, la configuración para manejar estos registros se muestra en el Anexo X

Durante el proceso de desarrollo es conveniente mostrar los errores en pantalla para que los programadores conozcan los errores que lanza la aplicación durante las pruebas realizadas en esta fase. Luego de la implantación del sistema, se recomienda dejar que se registren los errores en un archivo de Logs para el posterior análisis de los mismos.

Desde las aplicaciones de Symfony es también posible modificar los valores de algunas de estas variables mediante archivos de configuración. En `settings.yml` es posible modificar el valor de `error_reporting` definiendo una variable del mismo nombre, con el código del nivel de error que se desea manejar.

3.7.2 Logs de Symfony

Cuando se utiliza un Framework como Symfony, el núcleo de la aplicación está compuesto por las clases del Framework, por lo que para conocer lo que sucede dentro de dichas clases es necesario utilizar herramientas que permitan mostrarle a los desarrolladores todos los eventos ocurridos en el sistema, ya sea durante el proceso de desarrollo o posterior.

Symfony además de los Logs que brinda PHP tiene su propio sistema de Logs. Estos registran tanto como el desarrollador desee registrar dentro del núcleo del Framework o desde cualquier punto de la aplicación, lo que brinda un control muy riguroso y a la vez flexible sobre lo que se está realizando o de cómo se va a utilizar posteriormente.

3.7.3 Configuración de los Logs

El control de los logs de Symfony se lleva por aplicación y por entorno, se almacenan en el directorio `miproyecto/log/`. Se crea un archivo para cada una de las aplicaciones y para cada entorno con el nombre `miaplicacion_entorno.log`, donde se almacenan dichos registros. Los errores que se registran en los Logs de Symfony se definen al igual que en PHP mediante niveles, estos niveles son `emerg`, `alert`, `crit`, `err`, `warning`, `notice`, `info` y `debug`. El registro de logs para cada una de las aplicaciones se configura en `miaplicacion/config/logging.yml`, en este archivo se definen los niveles de profundidad de registro de los errores para cada entorno de trabajo de la aplicación.

3.7.4 Mensajes Personalizados

Los mensajes de Logs no solo son mostrados por el núcleo del Framework, además por el desarrollador, ya que se puede realizar un registro en cualquier parte de la aplicación que este lo decida. Ya sea para controlar los accesos a las acciones, a los módulos, etc. Para agregar un mensaje de Log desde una acción se utiliza:

```
$this->logMessage($mensaje, $nivel);
```

Fig. # 21: Imprimir un mensaje desde cualquier parte del controlador.

Para realizar un registro desde una plantilla, Layout u otro elemento de la vista se ejecuta mediante la llamada a un HELPER:

```
<?php use_helper('Debug') ?>
<?php log_message($mensaje, $nivel) ?>
```

Fig. # 22: Imprimir un mensaje desde cualquier parte de la vista.

Para guardar un log desde otro lugar de la aplicación se utiliza:

```
sfContext::getInstance()->getLogger()->info($mensaje);
```

Fig. # 23: Imprimir un mensaje desde cualquier otra parte de la aplicación.

También es posible personalizar las clases de errores definidas por Symfony, para esto las nuevas clases deben implementar la interfaz `sfLoggerInterfaz` y ser inicializadas en el archivo `config.php` de la aplicación en la que se va a utilizar.

3.7.5 Rotación y borrado de los Logs

Para realizar un buen análisis del contenido de los archivos de Logs, es necesario separarlos en fragmentos más pequeños, ya que el tamaño de estos archivos puede llegar a alcanzar los varios Megas de memoria en pocos días y ser abiertos por la aplicación puede ser una tarea tediosa y que entorpezca el correcto funcionamiento del sistema. Para evitar estos problemas los Logs además de ser separados por aplicaciones y entornos como se comentó anteriormente son separados por fechas y se van moviendo a otra carpeta donde se puede realizar un mejor análisis de los mismos sin entorpecer la continuidad del proceso de registro. Este proceso es llamado "rotación de los Logs" y en Symfony se realiza mediante la configuración del archivo `logging.yml` de los valores `period` y `history`. Los valores de estas variables especifican cada cuantos días se realizará el proceso de rotación y cuantas copias de seguridad se irán guardando respectivamente. Otra manera de realizar esta tarea es de forma manual mediante el comando `log-rotate`, este comando solo se ejecuta para las aplicaciones que tienen el valor de `rotate` en `logging.yml` en `on`.

```
prod:
  rotate: on
  period: 7 ## Los archivos se rotan cada 7 días
  history: 10 ## Se mantiene un histórico de 10 archivos
```

Fig. # 24: Ejemplo de configuración para la rotación automática en el archivo logging.yml.

Otra forma es especificar la aplicación y el entorno en el momento de la llamada a `log-rotate` para especificar a qué archivo se le desea realizar el proceso de rotación.

```
> symfony log-rotate miaplicacion prod
```

Fig. # 25: Ejemplo de comandos para la rotación manual de una aplicación.

El proceso de rotación por sí trae implícito el borrado de los archivos que son rotados, pero hay otra forma de realizar el borrado de los archivos de forma manual, es mediante el comando `log-purge`, al igual que `log-rotate` solo se ejecuta en los archivos para los que la propiedad correspondiente está en `on` por lo que se debe evitar tener esta opción (`purge`) en el valor de `on` para evitar accidentes en los que se pierdan los registros.

Hasta ahora se ha visto la importancia que brindan los Logs desde el punto de vista de la optimización y monitoreo del comportamiento de las aplicaciones pero no se puede quitar mérito al uso de estos durante el desarrollo de las mismas. Pues son una herramienta que brinda la posibilidad de desarrollar los sistemas en un tiempo menor y con una mayor calidad, dado que posibilita identificar una mayor cantidad de errores en el menor tiempo posible y este aspecto es primordial en el desarrollo de software.

3.8 Conclusiones

En el capítulo se explica el funcionamiento interno del Framework Symfony para una mejor comprensión de cómo implementar sistemas bajo esta arquitectura. Así como las ventajas de optimización, seguridad y extensión que brinda utilizar Symfony.

CONCLUSIONES

Se ha desarrollado una solución Arquitectónica para el Polo Sistemas Tributarios y de Aduanas para resolver las necesidades de contar con una arquitectura que permita desarrollar productos de manera rápida y flexible, pero siguiendo los estándares de desarrollo y las mejores prácticas de programación logrando así alcanzar productos de alta calidad, usabilidad y seguridad. Lo que permite cumplir con los Requisitos No Funcionales descritos para las aplicaciones de este tipo.

Se ha descrito la solución propuesta basándose en las principales características de las definiciones arquitectónicas descritas en la Fundamentación Teórica y las restricciones del cliente para los sistemas de gestión de procesos aduanales. Par esta solución se ha seleccionado el Framework para PHP Symfony, para el Acceso a Datos el Framework de persistencia Propel y para la validación de los datos del usuario y el diseño de la interfaz Web el Framework para JavaScript, Ext JS.

RECOMENDACIONES

Se recomienda:

- Continuar el estudio de las versiones superiores de los Frameworks Symfony, Propel y Doctrine para facilitar el desarrollo de aplicaciones en el momento en que la migración de la Base de Datos lo permita.

- Aumentar los Requisitos No Funcionales de manera más específica a medida que se identifiquen más productos a desarrollar, permitiendo así poder acercar la Línea Base Arquitectónica a las necesidades reales de los productos.

- Continuar en el estudio de los estándares a aplicar para la comunicación entre sistemas de gestión de procesos aduanales para lograr incorporarlos a los productos que manejen cada uno de los procesos específicos.

- Aumentar las opciones de seguridad mediante la implementación de componentes que manejen la seguridad en las aplicaciones basadas en esta arquitectura de manera vertical, ya que la seguridad es un aspecto que siempre se debe mejorar en este tipo de sistemas.

- Implementar una arquitectura de dominio específico en los temas aduanales que incluya un grupo de componentes estándares que pueden ser utilizados para facilitar la implementación de sistemas vinculados a los procesos que se desarrollan en una aduana.

ANEXOS

Anexo I

Sistemas Automatizados Utilizados en la Aduana

Antes de comenzar a trabajar en colaboración con la Universidad de las Ciencias Informáticas la Aduana General de la República producía o utilizaba algunos de los Software que le servían para la gestión de sus procesos, pero estos estaban en diversos lenguajes, plataformas y Sistemas de Bases de Datos.

Nombre	Propósito	Lenguaje y Plataforma	SBD
SIDUNEA	Sistema de despacho que da la ONU como sistema estándar.	ABAL[61] / Unix Santa Cruz[62]	…
AFORO	Sistema de despacho postal.	FoxPro[63] / MS-DOS[64]	FoxPro
Tripulante	Sistema de despacho tripulante.	Developer 2000[65] / Windows	Oracle[66]
SACADA	Sistema automatizado de control de decomiso aduanal.	Visual Basic[67] / Windows	Access[68]
SACY	Sistema automatizado de control de yates.	FoxPro / Windows	FoxPro
SAPIA	Sistema automatizado de personas de interés aduanal.	FoxPro / Windows	FoxPro
SICA	Sistema automatizado de control de aplazados.	FoxPro / MS-DOS	FoxPro
SIREN	Sistema integral de recursos humanos	Visual Basic / Windows	Access

Tabla 1: Algunos de los sistemas que utiliza la Aduana General de la República de Cuba y las tecnologías que utilizan.

Anexo II

Comparación entre Frameworks de PHP

a. Tendencia al uso de Frameworks de PHP en 2008.

Con la ayuda de Gloogle Trends se ha realizado un diagrama de las tendencias a nivel mundial en el uso de tres importantes Frameworks que se presentan en este trabajo: CakePHP, Symfony y Zend Framework para lo que va de año 2008. Los resultados muestran que en el mundo CakePHP es el más utilizado gracias a lo sencillo que es de utilizar para realizar aplicaciones rápidas y tener una línea de aprendizaje muy alta, seguido de Symfony y Zend Framework respectivamente, estos dos últimos se encuentran muy parejos.

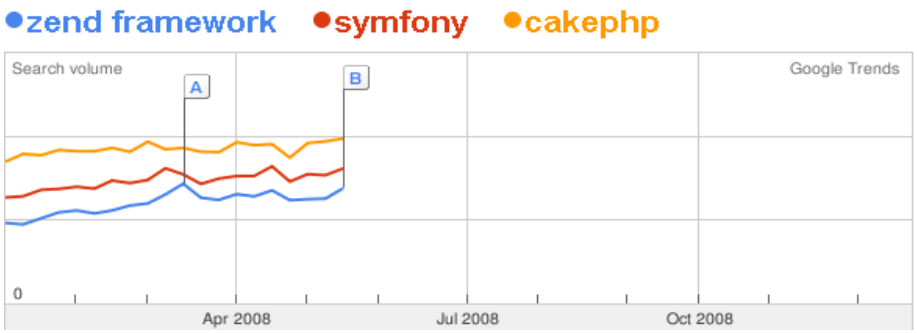

Fig. # 26: Tendencia a utilizar CakePHP, Symfony y Zend Framework a nivel mundial

La comparación fue llevada a Estados Unidos y Francia, países nativos de estas tres herramientas y los resultados son mostrados a continuación:

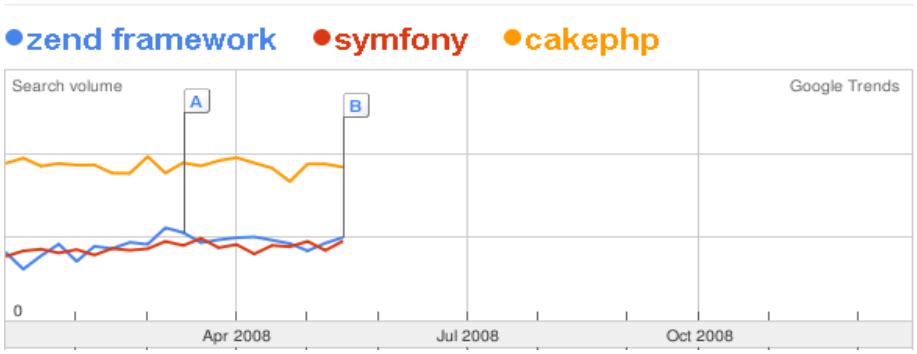

Fig. # 27: Tendencia a utilizar CakePHP, Symfony y Zend Framework en Estados Unidos

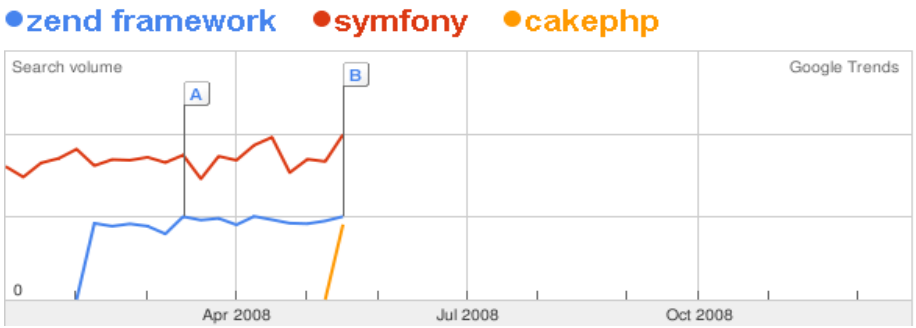

Fig. # 28: Tendencia a utilizar CakePHP, Symfony y Zend Framework en Francia.

Los resultados son halagadores para los seguidores de Symfony pues se mantiene muy cerca de Zend Framework en los Estados Unidos y en Francia la ventaja sobre ambos Frameworks es total, incluso Zend Framework y CakePHP se han comenzado a utilizar hace muy poco en este país.

b. Tabla de comparación entre Frameworks de PHP

Características	CakePHP	Symfony	Zend FW
Arquitectura de aplicaciones			
Incorporación del patrón Modelo Vista Controlador Orientado a Objetos.	X	X	X
Operaciones CRUD (Create, Retrieve, Update y Delete) asociadas a patrón Active Record.	X	X	
Mapeado de objetos a bases de datos relacionales (ORM).	X	X	
Independiente del manejador de base de datos.	X	X	X
Estructura por defecto para aplicaciones (scaffolding).	X	X	
Archivos de configuración de la aplicación.	.PHP	.YML	.PHP
Acceso vía web			
Despachador de peticiones HTTP.	X	X	X
Generación de URLs amigables.	X	X	X
Implementación de código HTML			

Uso de plantillas en PHP.	X	X	X
Posibilidad de uso de plantillas en Smarty.	X	X	
Implementación de ayudantes de plantillas (HELPERs).	X	X	
Seguridad			
Manejo propio de sesiones por usuarios.	X	X	X
Manejo de privilegios de acceso a secciones de la aplicación (Access Control List).	X	X	X
Verificación de la salida generada en HTML por procesamiento de peticiones (Data Sanitization).	X	X	
Validación de Formularios	X	X	X
Prevención XSS	X	X	X
Prevención SQL Injection		X	
Usabilidad y acceso rápido			
Almacenamiento en caché de las vistas.	X	X	
Almacenamiento en caché de configuración de las aplicaciones.		X	
Soporte implícito para AJAX	X	X	
Documentación para su uso			
Manual de referencia.	X	X	X
Documentación de la Interfaz de Programación de Aplicaciones (API).	X	X	X
Herramientas de programación			
Generación de código PHP.	X	X	
Herramientas de prueba y depuración.		X	X
Interfaz de línea de comandos para la creación y mantenimiento de aplicaciones.	X	X	
Almacenamiento de logs de funcionamiento del Framework.		X	
Extensibilidad y opciones adicionales			

Integración con otras herramientas a través de plugins.	X	X	
Implementación propia de llamadas Asyncronous JavaScript and XML (AJAX)	X	X	
Soporte para Web Services.		X*	X
Soporte para envío de correo electrónico.		X	X
Generación de contenido sindicalizado (RSS).		X	X
Generación de archivos PDF.	X	X	X
Soporte para internacionalización y localización de contenidos.		X	X
Soporte PHP			
Soporte para PHP4.	X		
Soporte para PHP5.	X	X	X
Características adicionales			
Licencias libres.	MTI	MTI	BSD
Comunidad activa de usuarios.	X	X	X

Tabla 2: Comparación entre CakePHP, Symfony y Zend Framework en distintos aspectos.

* Mediante la utilización del plugins ckWebServicesPlugins.

Anexo III

Comparación entre Acciones y Componentes

La tabla a continuación muestra las diferencias entre los componentes y las acciones en cuanto a los diferentes archivos que estos necesitan para su funcionamiento.

Convención	Acciones	Componentes
Archivo de la lógica	actions.class.php	components.class.php
Clase de la que hereda la lógica	sfActions	sfComponents
Nombre de los métodos	executeMiAccion()	executeMiComponente()
Nombre del archivo de presentación	miAccionSuccess.php	_miComponente.php

Tabla 3: Comparación entre las características de las acciones y los componentes.

Anexo IV

Conversión del código SQL por Criteria.

La utilización de la clase Criteria facilita la interpretación del lenguaje de consultas a la Base de Datos por los programadores entre otras ventajas, a continuación se muestra una tabla que describe la equivalencia entre las llamadas a métodos de esta clase y las sentencias SQL.

SQL	Criteria
WHERE columna = valor	->add(columna, valor);
WHERE columna <> valor	->add(columna, valor, Criteria::NOT_EQUAL);
Otros operadores de comparación	
> , <	Criteria::GREATER_THAN, Criteria::LESS_THAN
>=, <=	Criteria::GREATER_EQUAL, Criteria::LESS_EQUAL
IS NULL, IS NOT NULL	Criteria::ISNULL, Criteria::ISNOTNULL
LIKE, ILIKE	Criteria::LIKE, Criteria::ILIKE
IN, NOT IN	Criteria::IN, Criteria::NOT_IN
Otras palabras clave de SQL	
ORDER BY columna ASC	->addAscendingOrderByColumn(columna);
ORDER BY columna DESC –	>addDescendingOrderByColumn(columna);
LIMIT limite	->setLimit(limite)
OFFSET desplazamiento	->setOffset(desplazamiento)
FROM tabla1, tabla2 WHERE tabla1.col1 = tabla2.col2	->addJoin(col1, col2)
FROM tabla1 LEFT JOIN tabla2 ON tabla1.col1 = tabla2.col2	->addJoin(col1, col2,
Criteria::LEFT_JOIN) FROM tabla1 RIGHT JOIN tabla2 ON tabla1.col1 = tabla2.col2	->addJoin(col1, col2, Criteria::RIGHT_JOIN)

Tabla 4: Equivalencia entre SQL y la clase Criteria

Anexo V

Directorios utilizados por Symfony

a. Por proyecto.

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).
`batch/`	Contiene los scripts de PHP que se ejecutan mediante la línea de comandos o mediante la programación de tareas para realizar procesos en lotes (*batch processes*).
`cache/`	Contiene la versión cacheada de la configuración y 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 pre-procesados.
`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, etc.
`doc/`	Contiene la documentación del proyecto, formada por tus propios documentos y por la documentación generada por PHPDOC.
`lib/`	Almacena las clases y librerías externas. Se suele guardar todo el código común a todas las aplicaciones del proyecto. El subdirectorio model/ 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 5: Descripción de los directorios utilizados por un Proyecto

b. Por aplicación

Directorio	Descripción
config/	Contiene un montón de archivos de configuración creados con YAML. Aquí se almacena la mayor parte de la configuración de la aplicación, salvo los parámetros propios del Framework. También es posible redefinir en este directorio los parámetros por defecto si es necesario.
i18n/	Contiene todos los archivos utilizados para la internacionalización de la aplicación, sobre todo los archivos que traducen la interfaz. La internacionalización también se puede realizar con una base de datos, en cuyo caso este directorio no se utilizaría.
lib/	Contiene las clases y librerías utilizadas exclusivamente por la aplicación.
modules/	Almacena los módulos que definen las características de la aplicación.
templates/	Contiene las plantillas globales de la aplicación, es decir, las que utilizan todos los módulos. Por defecto contiene un archivo llamado Layout.php, que es el *Layout* principal con el que se muestran las plantillas de los módulos.

Tabla 6: Descripción de los directorios utilizados por una aplicación.

c. Por módulo.

Directorio	Descripción
actions/	Normalmente contiene un único archivo llamado actions.class.php y que corresponde a la clase que almacena todas las acciones del módulo. También es posible crear un archivo diferente para cada acción del módulo.
config/	Puede contener archivos de configuración adicionales con parámetros exclusivos del módulo
lib/	Almacena las clases y librerías utilizadas exclusivamente por el módulo
templates/	Contiene las plantillas correspondientes a las acciones del módulo. Cuando se crea un nuevo módulo, automáticamente se crea la plantilla llamada indexSuccess.php
validate/	Contiene archivos de configuración relacionados con la validación de formularios.

Tabla 7: Descripción de los directorios utilizados por un módulo.

Anexo VI

Archivos de configuración de Symfony

Los archivos de configuración que se deben tener en cuenta en la creación de cualquier aplicación y/o proyecto son los siguientes:

Archivo	Nivel de Definición	Descripción
`config.php`	Proyecto	Se trata del primer archivo que se ejecuta con cada petición o comando. Contiene la ruta a los archivos del Framework y se puede modificar si se ha realizado una instalación personalizada. Se pueden añadir instrucciones *define* de PHP al final de este archivo para que esas constantes sean accesibles en cualquier aplicación del proyecto.
`databases.yml`	Proyecto	Contiene la definición de los accesos a bases de datos y las opciones de conexión de cada acceso.
`Properties.ini`	Proyecto	Contiene algunos parámetros que utiliza la herramienta de línea de comandos, como son el nombre del proyecto y las opciones para conectar con servidores remotos.
`rsync_exclude.txt`	Proyecto	Indica los directorios que se excluyen durante la sincronización entre servidores. El Capítulo 16 también incluye una explicación de este archivo.
`schema.yml`	Proyecto	Contiene la representación del modelo de datos relacional del proyecto.
`propel.ini`	Proyecto	Toda la información que utiliza Propel para conocer las librerías utilizadas, ubicación de los archivos generados, Base de Datos, etc.
`Controlador Frontal`	Aplicación	Es el primer script que se ejecuta cuando se accede a una aplicación por un entorno dado. Contiene información referente a la aplicación.
`app.yml`	Aplicación	Contiene la configuración específica de la aplicación, variables globales que son utilizadas en el negocio de la aplicación, etc.

config.php	Aplicación	Este archivo inicia la ejecución de la aplicación, ya que realiza todas las inicializaciones necesarias para que la aplicación se pueda ejecutar. En este archivo se puede personalizar la estructura de directorios de la aplicación y se pueden añadir constantes que manejan las aplicaciones.
databases.yml	Aplicación	Redefine las credenciales de Acceso a la Base de Datos para la aplicación.
schema.yml	Aplicación	Contiene la representación del modelo de datos relacional de la aplicación. Este archivo es opcional, ya que toda la información puede estar almacenada en el archivo a nivel de proyecto.
factories.yml	Aplicación	Symfony incluye sus propias clases para el manejo de la vista, de las peticiones, de las respuestas, de la sesión, etc. No obstante, es posible definir otras clases propias para realizar estas tareas.
filters.yml	Aplicación	En este archivo se definen los filtros que se van a procesar.
logging.yml	Aplicación	Permite definir el nivel de detalle con el que se generan los archivos de log, utilizados para el mantenimiento y la depuración de las aplicaciones.
routing.yml	Aplicación	Almacena las reglas de enrutamiento, que permiten transformar las URL habituales de las aplicaciones web en URL limpias y sencillas de recordar.
settings.yml	Aplicación	Contiene las principales opciones de configuración de una aplicación. Entre otras, permite especificar si la aplicación utiliza la internacionalización, el idioma por defecto de la aplicación, el tiempo de expiración de las peticiones y si se activa o no la cache.
view.yml	Aplicación	Establece la estructura inicial de la vista por defecto: el nombre del Layout, el título de la página y las etiquetas <meta>; las hojas de estilos y los archivos JavaScript que se incluyen; el Content-Type por defecto, etc. También permite definir el valor por defecto de las etiquetas <title> y <meta>.
i18n.yml	Aplicación	En este archivo se establecen las opciones generales de traducción de páginas.
filters.yml	Módulo	Se identifican los filtros que se deben ejecutar en el módulo.
view.yml	Módulo	Permite configurar las vistas de una o de todas las acciones del

		módulo. Redefine las opciones del archivo view.yml de la aplicación.
generator.yml	Módulo	Se utiliza para los módulos utilizados en el scaffolding y para las partes de administración creadas de forma automática.
module.yml	Módulo	Contiene la configuración de la acción y otros parámetros específicos del módulo
security.yml	Módulo	Permite restringir el acceso a determinadas acciones del módulo. En este archivo se configura que una página solamente pueda ser accedida por los usuarios registrados o por un grupo de usuarios registrados con permisos especiales.
Validación	Módulo	Son archivos YML que se encuentran en el directorio /valídate y se encargan de configurar la validación de los datos entrados por distintos formularios.
Configuración del Núcleo de Symfony		
autoload.yml		Contiene las opciones relativas a la carga automática de clases. Esta opción permite utilizar clases propias sin necesidad de incluirlas previamente en el script que las utiliza, siempre que esas clases se encuentren en algunos directorios determinados.
constants.php		Define la estructura de archivos y directorios por defecto.
bootstrap_compile.yml		Define la lista de clases que se incluyen al iniciar la aplicación.
core_compile.yml		Define la lista de clases que se incluyen al procesar una petición.
config_handlers.yml		Permite añadir o modificar los manejadores de archivos de configuración.
php.yml		Este archivo se utiliza para comprobar que las directivas del archivo de configuración de PHP php.ini tienen los valores adecuados y permite redefinirlas si hace falta.

Tabla 8: Listado de los Archivos de Configuración de Symfony

Anexo VII

Características en la Configuración

A partir del Apache 2.0 no es necesaria la configuración del fichero http.conf, ya que toda la configuración del mismo se hace en el mismo default, como se muestra a continuación:

a. Configuración del fichero default del apache2.

```
NameVirtualHost *
<VirtualHost *>
      ServerAdmin admin@domain.com

      DocumentRoot /var/www/htdocs
      <Directory />
            Options FollowSymLinks
            AllowOverride None
      </Directory>
      <Directory /var/www/htdocs/>
            Options Indexes FollowSymLinks MultiViews
            AllowOverride None
            Order allow, deny
            Allow from all
      </Directory>

      Alias /rrhh /var/www/rrhh/web
      Alias /est /var/www/estructura/web

      ScriptAlias /cgi-bin/ /usr/lib/cgi-bin/

      <Directory "/usr/lib/cgi-bin">
            AllowOverride None
            Options ExecCGI -MultiViews +SymLinksIfOwnerMatch
            Order allow,deny
            Allow from all
      </Directory>

      ErrorLog /var/log/apache2/error.log
      LogLevel warn
      CustomLog /var/log/apache2/access.log combined
      ServerSignature On
```

```
    Alias /doc/ "/usr/share/doc/"

    <Directory "/usr/share/doc/">
        Options Indexes MultiViews FollowSymLinks
        AllowOverride None
        Order deny,allow
        Deny from all
        Allow from 127.0.0.0/255.0.0.0 ::1/128
    </Directory>

</VirtualHost>
```

En cambio para habilitar los módulos del apache, como son el `mod-write` o el modulo del PHP 5 se realiza directo en la consola, simplemente introduciendo los comandos `a2enmod <nombre del modulo>`, para habilitar el modulo necesario, y `a2dimod <nombre del modulo>`, para deshabilitarlo, por ejemplo al habilitar los módulos necesarios quedaría:

```
a2enmod mod-write
a2enmod php5
```

b. Configuración del PHP.ini

En cuanto a la configuración del PHP 5, se haría en el PHP.ini donde se puede utilizar el que se crea al compilar el PHP 5.0.4, ya que se habilitan los módulos necesarios, como son los soporte para Oracle 8i, con las librerías de oci8, PostgreSQL quedando así:

```
[PHP]
engine = On
zend.ze1_compatibility_mode = Off
short_open_tag = On
asp_tags = Off
precision = 12
y2k_compliance = On
output_buffering = Off
zlib.output_compression = Off
implicit_flush = Off
unserialize_callback_func=
serialize_precision = 100
allow_call_time_pass_reference = On
```

```
safe_mode = Off
safe_mode_gid = Off
safe_mode_allowed_env_vars = PHP_
safe_mode_protected_env_vars = LD_LIBRARY_PATH
expose_php = On
max_execution_time = 30     ; Maximum execution time of each script, in
seconds
max_input_time = 60 ; Maximum amount of time each script may spend parsing
request data
memory_limit = 16M      ; Maximum amount of memory a script may consume
(16MB)
variables_order = "EGPCS"
register_globals = Off
register_long_arrays = On
register_argc_argv = On
auto_globals_jit = On
post_max_size = 8M
magic_quotes_gpc = On
magic_quotes_runtime = Off
magic_quotes_sybase = Off
default_mimetype = "text/html"
enable_dl = On
file_uploads = On
upload_max_filesize = 2M
allow_url_fopen = On
allow_url_include = Off
default_socket_timeout = 60

[Syslog]
define_syslog_variables  = Off

[mail function]
SMTP = localhost
smtp_port = 25

[SQL]
sql.safe_mode = Off

[ODBC]
odbc.allow_persistent = On
odbc.check_persistent = On
odbc.max_persistent = -1
odbc.max_links = -1
odbc.defaultlrl = 4096
odbc.defaultbinmode = 1
```

```
[PostgresSQL]
pgsql.allow_persistent = On
pgsql.auto_reset_persistent = Off
pgsql.max_persistent = -1
pgsql.max_links = -1
pgsql.ignore_notice = 0
pgsql.log_notice = 0

[Session]
session.save_handler = files
session.use_cookies = 1
session.name = PHPSESSID
session.auto_start = 0
session.cookie_lifetime = 0
session.cookie_path = /
session.serialize_handler = php
session.gc_divisor     = 100
session.gc_maxlifetime = 1440
session.bug_compat_42 = 1
session.bug_compat_warn = 1
session.entropy_length = 0
session.cache_limiter = nocache
session.cache_expire = 180
session.use_trans_sid = 0
session.hash_function = 0
session.hash_bits_per_character = 4
url_rewriter.tags = "a=href,area=href,frame=src,input=src,form=,fieldset="

[soap]
soap.wsdl_cache_enabled=1
soap.wsdl_cache_dir="/tmp"
soap.wsdl_cache_ttl=86400
```

Anexo VIII

Estructura de un Plugins

A continuación se muestra la estructura física de la composición de un Plugins para Symfony.

```
nombrePlugin/
  config/
    *schema.yml           // Esquema de datos
    *schema.xml
    config.php            // Configuración específica del plugins
  data/
    generator/
      sfPropelAdmin
        */                // Temas para el generador de administraciones
          templates/
          skeleton/
    fixtures/
      *.yml               // Archivos de fixtures
    tasks/
      *.php               // Tareas de Pake
  lib/
    *.php                 // Clases
    helper/
      *.php               // Helpers
    model/
      *.php               // Clases del modelo
  modules/
    */                    // Módulos
      actions/
        actions.class.php
      config/
        module.yml
        view.yml
        security.yml
      templates/
        *.php
      validate/
        *.yml
  web/
    * // Archivos estáticos
```

Fig. # 29: Estructura de Directorios de un Plugins

Anexo IX

Flujo de Trabajo de los procesos de Validación

El flujo de trabajo en la validación de los datos y el manejo de los errores se muestra a continuación.

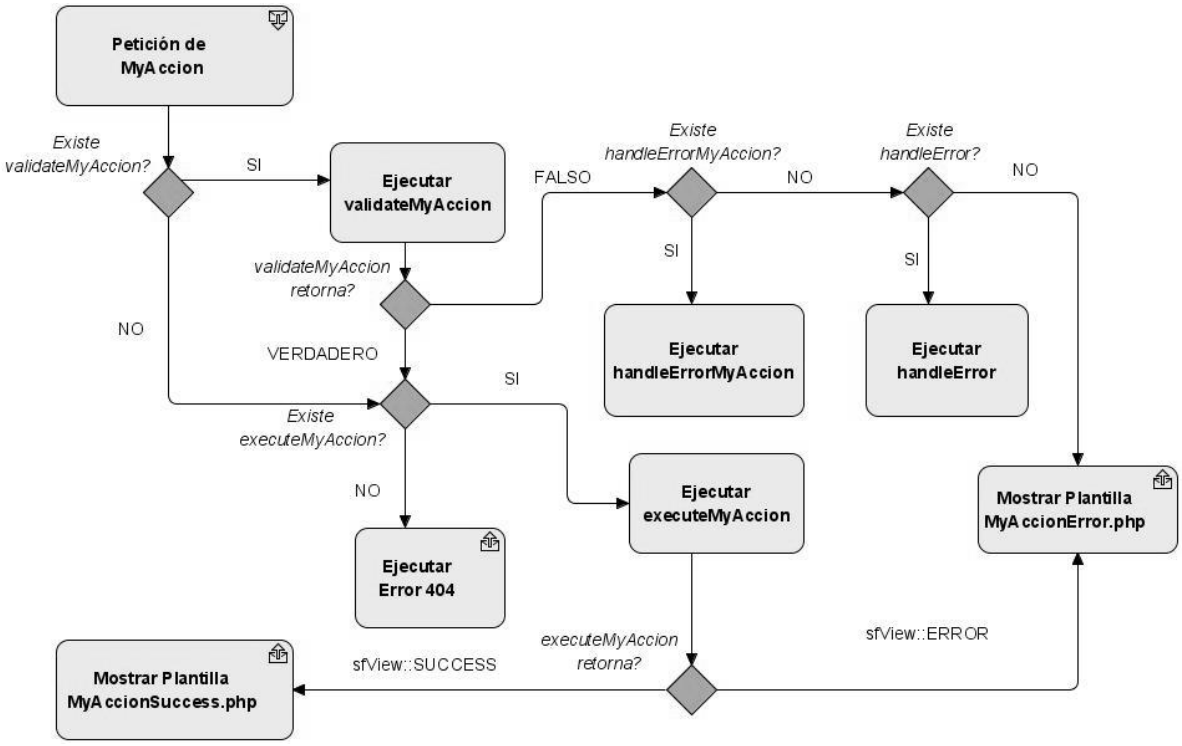

Fig. # 30: Flujo de Trabajo del proceso de Validación y Manejo de Errores para una acción.

Anexo X

Variables de PHP para el manejo de Errores

error_reporting	Variable que controla en nivel de registro de los errores, los niveles son:
E_ALL	Todos los tipos de errores
E_ERROR	Errores fatales de ejecución
E_WARNING	Mensajes de Alerta en tiempo de ejecución (Warnings).
E_PARSE	Errores de PARSE en tiempo de compilación.
E_NOTICE	Noticias en tiempo de ejecución.
E_CORE_ERROR	Errores ocurridos durante la inicialización del núcleo de PHP.
E_CORE_WARNING	Alertas ocurridos durante la inicialización del núcleo de PHP.
E_COMPILE_ERROR	Errores ocurridos en tiempo de compilación.
E_COMPILE_WARNING	Aletas ocurridas en tiempo de compilación.
E_USER_ERROR	Errores generados por el usuario
E_USER_WARNING	Alertas generadas por el usuario
E_USER_NOTICE	Noticias generadas por el usuario
display_errors	Define si los errores serán mostrados en pantalla. Los valores definibles son On y Off.
log_errors	Define si los errores serán registrados en un archivo de tipo Log. Los valores definibles son On y Off.
error_log	Especifica el archivo donde serán guardados los Logs de errores.

Tabla 9: Listado de las variables para manejar los errores de PHP.

BIBLIOGRAFÍA

1. **Garlan, D. y Shaw, M.** *"An Introduction to Software Architecture".* Pittsburgh : School of Computer Science, Carnegie-Mellon University, 1994.

2. **Kaisler, S. H.** *"Software Paradigms".* s.l. : John Wiley & Sons, Inc., 2005.

3. **Perry, D. y Wolf, A. L.** *"Foundations for the Study of Software Architecture".* s.l. : ACM Software Engineering Notes., 1992. págs. 40-52.

4. **Garlan, D. y Shaw, M.** *"Software Architecture: Perspectives on an Emerging Discipline".* Saddle River : Prentice Hall, 1996.

5. "Software Engineering Institute | Carnegie Mellon®.". [En línea] http://www.sei.cmu.edu.

6. **Jacobson, Ivar, Booch, Grady y Rumbaugh, James.** *"The Unified Software Development Process".* Reading : Addison-Wesley, 1999.

7. **Bass, L., Clements, P. y Kazman, R.** *"Software Architecture in Practice".* 2dn Edition. Reading : Addison-Wesley, 2003.

8. **Szyperski, C.** *"Component-Based Software: Beyond Object-Oriented Programming".* Pergamon : Addison-Wesley, 1998.

9. **Yorio, Ing. Dario.** "Identificación y Clasificación de Patrones en el Diseño de Aplicaciones Móviles". [Trabajo de Diploma para alcanzar el Magíster en Ingeniería de Software. U.N.L.P.].

10. *"Understanding and Using Patterns in Software Development.".* **Riehle, D. y Züllighoven, H.** 1996, Theory and Practice of Object Systems.

11. **Buschmann, F.** *"Pattern-Oriented Software Architecture: A System of Patterns".* s.l. : John Wiley and Sons Ltd, 1996. ISBN 0-471-95869-7.

12. **Johnson, R. E. y Helm, R.** *"Design Patterns: Elements of Reusable Object-Oriented Software".* s.l. : Addison-Wesley Pub Co., 1995.

13. *"Designing Reusable Classes.".* **Johnson, R. E. y Foote, B.** 1988, Journal of Object-Oriented Programming, pág. 35.

14. **Johnson, R. E. y Russo, V. F.** *"Reusing Object-Oriented Designs".* s.l. : University of Technical Report, 1991. UIUCDCS 91-1696.

15. "CakePHP: the rapid development php Framework.". [En línea] http://www.cakephp.org.

16. **Gutierrez, Andrés F.** *"Libro de Kumbia | Porque programar debería ser más fácil".* 2007.

17. **Zend Tecnologies Ltd.** "Zend Framework". [En línea] http://www.framework.zend.com.

18. **Potencier, Fabien y Zaninotto, François.** "Symfony | Open-Source PHP Web Framework". [En línea] Sitio Oficial del proyecto Symfony. http://www.symfony-project.org.

GLOSARIO DE TÉRMINOS

[1] SUA: Sistema Único de Aduanas.

[2] AGR: Aduana General de la República.

[3] CADI: Centro de Automatización para la Información y la Dirección

[4] UCI: Universidad de las Ciencias Informáticas.

[5] GUI Widget: Símbolo gráfico de interfaz que permite interacción entre el usuario y el ordenador

[6] Glue Code: En términos de programación, es el código que no realiza ninguna funcionalidad para satisfacer algún requerimiento del programa en cuestión.

[7] API: (Application Programming Interface) Interfaz de Programación de Aplicaciones) es el conjunto de funciones y procedimientos que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.

[8] GoF: (Gang of Four) Grupo de los Cuatro, se refiere al grupo de los autores de este tipo de patrones de diseño.

[9] Ruby: Lenguaje de programación reflexivo y orientado a objetos(lenguaje interpretado), creado por el programador japonés Yukihiro "Matz" Matsumoto, quien comenzó a trabajar en Ruby en 1993, y lo presentó públicamente en 1995. Combina una sintaxis inspirada en Python, Perl con características de programación orientada a objetos similares a Smalltalk.

[10] Python: Lenguaje de programación creado por Guido van Rossum en el año 1990. Es comparado habitualmente con TCL, Perl, Scheme, Java y Ruby. En la actualidad Python se desarrolla como un proyecto de código abierto, administrado por la Python Software Foundation.

[11] Perl: Lenguaje de programación diseñado por Larry Wall creado en 1987. Perl toma características del C, del lenguaje interpretado shell (sh), AWK, sed, Lisp y, en un grado inferior, de muchos otros lenguajes de programación.

[12] Java: Lenguaje de programación orientado a objetos desarrollado por Sun Microsystems a principios de los años 90. El lenguaje en sí mismo toma mucha de su sintaxis de C y C++.

[13] Open Source MTI License: Licencia de Software Libre originaria de el Instituto de Tecnología de Massachusetts. Es una licencia permisiva, lo que significa que permite la reutilización de la propiedad del software, compatible con la licencia GPL.

[14] PEAR: (PHP Extensión and Application Repository) Es un entorno de desarrollo y sistema de distribución para componentes de código PHP. El proyecto PEAR fue fundado por Williams G. Molina G. en 1999 para promover la reutilización de código que realizan tareas comunes.

[15] Ruby on Rails: También conocido como RoR o Rails es un Framework de aplicaciones web de código abierto escrito en el lenguaje de programación Ruby, siguiendo el paradigma de la arquitectura Modelo Vista Controlador (MVC).

[16] AJAX: (Asynchronous JavaScript And XML) JavaScript asíncrono y XML, es una técnica de desarrollo web para crear aplicaciones interactivas. Esta se ejecutan en el cliente y mantiene comunicación asíncrona con el servidor en segundo plano. Esto significa aumentar la interactividad, velocidad y usabilidad en la misma.

[17] HELPER: Funciones especiales que permiten imprimir código para realizar ciertas operaciones más complejas, ya sea en HTML, JavaScript, AJAX o incluso en PHP.

[18] Javascript: es un lenguaje de programación interpretado, es decir, que no requiere compilación, utilizado principalmente en páginas web, con una sintaxis semejante a la del lenguaje Java y el lenguaje C.

[19] RSS: Es parte de la familia de los formatos XML desarrollado específicamente para todo tipo de sitios que se actualicen con frecuencia y por medio del cual se puede compartir la información y usarla en otros sitios web o programas. A esto se le conoce como redifusión web o sindicación web (una traducción incorrecta, pero de uso muy común).

[20] Caché: Es un conjunto de datos duplicados de otros originales, con la propiedad de que los datos originales son costosos de acceder, normalmente en tiempo, respecto a la copia en el caché.

[21] Versión BETA: Una versión beta o lanzamiento beta representa generalmente la primera versión completa del programa informático o de otro producto, que es probable que sea inestable pero útil para que las demostraciones internas y las inspecciones previas seleccionen a clientes.

[22] Scaffold: Es un patrón de desarrollo que permite crear capturas de formularios y vistas de forma dinámica según los atributos de una entidad en el modelo de datos.

[23] Callbacks: Funciones del lado del cliente o del servidor que se ejecutan automáticamente antes o despues de una acción del usuario.

[24] Namespaces: Un namespace es un contexto en el que un grupo de uno o más identificadores pueden existir. Un identificador definido en un namespace está asociado con ese namespace.

[25] Prototype: Es un Framework escrito en JavaScript que se orienta al desarrollo sencillo y dinámico de aplicaciones web.

[26] JSON: (JavaScript Object Notation) Notación de Objetos JavaScript, es un formato ligero para el intercambio de datos. JSON es un subconjunto de la notación literal de objetos de JavaScript que no requiere el uso de XML.

[27] Script.Aculo.Us: Es una librería JavaScript que permite el uso de controles AJAX, drag 'n drop, y otros efectos visuales en una página web

[28] ActiveRecord: Es un enfoque al problema de acceder a los datos de una base de datos. Una fila en la tabla de la base de datos (o vista) se envuelve en una clase, de manera que se asocian filas únicas de la base de datos con objetos del lenguaje de programación usado.

[29] Licencia BSD: es la licencia de software otorgada principalmente para los sistemas BSD (Berkeley Software Distribution). Pertenece al grupo de licencias de software Libre. La licencia BSD al contrario que la GPL permite el uso del código fuente en software no libre.

[30] CRUD: (Create, Retrieve, Update, Delete) Es usado para referirse a las funciones básicas en bases de datos o la capa de persistencia en un sistema de software.

[31] ORM: (Object Relational Mapping) Mapeo de Objetos Relacional es una tecnica de programación para convertir tipos de datos incompatibles entre sistemas de bases de datos y lenguajes orientados a objetos.

[32] Doctrine: Es un Mapeo de Objetos Relacional (ORM) para PHP 5.2.3+ que se ubica en el tope de la capa de abstracción de la base de datos.

[33] Table Data Gateway: Patrón de diseño, referente a la capa de acceso a datos o modelo que permite que un objeto actúe como una puerta de enlace a una tabla de la base de datos.

[34] Row Data Gateway: Patrón de diseño, referente a la capa de acceso a datos o modelo que permite que un objeto actúe como una puerta de enlace a un registro en particular de una tabla.

[35] PDO: **P**HP **D**ata **O**bjects es una extensión que provee una capa de abstracción de acceso a datos para PHP 5.

[36] Mojavi: Es un marco MVC para PHP.

[37] YAML: (**Y**AML **A**in't a **M**arkup **L**anguage) Es un formato de serialización de datos legible por humanos inspirado en lenguajes como XML, C, Python y Perl. Fue propuesto por Clark Evans en 2001.

[38] Rake: Herramienta utilziada por Rails para automatizar tareas.

[39] Django: Es un Framework de desarrollo web de código abierto, escrito en Python, que cumple en cierta medida el paradigma del Modelo Vista Controlador.

[40] Prado: Es un Framewrok para aplicaciones Web en PHP 5 basado en componenetes y manejo de eventos.

[41] Cocoa: Es un conjunto de Frameworks orientados a objetos que permiten el desarrollo de aplicaciones nativas para Mac OS X.

[42] Apple Inc: Es una empresa estadounidense de tecnología informática fundada en 1976 por Steve Jobs y Steve Wozniak.

[43] Test::More: Framework de Testeo para aplicaciones desarrolladas en Perl.

[44] Lime: Framework de Testeo utilizado por Symfony para realizar las pruebas a las soluciones.

[45] TinyMCE: Es un editor WYSIWYG(What You See Is What You Get) para HTML de código abierto que funciona completamente en JavaScript y se distribuye gratuitamente bajo licencia LGPL

[46] Código Abierto: es el término con el que se conoce al software distribuido y desarrollado libremente.

[47] SVN: Subversion es un software de sistema de control de versiones. Es software libre bajo una licencia de tipo Apache/BSD y se le conoce también como svn por ser ese el nombre de la herramienta de línea de comandos.

[48] phpDocumentor: Es un estándar formal para comentar código PHP. Es una adaptación de Javadoc para el lenguaje PHP.

[49] CLI: es un programa informático que actúa como Interfaz de usuario para comunicar al usuario con el sistema operativo mediante líneas de comandos.

[50] Mixin: Es una clase que ofrece cierta funcionalidad para ser heredada por una subclase, pero no está ideada para ser autónoma.

[51] Layout: En Symfony se conoce como Layout el enmarcado que se le da a las páginas.

[52] Admin Generator: Herramienta que brinda Symfony para crear una interfaz de administración de datos generada a partir de la lectura de un Modelo específico.

[53] Data Access Objects: (DAO) es un componente de software que suministra una interfaz común entre la aplicación y uno o más dispositivos de almacenamiento de datos, tales como una Base de datos o un archivo.

[54] Rsync: Es una aplicación para sistemas de tipo Unix que ofrece transmisión eficiente de datos incrementales comprimidos y cifrados.

[55] XSS: (Cross-site scripting) es un ataque basado en la explotación de vulnerabilidades del sistema de validación de HTML incrustado.

[56] CSRF: (Cross-site request forgery) Es un ataque basado en la explosión de los Sitios web. Es muy similar a XSS.

[57] DHTML: (Dynamic HTML) Designa el conjunto de técnicas que permiten crear sitios web interactivos utilizando una combinación de lenguaje HTML estático, un lenguaje interpretado en el lado del cliente (como JavaScript), el lenguaje de hojas de estilo en cascada (CSS) y la jerarquía de objetos de un DOM.

[58] DOM: (Document Object Model) Modelo en Objetos para la representación de Documentos. Es un modelo computacional a través de la cual los programas y scripts pueden acceder y modificar dinámicamente el contenido, estructura y estilo de los documentos HTML y XML.

[59] LGPL: (Lesser General Public License) Licencia Pública General Reducida. es una licencia de software creada por la Free Software Foundation.

[60] Tab: En las interfaces de las aplicaciones se refiere a las pestañas que se utilizan para aumentar la navegabilidad.

[61] ABAL (Advanced Business Application Language) es un lenguaje de cuarta generación, propiedad de SAP. Plataforma que permitía a las grandes corporaciones construir aplicaciones de negocios para gestión de materiales y finanzas.

[62] Santa Cruz Operation (SCO) era una compañia de software con base en Santa Cruz, California, EE. UU., la cual era conocida por vender 3 variantes de Unix para procesadores Intel X86.

[63] FoxPro (acrónimo de FoxBASE Professional) es un lenguaje de programación orientado a procedimientos (procedures), a la vez que un Sistema Gestor de Bases de datos.

[64] MS-DOS: Siglas de MicroSoft Disk Operating System, Sistema operativo de disco de Microsoft.

[65] Oracle Developer Suite: Es una suite de herramientas de desarrollo liberadas por Oracle Corporation.

[66] Oracle: Es un sistema de gestión de base de datos relacional

[67] Visual Basic: Es un lenguaje de programación desarrollado por Alan Cooper para Microsoft.

[68] Microsoft Access: Es un programa Sistema de gestión de base de datos relacional creado y modificado por Microsoft para uso personal de pequeñas organizaciones.