UNIVERSIDAD DE BELGRANO

Las tesinas de Belgrano

Facultad de Tecnología Informática
Ingeniería en Informática

Implementación de un framework para el desarrollo de aplicaciones web utilizando patrones de diseño y arquitectura MVC/REST

Nº 354 Edgardo Roberto Zulian

Tutor: Víctor M. Rodríguez

Departamento de Investigaciones
Febrero 2010

Índice:

Introducción

El lenguaje de Internet para la creación de documentos electrónicos más conocido es el HTML, que es con el que se define la estructura y contenido de una página web. Consta de una serie de etiquetas (markups) predefinidas que permiten construir documentos que contienen títulos, párrafos y listas de texto, tablas, imágenes y otros elementos para la presentación de la información.

Con el avance de la tecnología y la aparición por ejemplo del XML (Extensible Markup Language), metalenguaje extensible creado por el Wide Word Web Consortium (W3C) que posibilita la creación de nuevos lenguajes o sublenguajes (XHTML, RSS, ATOM, MathML), surge la necesidad de que aquellos sistemas que simplemente ó solamente podían generar código HTML, sean modificados ó reemplazados por nuevos sistemas que si sean capaces de generar tantos formatos de salida como sean requeridos para poder orientar los negocios hacia las nuevas necesidades del mercado, de la forma más sencilla posible.

Hoy en día muchos sistemas que manejan información a través de Internet, necesitan no solamente poder generar códigos en HTML para representarse en los navegadores de los usuarios, sino que también existe la necesidad de intercambiarla entre diferentes plataformas ó sistemas para lograr que la misma pueda ser interpretada según la tecnología involucrada (celulares, feeds de noticias, etc.) pero siempre partiendo de una misma base desde donde se genere dicha información.

Para resolver la problemática descripta anteriormente, este proyecto trata sobre la implementación de un framework que permita desarrollar aplicaciones en las que pueda fácilmente separarse la lógica de negocios, el acceso de los datos y su forma de presentación de modo que la generación de información en diferentes formatos deje de ser un problema o una situación de difícil manejo que obligue a modificar los sistemas haciéndolos cada vez más complejos y repitiendo códigos y funcionalidades, agregándoles nuevas capas en su core cada vez que sea necesario generar una nueva salida.

1.1 Planteamiento del Problema

El proyecto que se presenta, ofrece un caso real que consiste en una empresa que se dedica a la distribución de contenidos audiovisuales, ya sea a través de portales de internet, ó directamente a terceros a través de interfaces donde los mismos pueden acceder a la información.
El framework actual, requiere día a día un mayor mantenimiento y agregados para que pueda seguir cumpliendo con su funcionalidad, debido a problemas de diseño (como ser su alto acoplamiento modular) esta tarea resulta cada vez más difícil y requiere de gente demasiado especializada para poder cumplir con dicha tarea. Asimismo se requiere un cambio en el modo de invocación a las interfaces expuestas a terceros, para que las mismas sean más simples y dinámicas.

Por lo tanto para realizar el proyecto, fue necesario plantear las limitaciones existentes en el framework actual, evaluando la posibilidad de su modificación y considerando también la opción de usar otros frameworks existentes de origen open source ó inclusive la creación de uno nuevo.

Será importante tener en cuenta los tiempos para cada una de las propuestas debido a que el éxito del proyecto depende en tener un framework consistente y funcional que pueda nutrir en el menor plazo posible de información a los primeros clientes Microsoft Silverlight, desarrollados en la misma empresa, para ser pioneros en el mercado de habla hispana.

También deberá considerarse que luego de implementado el framework, será necesaria la migración de las principales aplicaciones existentes en el framework anterior. Estas aplicaciones son las encargadas de exponer la información que es requerida por los nuevos clientes.

1.2 Objetivo

Reemplazar el framework actual, implementando un nuevo framework que permita separar la lógica de negocios, el acceso a datos y la capa de presentación, donde se puedan migrar las aplicaciones del framework anterior así como la generación de nuevas, sin depender de la capa de presentación, siendo está última manejada por el nuevo framework, en base a las necesidades del negocio.

1.3 Alcance

El desarrollo de la tesina comprende la implementación de un framework que permita incorporar aplicaciones donde pueda separase la lógica de negocio, el acceso a datos y la capa de presentación. De modo que puedan implementarse dos clientes desarrollados en Silverlight (nueva tecnología Microsoft) uno para redes sociales y otro para un nuevo portal web que sean alimentados por el nuevo sistema. Así mismo se deberán exponer las interfaces necesarias para seguir ofreciendo a terceros los contenidos audio visuales.

1.4 Límites

El desarrollo del proyecto deberá tener en cuenta las tecnologías disponibles por la empresa y la solución deberá ser encontrada dentro de ese ámbito, por lo tanto el sistema deberá:

- Funcionar en servidores Linux (distribución Debian) que corran Apache como webservers.
- Ser desarrollado en lenguaje PHP.
- Utilizar base de datos Oracle y basarse en el modelo de datos existente.
- Mantener la misma funcionalidad del framework existente, como ser: restricciones de acceso por país, por rangos de ip y manejo de idiomas.
- Implementar algún sistema de cacheo contra la base de datos para que sea performante.

1.5 Hipótesis

Es posible la implementación de un sistema que permita la generación y migración de aplicaciones del sistema anterior, separando la lógica del negocio, el acceso a datos y la capa de presentación, de modo que puedan generarse salidas en diferentes formatos según las necesidades del negocio.

1.6 Justificación del trabajo

Este proyecto implementará un framework que permitirá a la empresa poder orientar su negocio a las nuevas tecnologías que vayan surgiendo, dándole la adaptabilidad necesaria para generar la información en los formatos que sean requeridos sin tener que modificar su core, su acceso a datos ó su lógica de negocios. El nuevo sistema permitirá agregar nuevas capas ó lógicas de presentación requeridas evitando la duplicación innecesaria de código, minimizando los esfuerzos de implementación, facilitando los testeos y las tareas de mantenimiento. Una vez migradas las aplicaciones podrán generarse diferentes tipos de salida según lo que el cliente requiera. (xml, xaml, html, wap, etc.)

En el proyecto de implementación del framework, como en la construcción del portal plus.mixplay.tv el cual utiliza el framework implementado y se presentará como caso de prueba, me desempeñe en el rol de Project Leader, coordinando también tareas de investigación y supervisión de los programadores.

2. Tecnologías Involucradas

Las tecnologías principales aplicadas para el desarrollo y cumplimiento de los objetivos propuestos en el presente trabajo son:

Patrón de Diseño: MVC (Model-View-Controller).
Patrón Arquitectónico: REST
Framework: ZendFramework
Lenguaje: PHP5.
Sistema de Versionamiento: SVN.
Webserver: Apache
Base de Datos: Oracle 10g.
Sistema de Cache: Memcached.

De las tecnologías citadas, tal como se menciona en la sección límites del trabajo tanto el lenguaje PHP, el Webserver Apache y la base de datos Oracle 10g, no podrán cambiarse debido a que forman parte del know how y core de la empresa. El resto de las tecnologías involucradas y seleccionadas son explicadas en el siguiente apartado.

3. Marco Teórico

3.1 ¿Qué es un Patrón? [1]

Para definir el concepto de patrón nos trasladamos por un momento al ámbito de la arquitectura, donde el arquitecto Christopher Alexander creó el término lenguaje de patrón definiéndolo de la siguiente manera:

> "Cada patrón describe un problema que ocurre una y otra vez en nuestro entorno, para describir después el núcleo de la solución a ese problema, de tal manera que esa solución pueda ser usada más de un millón de veces sin hacerlo siquiera dos veces de la misma forma"
> Christopher Alexander

Como vemos, los patrones fueron inicialmente concebidos como una manera de formalizar la solución a problemas comunes a muchos proyectos de arquitectura. Más tarde, vista la eficacia de los patrones en el campo de la arquitectura, otras disciplinas los añadieron a su repertorio de herramientas.

La informática no ha sido una excepción y la adaptación de este concepto a la ingeniería del software ha dado lugar a lo que conocemos como patrones de software. Estos patrones capturan soluciones a situaciones de especial interés, bien por ser muy comunes, bien por ser especialmente complejas.

A modo de resumen podríamos decir que los patrones de software:

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

3.2 ¿Qué es MVC?[2]

MVC es un patrón de diseño que proviene de las siglas en inglés (Model View Controller / Modelo Vista Controlador). Una correcta implementación y uso de este patrón logran separar la lógica de negocios de la interfaz de usuario, resultando en aplicaciones donde es más fácil modificar tanto la apariencia visual como las reglas de negocio sin que la otra se vea afectada.

¿Por qué surge el patrón MVC?

El Problema:

Es muy frecuente que en un sistema se soliciten cambios a la interfaz. Los cambios a la interfaz deberían ser fáciles y efectuados en tiempo de ejecución. El cambio de interfaz no debería de tener consecuencias para el núcleo o core del código de la aplicación.

La Solución:

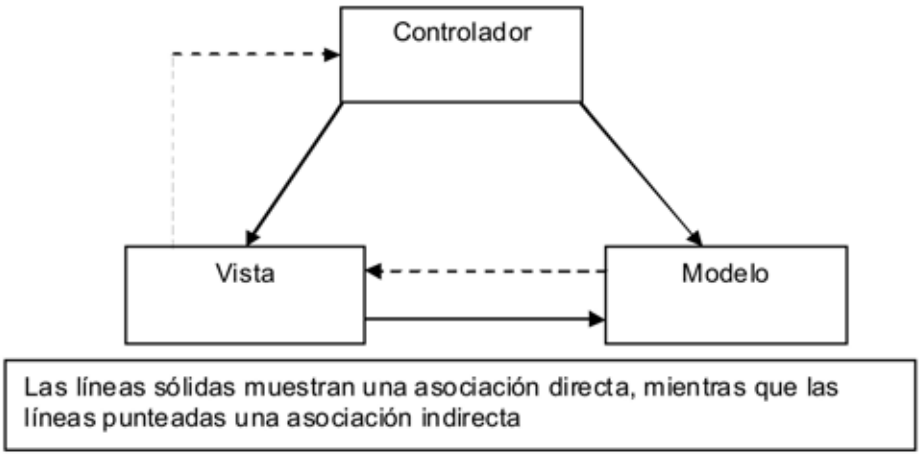
El sistema se divide en tres partes: procesamiento, entradas y salidas.

- El Modelo como la representación de la información (los datos) y las reglas de negocio usadas para manipular dichos datos, por lo tanto el modelo encapsula los datos y la funcionalidad de la aplicación.
- La Vista despliega la información contenida en el modelo como una salida en una forma que permita interacción, generalmente a través de una interfaz de usuario. También pueden existir múltiples vistas

1. Patrones de Diseño Software LIBRO: GOF Erich Gamma, Richard Helm, Ralph Jonson y John Vlissides.
2. MVC Blueprints: http://java.sun.com/blueprints/patterns/MVC-detailed.html

para un mismo modelo con diferentes propósitos. (Ej: una salida HTML, XML, RSS, XAML, etc.)
- El Controlador procesa y responde a acciones de eventos, generalmente son acciones del usuario y pueden invocar cambios en el modelo. El controlador está asociado a cada vista, recibe entradas que traduce en invocaciones de métodos del Modelo o de Vista. El usuario interactúa con el sistema solamente vía controladores.

Diagrama que representa al patrón MVC



Conclusión

Consideramos que a lo largo de la explicación acerca de para qué y cómo funciona el patrón MVC, queda más que justificada su elección para el framework que se necesita. Utilizando el patrón MVC se podrá implementar un sistema que permita desarrollar aplicaciones en las que pueda fácilmente separarse la lógica de negocios, el acceso de los datos y su forma de presentación.

3.3 ¿Que es REST?[3]

REST proviene de las siglas en inglés Representational State Transfer / Transferencia de Estado Representacional) y define un conjunto de principios arquitectónicos por los cuales se diseñan servicios web haciendo foco en los recursos del sistema, incluyendo cómo se accede al estado de dichos recursos y cómo se transfieren por HTTP hacia clientes escritos en diversos lenguajes. REST emergió en los últimos años como el modelo predominante para el diseño de servicios. De hecho, REST logró un impacto tan grande en la web que prácticamente logró desplazar a SOAP y las interfaces basadas en WSDL por tener un estilo bastante más simple de usar.

Los 4 principios de REST

Una implementación concreta de un servicio web REST sigue cuatro principios de diseño fundamentales:

- Utiliza los métodos HTTP de manera explícita.
- No mantiene estado.
- Expone URIs con forma de directorios.
- Transfiere XML, JavaScript Object Notation (JSON), o ambos.

A continuación se explica cada uno de estos principios

I. REST utiliza los métodos http de manera explicíta:

Una de las caraterísticas claves de los servicios web REST es el uso explícito de los métodos HTTP, siguiendo el protocolo definido por RFC 2616. Por ejemplo, HTTP GET se define como un método productor de datos, cuyo uso está pensado para que las aplicaciones cliente obtengan recursos, busquen

3. Explicación transcripta del site http://www.dosideas.com/java/314-introduccion-a-los-servicios-web-restful.html, con agregados personales.

datos de un servidor web, o ejecuten una consulta esperando que el servidor web la realice y devuelva un conjunto de recursos.

REST hace que los desarrolladores usen los métodos HTTP explícitamente de manera que resulte consistente con la definición del protocolo. Este principio de diseño básico establece una asociación uno-a-uno entre las operaciones de crear, leer, actualizar y borrar y los métodos HTTP. De acuerdo a esta asociación:

- Se usa POST para crear un recurso en el servidor
- Se usa GET para obtener un recurso
- Se usa PUT para cambiar el estado de un recurso o actualizarlo
- Se usa DELETE para eleminar un recurso

Sin embargo existen muchas APIs web que usan el método HTTP GET para ejecutar algo transaccional en el servidor; por ejemplo, agregar registros a una base de datos. En estos casos, no se utiliza adecuadamente el URI de la petición HTTP, o al menos no se usa “a la manera REST”. Si el API web utiliza GET para invocar un procedimiento remoto, seguramente se verá algo como lo siguiente:

GET /adduser?name=Juan HTTP/1.1

Este no es un diseño muy atractivo porque el método aquí arriba expone una operación que cambia de estado sobre un método HTTP GET. Dicho de otra manera, la petición HTTP GET de aquí arriba tiene efectos secundarios. Si se procesa con éxito, el resultado de la petición es agregar un usuario nuevo (en el ejemplo, Juan) a la base de datos. El problema es básicamente semántico. Los servidores web están diseñados para responder a las peticiones HTTP GET con la búsqueda de recursos que concuerden con la ruta (o el criterio de búsqueda) en el URI de la petición, y devolver estos resultados o una representación de los mismos en la respuesta, y no añadir un registro a la base de datos. Desde el punto de vista del protocolo, y desde el punto de vista de servidor web compatible con HTTP/1.1, este uso del GET es inconsistente.

Más allá de la semántica, el otro problema con el GET es que al ejecutar eliminaciones, modificaciones o creación de registros en la base de datos, o al cambiar el estado de los recursos de cualquier manera, provoca que las herramientas de caché web y los motores de búsqueda (crawlers) puedan realizar cambios no intencionales en el servidor, cómo se solucionó esto hasta ahora, agregándole un parámetro extra con un timestamp o un valor al azar que nunca se repita, de esta forma se evita el cacheo. Sin embargo una forma simple de evitar este problema es mover los nombres y valores de los parámetros en la petición del URI a tags XML. Los tags resultantes, una representación en XML de la entidad a crear, pueden ser enviados en el cuerpo de un HTTP POST cuyo URI de petición es el padre de la entidad.

Antes:

```
GET /adduser?name=Juan HTTP/1.1
```

Después:

```
POST /users HTTP/1.1
Host: server
Content-type: application/xml
<user>
    <name>Juan</name>
</user>
```

El método indicado arriba es un ejemplo de una petición REST: hay un uso correcto de HTTP POST y la inclusión de los datos en el cuerpo de la petición.

Luego, una aplicación cliente podría obtener una representación del recurso usando la nueva URI, sabiendo que al menos lógicamente el recurso se ubica bajo /users

```
GET /users/Juan HTTP/1.1
Host: server
Accept: application/xml
```

Es explícito el uso del GET de esta manera, ya que el GET se usa solamente para recuperar datos. GET es una operación que no debe tener efectos secundarios.

De forma similar un método web que realice actualizaciones a través del método HTTP GET, debería ser modificado para que haga uso explícito de los métodos HTTP, un enfoque REST sería enviar un HTTP PUT para actualizar el recurso, en vez de usar HTTP GET.

Antes:

```
GET /modifyuser?id=100&name=Pedro HTTP/1.1
```

Después:

```
PUT /users/100 HTTP/1.1
Host: server
Content'Type: application/xml
<user>
   <id>100</id>
   <name>Pedro</name>
</user>
```

Al usar el método PUT para reemplazar al recurso original se logra una interfaz más limpia que es consistente con los principios de REST y con la definición de los métodos HTTP. La petición PUT que se muestra arriba es explícita en el sentido que apunta al recurso a ser actualizado identificándolo en el URI de la petición, y también transfiere una nueva representación del recurso del cliente hacia el servidor en el cuerpo de la petición PUT, en vez de transferir los atributos del recurso como un conjunto de parámetros (nombre = valor) en el mismo URI de la petición.

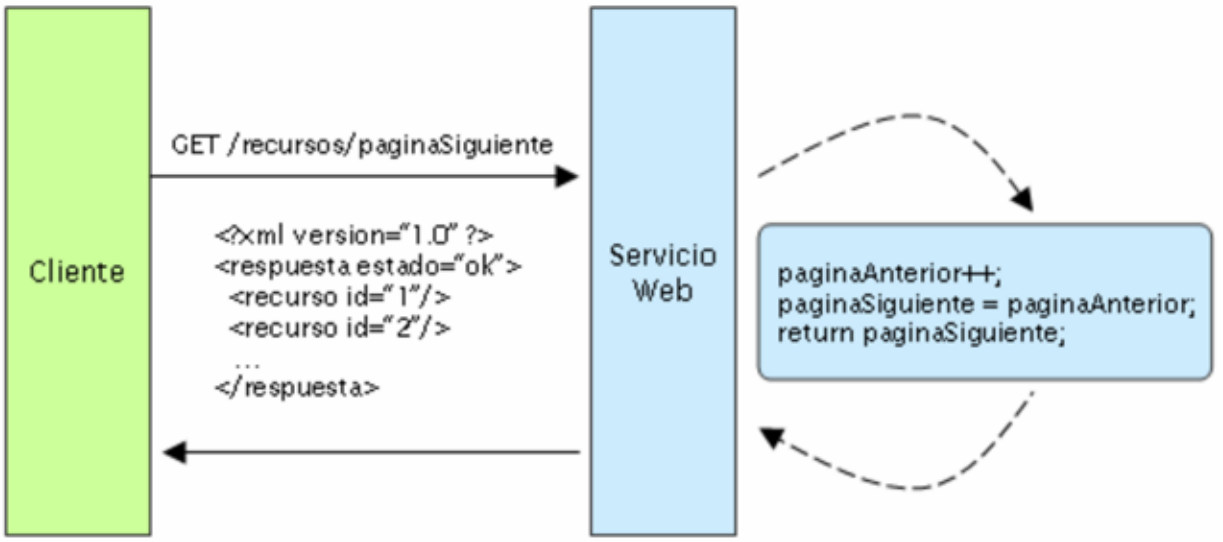
Como un principio de diseño general, ayuda seguir las reglas de REST que aconsejan usar sustantivos en vez de verbos en las URIs. En los servicios web REST, los verbos están claramente definidos por el mismo protocolo: POST, GET, PUT y DELETE. Idealmente, para mantener una interfaz general y para que los clientes puedan ser explícitos en las operaciones que invocan, los servicios web no deberían definir más verbos o procedimientos remotos, como ser /adduser y /modifyuser. Este principio de diseño también aplica para el cuerpo de la petición HTTP, el cual debe usarse para transferir el estado de un recurso, y no para llevar el nombre de un método remoto a ser invocado.

II. REST no mantiene estado

Para lograr una petición completa e independiente, de modo que el servidor no tenga que recuperar ninguna información de contexto o estado para procesar dicha petición, una aplicación o cliente de servicio web REST debe incluir en la llamada todos los parámetros y datos que sean necesarios para que el servidor pueda generar una respuesta. De esta manera, al no mantener estado se mejora el rendimiento de los servicios web y se simplifica el diseño e implementación de los componentes del servidor, ya que con la ausencia de los estados se elimina la necesidad de sincronizar los datos de la sesión con una aplicación externa.

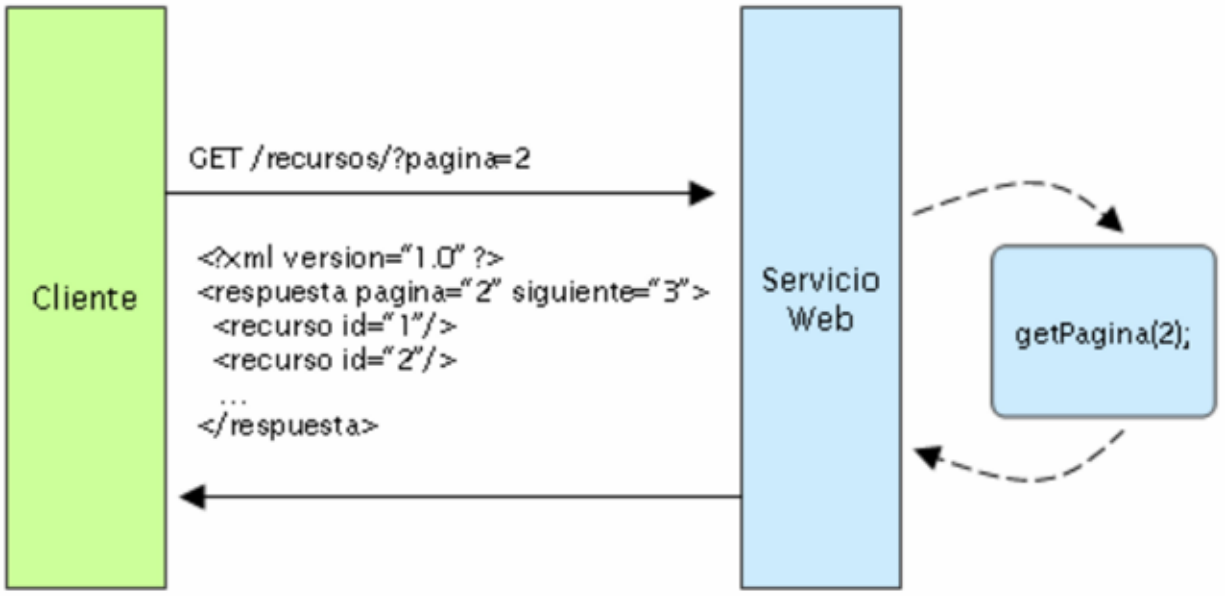
Servicios con estado versus servicios sin estado

La siguiente ilustración nos muestra un servicio con estado, desde el cual una aplicación realiza peticiones para la página siguiente, asumiendo que el servicio mantiene información sobre la última página que pidió el cliente. En un diseño con estado, el servicio incrementa y almacena en algún lugar (por ejemplo en una cookie ó en sesión) una variable paginaAnterior para poder responder a las peticiones.



Lo que sucede con los servicios que mantienen estado es que tienden a volverse complicados. Por ejemplo, en sistemas que utilizan PHP los estados tienden a manejarse generalmente con variables de sesión. En casos donde exista un pool o conjunto de servidores, como las sesiones son mantenidas por cada servidor, es necesario recurrir a otras tecnologías (por ejemplo balanceadores de carga), para asegurarse que cada usuario sea siempre atendido por el mismo servidor para que no pierda su actual sesión.

Por otro lado, los servicios sin estado son mucho más simples de diseñar, escribir y distribuir a través de múltiples servidores. Un servicio sin estado traslada la responsabilidad de mantener el estado al cliente de la aplicación. En un servicio web REST, el servidor es responsable de generar las respuestas y proveer una interfaz que le permita al cliente mantener el estado de la aplicación por su cuenta. Por ejemplo, en el mismo ejemplo de una petición de datos en múltiples páginas, el cliente debería incluir el número de página a recuperar en vez de pedir “la siguiente”, tal como se muestra en la siguiente figura:



Un servicio web sin estado genera una respuesta que se enlaza a la siguiente página del conjunto y le permite al cliente hacer todo lo que necesita para almacenar la página actual.

III. REST expone URIs con forma de directorios

Desde el punto de vista del cliente de la aplicación que accede a un recurso, la URI determina qué tan intuitivo va a ser el web service REST, y si el servicio va a ser utilizado tal como fue pensado al momento de diseñarlo. La tercera característica de los servicios web REST es justamente sobre las URIs.

Las URI de los servicios web REST deben ser intuitivas, hasta el punto de que sea fácil adivinarlas. Pensemos en las URI como una interfaz auto-documentada que necesita de muy poca o ninguna explicación o referencia para que un desarrollador pueda comprender a lo que apunta, y a los recursos derivados relacionados.

Una forma de lograr este nivel de usabilidad es definir URIs con una estructura al estilo de los directorios. Este tipo de URIs es jerárquica, con una única ruta raíz, y va abriendo ramas a través de las subrutas para exponer las áreas principales del servicio. De acuerdo a esta definición, una URI no es solamente una cadena de caracteres delimitada por barras, sino más bien un árbol con subordinados y padres organizados como nodos. Por ejemplo, en un servicio de hilos de discusiones que tiene varios temas, se podría definir una estructura de URIs como la siguiente:

http://misitio/discusion/temas/{tema}

La raíz, /discusion, tiene un nodo /temas como hijo. Bajo este nodo hay un conjunto de nombres de temas (como ser tecnología, actualidad, y otros), cada uno de los cuales apunta a un hilo de discusión. Dentro de una estructura como la mencionada, resulta más fácil e intuitivo recuperar hilos de discusión al tipear algo después de /temas/.

A continuación, se enumeran algunas guías generales al tener en cuenta al momento de crear URIs para un servicio web REST:

- Ocultar la tecnología usada en el servidor que aparecería como extensión de archivos (.jsp, .php, .asp), de manera de poder portar la solución a otra tecnología sin cambiar las URI.
- Mantener todo en minúsculas.
- Sustituir los espacios con guiones o guiones bajos (uno u otro).
- Evitar el uso de strings de consulta.

En vez de usar un 404 Not Found si la petición es una URI parcial, devolver una página o un recurso predeterminado como respuesta.

Las URI deberían ser estáticas de manera que cuando cambie el recurso o cambie la implementación del servicio, el enlace se mantenga igual. Esto permite que el cliente pueda generar “favoritos” o bookmarks. También es importante que la relación entre los recursos que está explícita en las URI se mantenga independiente de las relaciones que existen en el medio de almacenamiento del recurso.

IV. REST transfiere XML, JSON, o ambos

La representación de un recurso en general refleja el estado actual del mismo y sus atributos al momento en que el cliente de la aplicación realiza la petición. La representación del recurso son simples “fotos” en el tiempo. Esto podría ser una representación de un registro de la base de datos que consiste en la asociación entre columnas y tags XML, donde los valores de los elementos en el XML contienen los valores de las filas.

La última restricción al momento de diseñar un servicio web REST tiene que ver con el formato de los datos que la aplicación y el servicio intercambian en las peticiones/respuestas.

Los objetos del modelo de datos generalmente se relacionan de alguna manera, y las relaciones entre los objetos del modelo de datos (los recursos) deben reflejarse en la forma en la que se representan al momento de transferir los datos al cliente.

Esto permite que el servicio sea utilizado por distintos clientes escritos en diferentes lenguajes, corriendo en diversas plataformas y dispositivos. El uso de los tipos MIME y del encabezado HTTP Accept es un mecanismo conocido como negociación de contenido, el cual le permite a los clientes elegir qué formato de datos puedan leer, y minimiza el acoplamiento de datos entre el servicio y las aplicaciones que lo consumen.

Conclusión

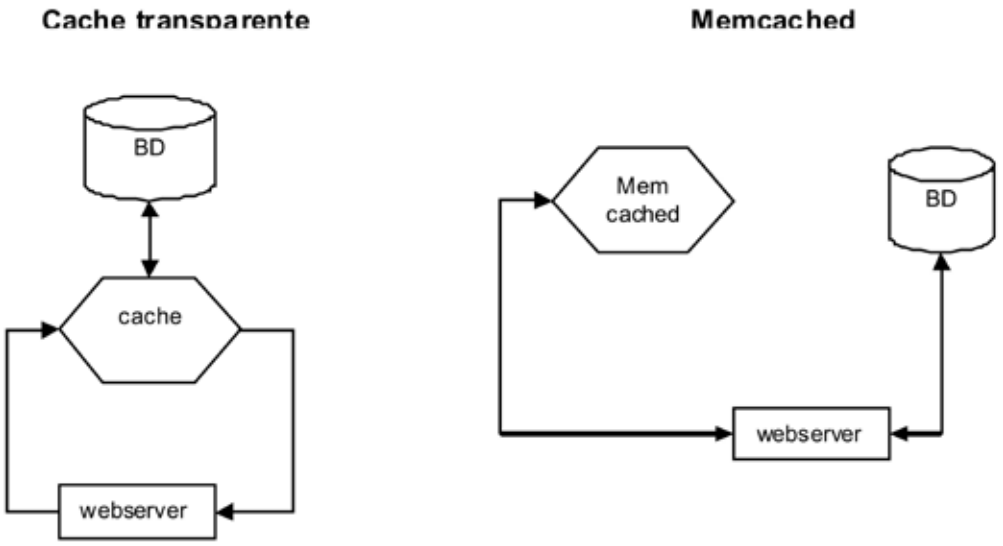
Si bien REST no siempre es la mejor opción, por ejemplo surgen preguntas acerca del manejo de la seguridad, como controlar validaciones ó autenticaciones de usuarios ó sistemas donde haya transacciones que impliquen información segura o monetaria, es evidente que está surgiendo como una alternativa para diseñar servicios web con menos dependencia en middleware propietario (por ejemplo, un servidor de aplicaciones), que su contraparte SOAP y los servicios basados en WSDL. De algún modo, REST es la vuelta a la Web antes de la aparición de los grandes servidores de aplicaciones, ya que hace énfasis en los primeros estándares de Internet, URI y HTTP. XML sobre HTTP es una interfaz muy poderosa que permite que aplicaciones internas, como interfaces basadas en JavaScript Asincrónico + XML (AJAX) puedan conectarse, ubicar y consumir recursos. De hecho, es justamente esta gran combinación con AJAX que generó esta gran atención que tiene REST hoy en día.

Para el framework que se quiere implementar, resulta muy flexible el poder exponer los recursos del sistema con un API REST, de manera de brindar datos a distintas aplicaciones, formateados en distintas maneras. REST ayuda a cumplir con los requerimientos de integración que son críticos para construir sistemas en donde los datos tienen que poder combinarse fácilmente y extenderse. Desde este punto de vista, los servicios REST se convierten en algo mucho más grande y serán utilizado para todos aquellos módulos o aplicaciones que puedan exponerse sin restricciones o problemas de seguridad.

3.4 ¿Qué es Memcached?[4]

Memcached es un sistema de cache de objetos/variables. Funciona en memoria RAM, puede ser distribuido y posee un alto desempeño en cuanto a performance, es indicado para agilizar páginas web dinámicas y para aliviar la carga en la base de datos. Es un proyecto open source y su desarrollo fue motivado justamente por la alta carga que estaban teniendo en el sitio para cual fue desarrollado originalmente (livejournal). Tuvo tan buena recepción que hoy en día esta implementado en varios sitios de alta carga (como por ejemplo, slashdot para los comentarios, ciertas partes de facebook aunque con una modificación, wikipedia, sourceforge, fotoblog, y google tiene un proyecto llamado memcachedb que es algo similar aunque persistente ya que permite guardar en disco su contenido).

Si bien es un cache, no es una “solución inteligente” del tipo plug&play sino que está muy orientado a la performance y a su capacidad para escalar. A continuación se muestran las diferencias:



4. Memcached se encuentra desarrollado por Danga Interactive (En línea) En: http://www.danga.com/memcached/

En un “cache transparente”, todas las consultas van hacia el cache. El cache devuelve los resultados cacheados si es que los tiene, y sino se encarga de hacer la consulta correspondiente a la base de datos, con lo cual termina cacheando ese resultado y devolviéndolo al webserver. La siguiente consulta será resuelta por el cache directamente sin consultar a la base de datos.

Características del *Memcached*

Características

- Sencillo: es una estructura en memoria que guarda una clave/key y bajo esa clave/key, el dato que necesitemos. Por lo tanto, para recuperar el dato necesitamos saber bajo que clave esta guardado. Esta clave sería el índice del cache.
- Rápido: todas las operaciones se realizan en memoria. No posee sistema de autenticación y su protocolo de comunicación es muy liviano.
- Sin Bloqueos: A diferencia de Oracle y sus “buffer busy waits” o sus “hot blocks”, memcached maneja de forma muy eficiente la concurrencia, permitiendo mantener grandes cantidades de conexiones sin bloquear ningún área de memoria ni encolar pedidos porque haya datos mas utilizados (hot blocks).
- Resultados en Array: actualmente la implementación de PHP serializa los datos antes de guardarlos, y trabaja con arrays sin inconvenientes. Esta es una ventaja ya que actualmente en la plataforma trabajamos con los resultados convertidos en array una vez obtenidos del Oracle.
- Timeout o LRU: Los objetos/resultados almacenados pueden invalidarse automáticamente mediante un parámetro de timeout; por otra parte cuando memcached se queda sin espacio, para agregar nuevos resultados utiliza un algoritmo de expulsión basado en LRU (last recently used) con lo cual podemos estar seguros que los resultados mas utilizados siempre quedarán en el cache. Para el timeout se usa un algoritmo de “lazy expiration”, con lo cual no existe un “garbage collector” ni un proceso que invalida las entradas todo el tiempo, sino que es el mismo proceso de lectura quien invalida el resultado al momento de obtenerlo.
- Distribuido/Failover: si bien no tiene un proceso central que sincronice failovers, memcached ofrece la posibilidad de levantar más de un server y distribuir la carga. En cuanto al failover, no hay una estrategia definida a priori, puede implementarse la que mejor convenga (desde no hacer nada hasta agregar dinámicamente nuevos servers).

Limitaciones

- Tamaño de objetos: está limitado a 1 MB, porque es el tamaño máximo de slab (por default no usa malloc/free sino que usa el slab allocator).
- Tamaño de claves: limitado a 250 caracteres.

Conclusión

Memcached es la opción seleccionada de cache para integrar con el framework a implementarse, el motivo de la elección es debido a su sencillez de uso, facilidad de implementación y por encontrarse importantes casos de éxito en su uso. Tampoco debemos olvidar que es un aplicación con licencia open source y por lo tanto gratuita, otros sistema de cache, como por ejemplo aquellos ofrecidos por Oracle, requieren la contratación de costosas licencias y hardware con mayores recursos.

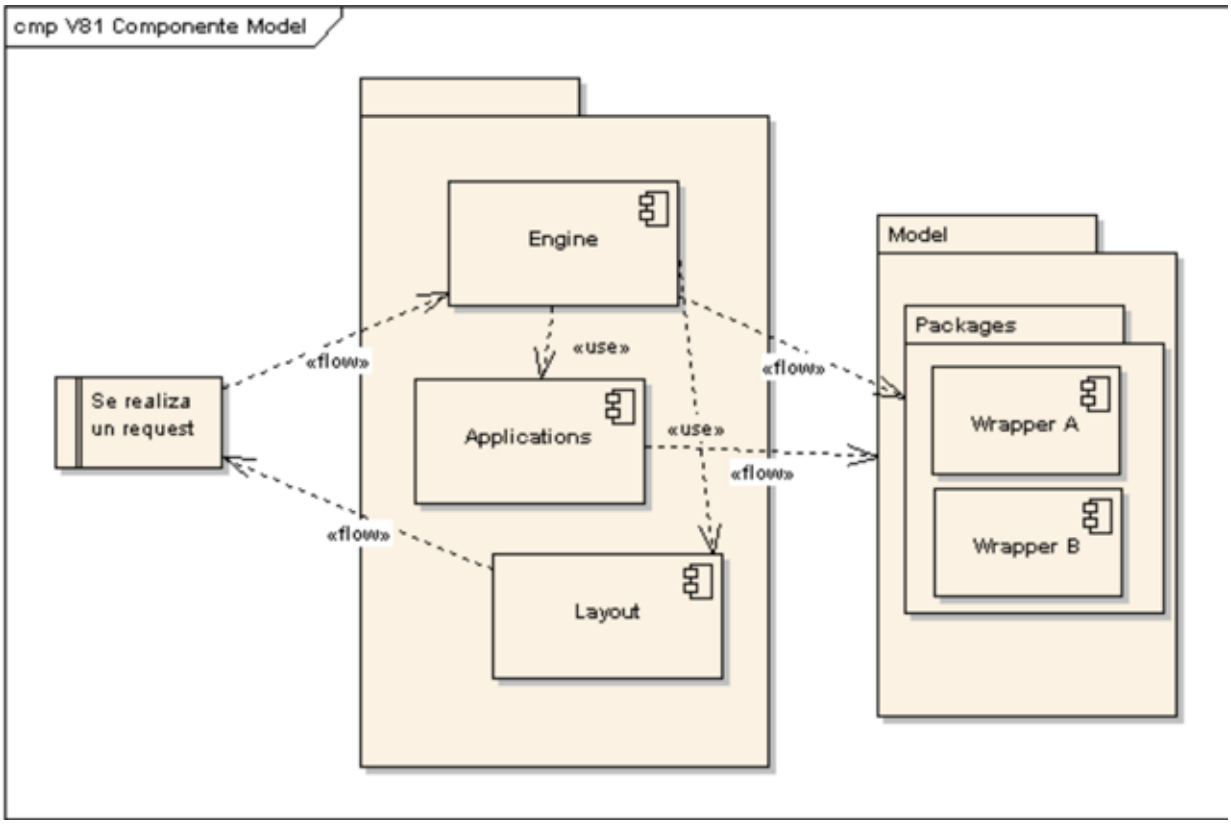
3.5 ¿Porqué ZEND Framework?

Tal como se plantea al principio del trabajo, nos encontramos ante tres opciones: modificar el framework actual, construir uno desde cero ó utilizar un framework open source ya existente en el mercado. De estás tres alternativas, se tomó a la última como mejor opción. A continuación se explican los motivos:

Modificar el framework actual, implicaba entre otras cosas tener que migrarlo de PHP4 a PHP5, con lo cual hubiese sido necesario prácticamente rescribir todo el código, además de tener que lidiar con todas las debilidades del framework actual y todas las modificaciones necesarias para que funcione con el modelo MVC propuesto, prácticamente el proceso tenía un costo similar a armarlo de cero.

A continuación se muestra un gráfico de componentes del framework actual donde se ve una separación entre el modelo y el resto del sistema, pero no entre los controladores y las vistas.

Diagrama de componentes del framework actual:



Otra debilidad o problema con el framework actual es que el mismo no fue pensado para poder manejar los llamadas tipo REST, una de las condiciones que se quieren en el nuevo framework. Por ejemplo una llamada al buscador se hace de la siguiente manera:

http://www.mixplay.tv/scripts/app/framework.php?FRAME=principal&APP=buscadorcontenidos&principal[pag]=1&&principal[tokens]=cadillacs&principal[id_nodo]=&principal[vista]=results&principal[origen]=simple

Como se puede ver cada valor en la url se pasa en forma de parámetro cuando lo que se quiere lograr es lo siguiente:

http://www.mixplay.tv/buscador/results/words/cadillacs/from/1/quantity/20

Utilizando el patrón REST, vemos como la llamada se transforma en algo sencillo e intuitivo.
Sería como llamar a la aplicación buscador con la acción results, diciéndole que busque la palabra “cadillacs” y que traiga desde el elemento número uno los primeros veinte resultados.

Se consideró la posibilidad de construir un framework desde cero, pero viendo que en el mercado ya existían frameworks reconocidos y estables, y teniendo en cuenta que el factor tiempo era una variable importante, se decidió que la mejor opción era obtener un framework de terceros open source que tuviese las características buscadas e invertir el tiempo en terminar de adaptarlo a las necesidades propias y no en reinventar la rueda armando uno desde el inicio.

Determinada como mejor elección el uso de un framework ya existente, se procedió a evaluar las ventajas y desventajas de los mismos:

De los frameworks más reconocidos dentro del mercado se optaron por los siguientes:

- CakePhp
- ZendFramework
- Symfony

Resultando como mejor opción el Zend Framework.

Si bien los frameworks citados anteriormente poseen características similares, por ejemplo todos están implementados sobre el patrón MVC, la decisión fue por el uso del Zend Framework, principalmente por los siguientes motivos: está totalmente implementado en PHP5, está desarrollado por gente de ZEND, (quienes desde el año 1997, vienen desarrollando y dando soporte a la comunidad PHP, entre otras cosas participaron en rescribir el CORE del lenguaje mismo, así como una activa participación en el desarrollo de PHP5 y próximamente en la nueva versión estable de PHP6), por lo tanto luego de analizarse el Zend Framework, y prácticamente revisarse línea a línea los objetos mas importantes (ej: Zend_Controller, Zend_View, etc.), se llegó a la conclusión de que implementar un framework MVC que este diseñado correctamente a nivel de responsabilidades de cada objeto, y poder extenderlo se estaría prácticamente rescribiendo cosas muy similares.

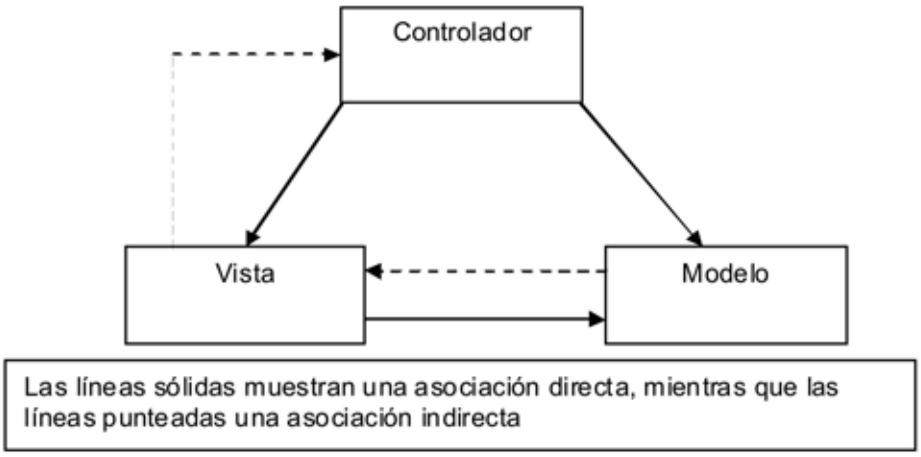
Esto sumado a que al utilizar un framework que va a ir creciendo en utilidades y que va a ser mantenido por toda un comunidad (tampoco podemos olvidar que son los primeros pasos de un framework oficial del lenguaje PHP) hacen que su elección se encuentre bien justificada.

3.6 Funcionamiento del ZF

En el siguiente apartado se explica como es el funcionamiento del Zend Framework:

A continuación se muestra como el ZF implementa al patrón MVC:

Diagrama que representa al patrón MVC



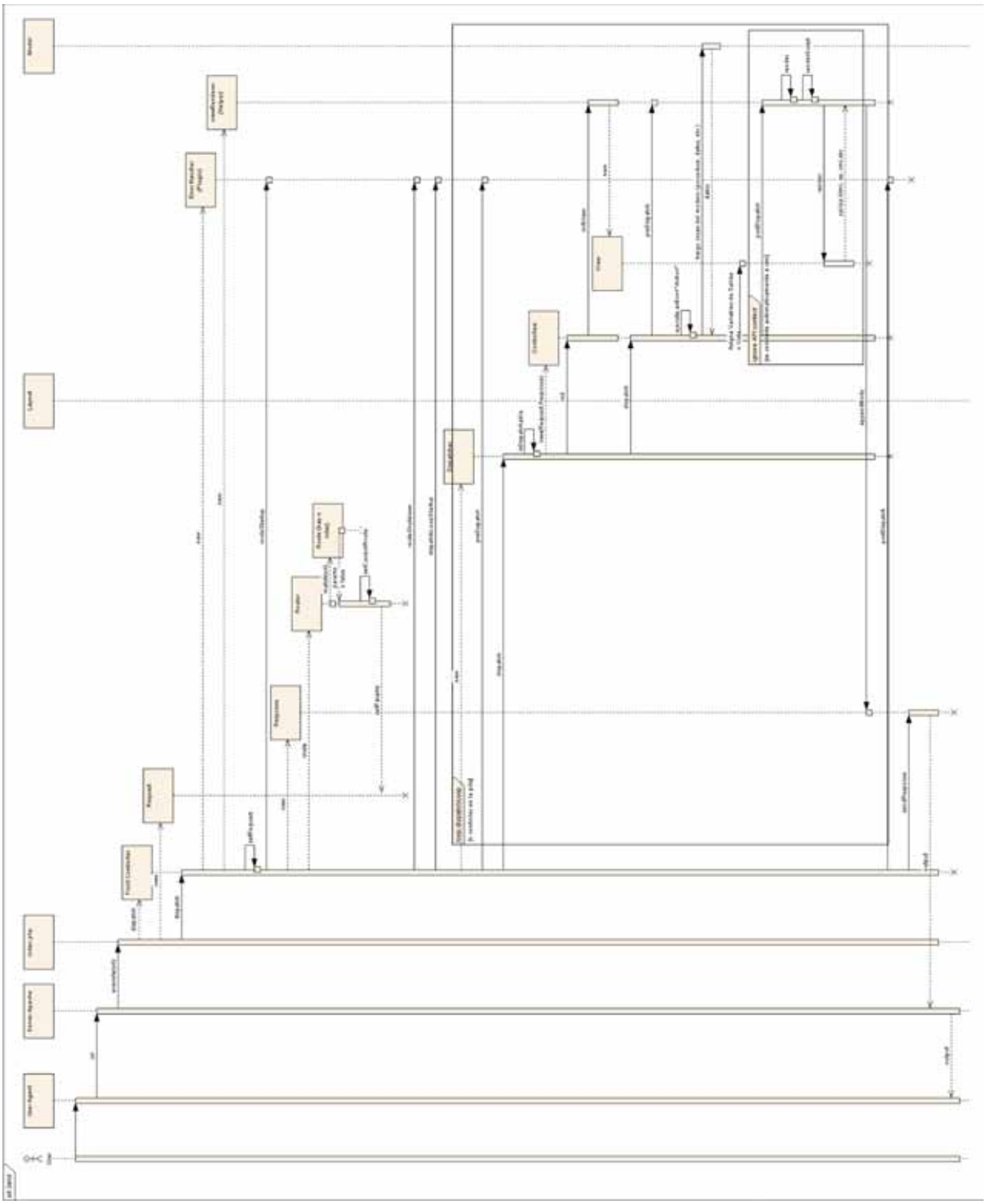
Modelo: Esta es la parte de la aplicación que define la funcionalidad básica detrás de un conjunto de abstracciones. La rutina para el acceso de datos y parte de la lógica del negocio pueden ser definidas en el modelo.

Vista: Definen exactamente que se le presenta al usuario. Generalmente los controladores pasan datos a cada vista para que se representen en algún formato. Las vistas en muchos casos obtendrán los datos del usuario también.

Controlador: Los controladores son los encargados de atar a todo el patrón. Manipulan los modelos, deciden que vista se debe mostrar basado en los pedidos del usuario y otros factores, pasan los datos que cada vista va a necesitar, ó pueden transferir el control a otro controlador. Siempre es recomendable que cada controlador sea los más sencillo posible.

Dentro de lo que es el ZF podemos destacar cuatro componentes para representar al patrón MVC: Zend_Controller para lo que sería el controlador, Zend_Layout y Zend_View para la parte de Vista y Zend_DB para la parte del modelo.

Flujo del ZF:[5]



5. Diagrama armado en la herramienta Enterprise Architect (EA Sparx Systems)

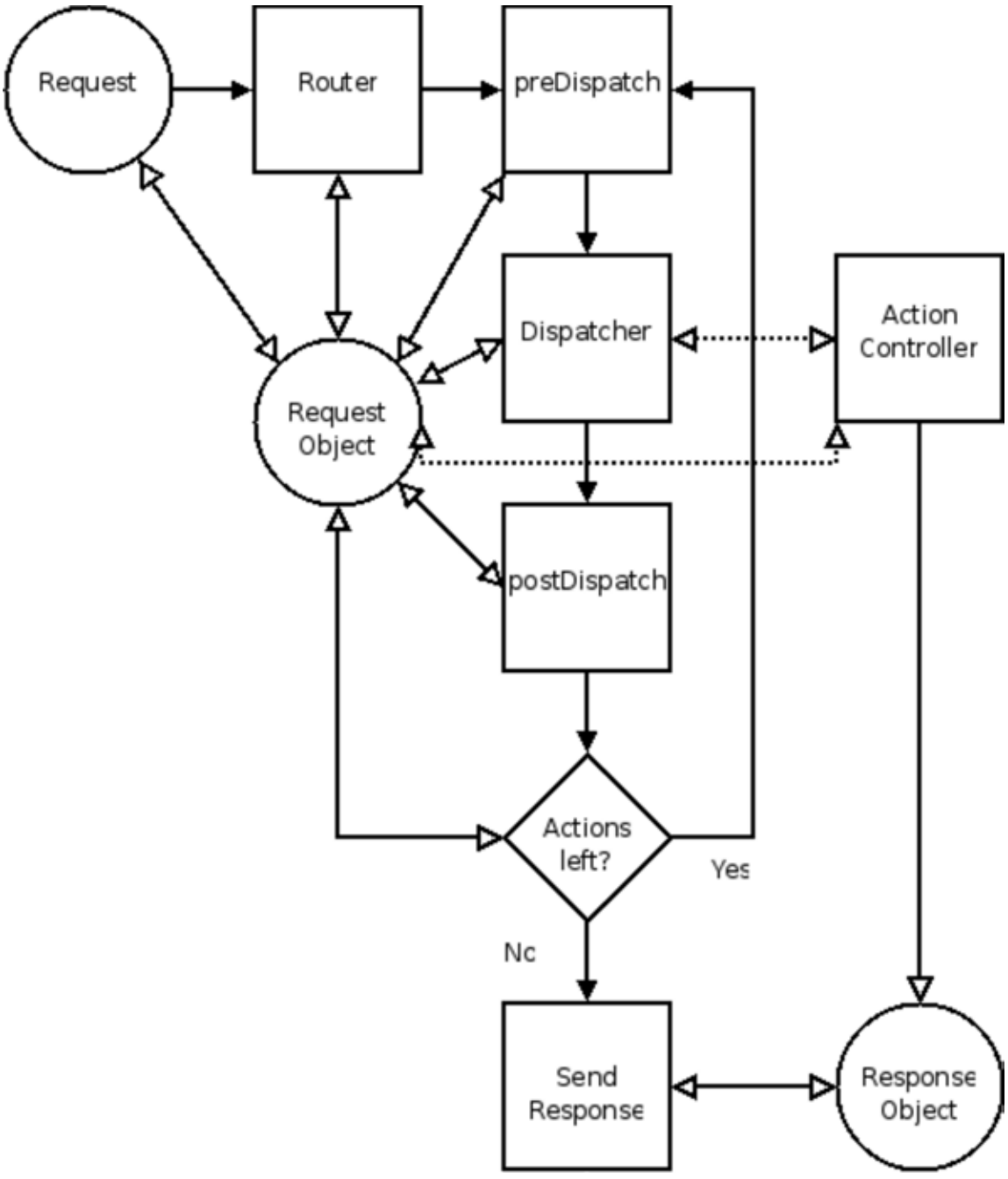
Patrones a Destacar dentro del Zend Framework

¿Qué es el Front Controller?

El front controller es un patrón que define a un solo componente como responsable del procesamiento de los pedidos de las aplicaciones. El front controller es el encargado de centralizar las funciones como por ejemplo la selección de vista, seguridad y las aplica en forma consistente a través de todas las páginas. Consecuentemente, cuando el comportamiento de estas funciones necesita cambiarse, solamente es necesario modificar una pequeña parte de la aplicación. Como por ejemplo el controlador y sus clases de ayuda (helper classes).

Implementación del front controller en el ZF

Diagrama de flujo del Zend Controller[6]



6. Gráfico tomado de http://framework.zend.com/manual/en/figures/zend.controller.basiucs.png

Explicación del diagrama de flujo:[7]

El Zend_Controller_Front recibe una petición (Request) que termina llamando al Zend_Controller_Router_Rewrite para determinar a que control (y acción en ese controlador) debe hacer el despacho. El Zend_Controller_Router_Rewrite descompone la URI en orden de establecer el nombre del controlador y las acciones de la petición. Luego el Zend_Controller_Front entra en un ciclo de despachos. Llama al Zend_Controller_Dispatcher_Standard, pasándole la petición o pedido para que sea despachado al controlador y acción especificada. Luego de que el controlador haya terminado, el control vuelve al Zend_Controller_Front. Si el controlador indicara que otro controlador necesitaría ser despachado, lo hace reiniciando el estado del despachador del pedido, luego el ciclo continúa y otro despacho es realizado. En caso contrario el proceso termina.

Para más detalles ver el Anexo I.

Two Step View (ó vistas compuestas):[8]

Otro patrón para destacar dentro del ZF es el de vistas compuestas o en dos etapas, este patrón fue ideado para lograr adaptabilidad y fácil modificaciones en los sitios web sin la necesidad de tener que tocar cada html, página ó layout en caso de un cambio de diseño que afecte a todo el sitio. Por ejemplo supongamos que cada página posee un encabezado y un pie que se repiten a lo largo del sitio, al cual es necesario introducirle una modificación o hacer que tan solo la mitad de las páginas ahora lo muestren. En ese caso y suponiendo que cada página incluye el encabezado y el pie por ejemplo del siguiente modo:

```
<?php
include (“encabezado.html);
?>
<div>contenido de la página</div>
<?
include (“pie.html”);
?>
```

Sería necesario entrar página por página para llamar al nuevo encabezado y pie ó eliminarlo de aquellas donde no vaya más. Para evitar este tedioso trabajo el patrón de vistas compuestas, divide en dos partes al rendereo de una página, primero la aplicación generará su propia salida en el caso del ejemplo solo generará la frase “cuerpo de la página” sin importar donde tenga que salir y luego dicha salida será rendereada por el layout que corresponda el cual será genérico para todas las aplicaciones de la página.

```
<html>
Encabezado
<div><?php echo $this->layout()->content
?></div>
Pie
</html>
```

En el ZF esto podemos verlo en la siguiente estructura:

Dentro del directorio views tenemos dos carpetas layouts y scripts. Dentro de scripts cada controller tendrá su render o vista simple. En la carpeta layout, tendremos el template donde luego será rendereada cada vista. Las dos clases principales de ZF que implementan estos métodos son la Zend_View y la Zend_Layout.

7. Traducción realizada del sitio http://framework.zend.com/manual/en/zend.controller.basics.html

8. Martin Fowler, Patterns of Enterprise Application Architecture (P of EAA): Two Step View (En línea): En: http://martinfowler.com/eaaCatalog/tweoSAtepView.html

4. Implementación del Sistema

Luego de introducidas, explicadas y justificadas las tecnologías que conformaran al sistema se procederá a la implementación del mismo:

4.1 Arquitectura del Sistema (Nivel Físico)

Ambiente de Desarrollo:

Se cuenta con un servidor donde se encuentra implementado el sistema, en el mismo se encuentra instalado el servidor de versionamiento (SVN) y corren sobre el sistema operativo Linux con distribución Debian con un webserver Apache 1.3.34 y PHP 5.

Un servidor donde se encuentra instalado el Oracle 10g de Desarrollo y una instancia del Memcached para el sistema de cacheo corriendo sobre el sistema operativo Linux con distribución Debian.

Ambiente de Producción:

Se cuenta con un pool de 5 servidores donde se encuentra implementado el sistema, corren sobre el sistema operativo Linux con distribución Debian con un webserver Apache 1.3.34 y PHP 5.

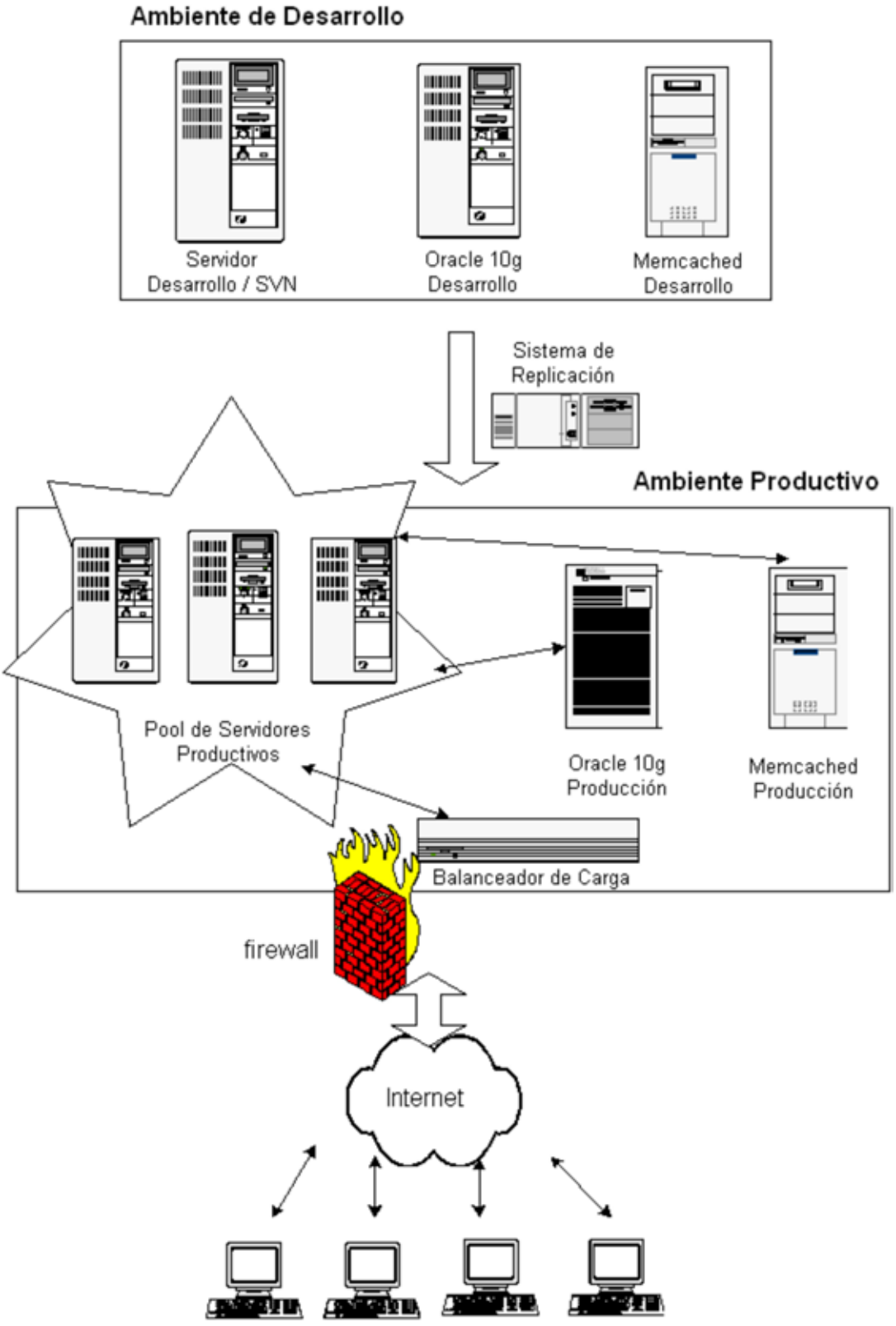
Un servidor dedicado al Oracle 10g, con sistema operativo Linux distribución Debian.

Un servidor dedicado a Memcached con dos instancias con Linux con distribución Debian.

Un balanceador de carga para el manejo de las conexiones.

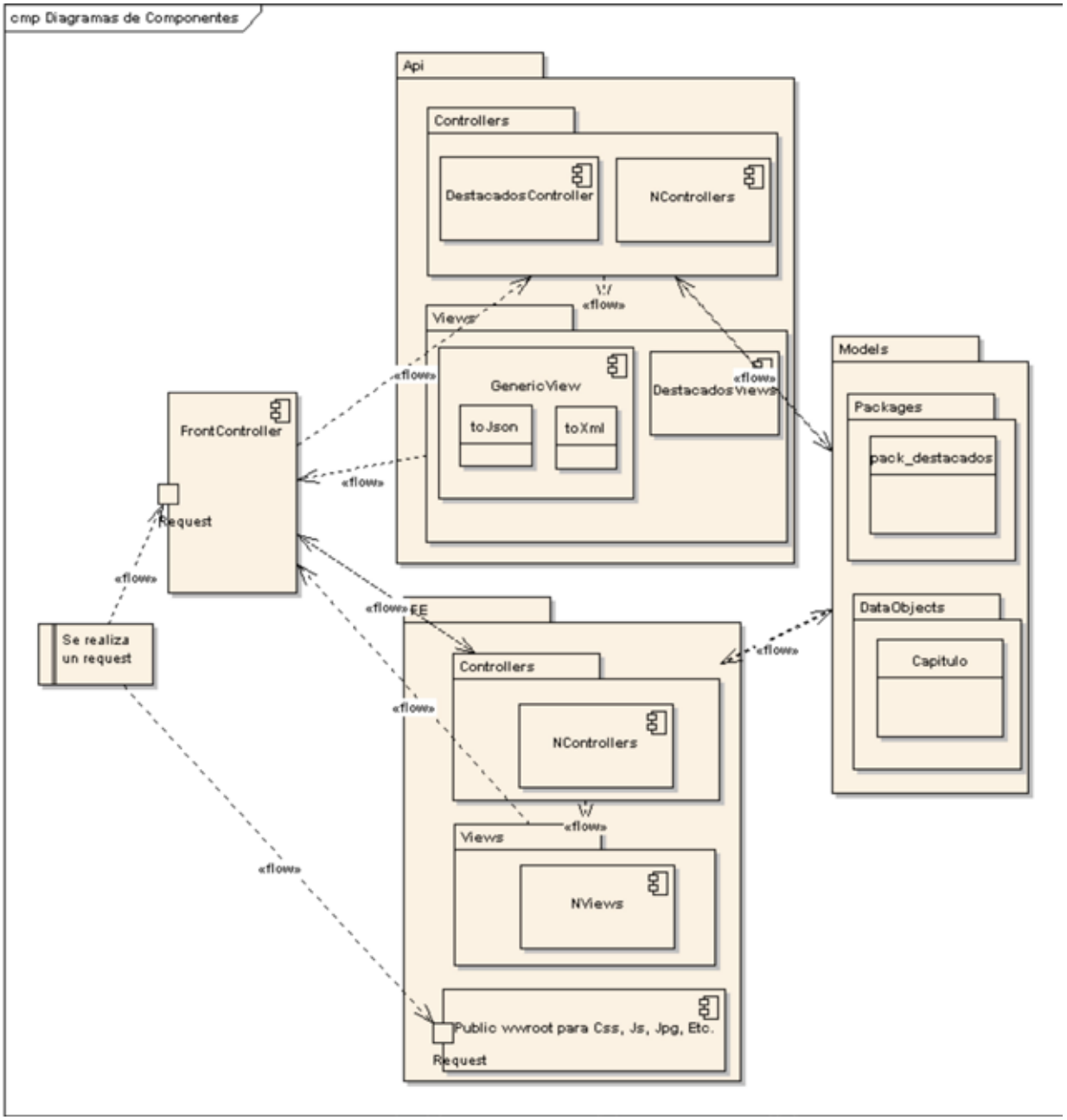
Los servidores donde corren los Oracle y el framework son HP ProLiant DL365 G5 2352 2.1GHz Quad Core. Con la diferencia que donde se encuentra instalado el Oracle productivo posee 8GB de Ram y 6 discos de 72GB, mientras que el resto cuentan con 4GB. y 4 discos de 72GB.

En el siguiente diagrama se muestra como se encuentran distribuidos los servidores que forman parte de la arquitectura del sistema.


Ambiente de Desarrollo
Servidor
Desarrollo / SVN
Oracle 10g
Desarrollo
Memcached
Desarrollo
Sistema de
Replicación
Ambiente Productivo
Pool de Servidores
Productivos
Oracle 10g
Producción
Memcached
Producción
Balanceador de Carga
firewall
Internet

4.2 Implementación del Framework

Diagrama de Componentes[9]



¿Cómo se utilizará el ZF?

Para lograr una correcta implementación del Zend Framework fue necesario resolver los siguientes interrogantes:

- ¿Cómo hacer que los cambios de versiones no impacten al producto?
- ¿Cómo hacer que el código y tiempo que se vayan a invertir para implementar el framework con las modificaciones que sean necesarias (aplicaciones, módulos, servicios, etc.) no tengan una dependencia del 100% con dicho framework?
- ¿Cómo hacer cosas particulares?

La idea entonces fue buscar la forma de generar un abstracción de dicho framework, realizando así una capa intermedia entre este y la utilización que hagamos del mismo.

9. Gráfico armado con la herramienta Enterprise Architect (EA Sparx Systems)

Para lograr un abstracción del ZF, la metodología utilizada será a través de la generación de clases con exactamente la misma estructura de directorios que en el propio Zend Framework, pero en donde las mismas serán cuerpos vacíos que extiendan a los originales.

A continuación se muestran los ejemplos para cada caso:

Clase:

Por ejemplo:
Zend/Controller/Action.php

```
class Zend_Controller_Action {}
```

La extensión propuesta será:
Controller/Action.php

```
class Controller_Action extends Zend_Controller_Action {}
```

Clase Abstracta:

Por ejemplo:
Zend/Controller/Router/Abstract.php

```
abstract class Zend_Controller_Router_Abstract {}
```

La extensión propuesta será:
Controller/Router/Abstract.php

```
abstract class Controller_Router_Abstract extends Zend_Controller_Router_Abstract {}
```

Clase Singleton:

Igual a una Clase normal, pero con la salvedad de que es necesario redefinir el método estático getInstances (sino la forma más normal de generar un Singleton en PHP que es como lo usa el ZF, devolviendo objetos de las clases Zend_ en vez de los del Port nuestro).

Interfaz:
Por ejemplo:
Zend/Controller/Router/Interface.php

```
interface class Zend_Controller_Router_Interface {}
```

En nuestro caso será:
Controller/Router/Interface.php

```
interface Controller_Router_Interface extends Zend_Controller_Router_Interface {}
```

Finalmente, también fue necesario agregar el path del ZF en la variable de php include_path, a través de la función set_Include_path[10] de modo que cada clase que sea extendida encuentre siempre la versión del ZF que se esté utilizando. Esto evita que si en el futuro se cambia a otra versión de ZF por ejemplo por ser una más nueva, sea solamente necesario modificar la configuración donde se define el include_path de la versión del ZF que se utilice.

Como se ve, la extensión de las clases e interfaces, genera una utilización del Zend Framework encapsulada en una jerarquía de clases e interfaces propias e independientes que permiten flexibilidad tanto al aplicar cambios sobre dicho framework como también independizarnos de las versiones del Zend Framework y de modo que las cosas particulares (módulos, controladores, plugins, etc.) usen la jerarquía de clases propia.

Consideraciones y utilidades adicionales agregadas al ZF

A continuación se listan aquellos temas que se agregaron al ZF, bien para no perder funcionalidad existente con el framewok anterior ó bien para la integración del mismo con las tecnologías a utilizarse (Oracle, Memcached).

10. Php.net, Description of core php.ini directives (En línea) En: http://ar.php.net/manual/en/ini.core.phpini.include-path

- Acceso a base de datos Oracle:

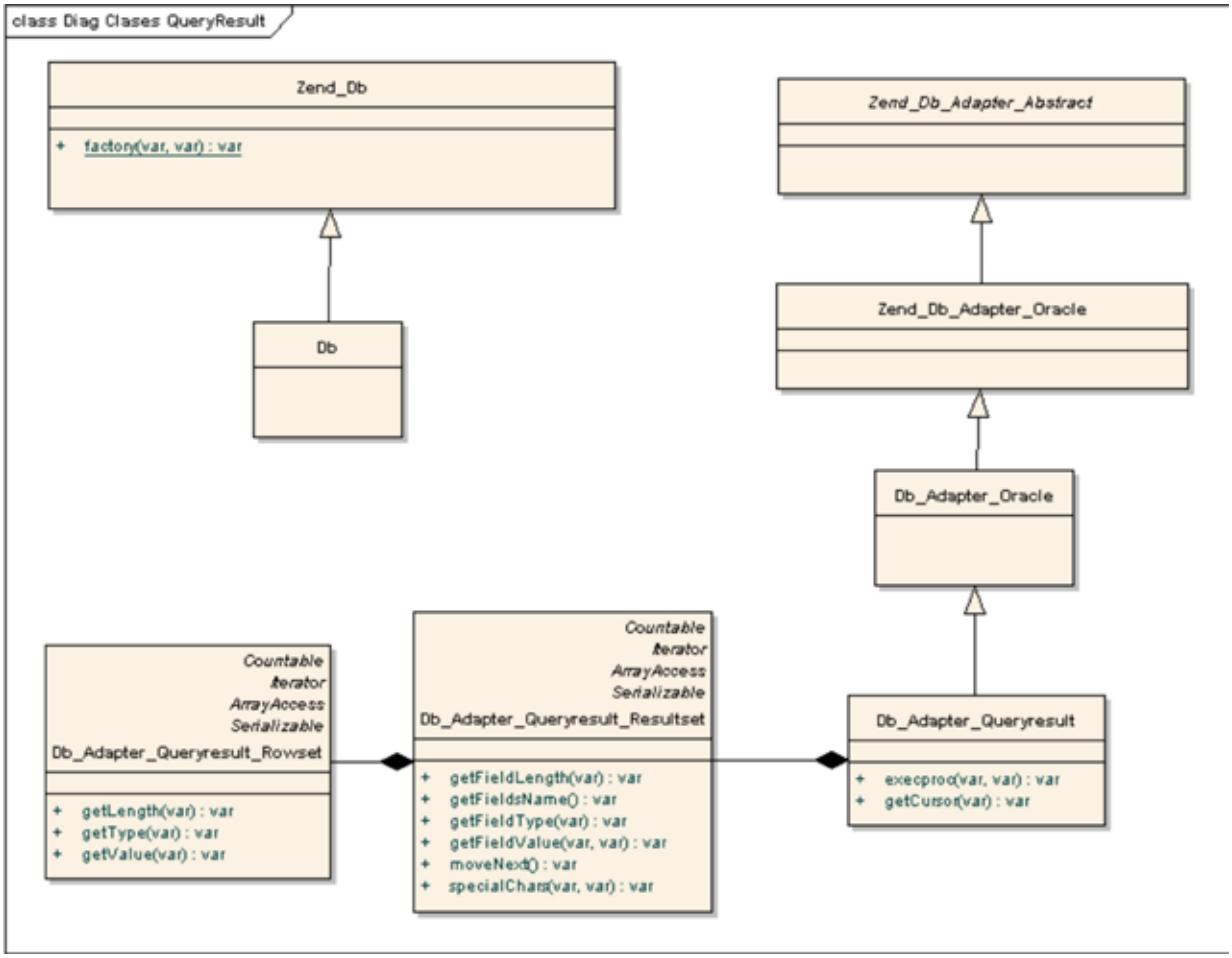
Se mejoró y reorganizó la Queryresult (clase para conexión utilizada en la versión anterior del framework), en los siguientes puntos:

- Separar la ejecución de los procedimientos, del recordset ó estructura de datos donde se almacenan los resultados de dicha ejecución.

- La posibilidad de generar múltiples cursores o resultados en un mismo procedimiento.

Las mejoras permiten llamar a un procedimiento como execproc (nombre del procedimiento, arreglo de parámetros para el procedimiento) tal como lo hacía antes, de modo de reducir la curva de aprendizaje de lo programadores que ya la utilizaban, pero sin obviar la necesidad de diseñar una separación mas correcta a nivel de objetos (la ejecución por un lado y el recordset por el otro), incluyendo las funcionalidades de múltiples cursores.

A continuación se muestra un diagrama de clases con la integración de la llamada a base de datos con el ZF:



En el gráfico se ve como a partir de la clase Zend_DB_Adapter_Oracle (nativa del framework) se agregaron las nuevas clases para integrar nuestro objeto de base de datos (queryresult) al ZF.
- Uso de Factory y Singleton de conexión a base de datos:
- Singleton para asegurarnos de reutilizar una sola conexión en todo momento durante la vida de un script.
- Posibilidad de generar otra conexión desde el Factory para casos especiales (scripts que deban conectarse a más de una BD).

- Implementar un Helper para los Modelos:

Dentro del ZF se denomina Helper a una clase que pueda ser accedida desde cualquier controlador y cuyo objetivo es facilitar el desarrollo de código.

Debido a que los modelos deben ser accedidos por cualquier controlador y la mayoría de estos implementan conexiones a base de datos, se optó por generar un helper que permita instanciar a cualquier modelo de modo transparente, o sea sin tener que preocuparse por controlar ó generar instancias a la base de datos ó cargar los procedimientos, etc.

El helper creado se llamó Models y de esta manera al helper se le piden los procedimientos en principio, pero no obviando la posibilidad de tener dataobjects en caso de que el aspecto del modelo de datos o negocio permita modelarlo de dicha forma.

Con este helper la llamada a un procedimiento dentro de un controlador se resume de la siguiente manera:

```
$this->Models->getPackage("buscador");
```

- Implementar en el Helper ContextSwitch particularidades de la salida:

El ZF no ofrece un helper denóminado ContextSwitch este helper es utilizado por el ZF para detectar en forma automática si debe devolver html por ejemplo cuando el método es un HttpRequest o Json, Xml si se utilizan utilidades para AJAX.

Por lo tanto por cada módulo nuestro, se generó un helper que herede del helper ContextSwitch de modo que según el tipo de módulo Api, Plus ó Gadget (los que existen por el momento) sepan que formatos devolver por defecto en el momento en que son instanciados.

- Generar vistas usando XSL:

El ZF genera las vistas de los controladores buscando por defecto en las carpetas views/scripts un archivo con exactamente el mismo nombre que el controlador que se está ejecutando pero con extensión phtml. Dentro de este archivo solo es posible mezclar código PHP con texto para generar cualquier tipo de salida.

Por ejemplo si el controlador a ejecutarse se denomina Buscador, cuando el ZF deba generar la vista buscará al archivo views/scripts/buscador.phtml

Sin embargo, uno de los puntos fuertes de la versión anterior del framework, era el uso del metalenguaje XSL para la generación de salidas en el código y no de un pseudo lenguaje armado con PHP

Para no perder esta importante característica, la solución fue generar en el adaptador de Zend_View (solo View en nuestro caso debido a la extensión de clases) una extensión para que en vez de buscar el phtml, busque primero la existencia de un archivo con el mismo nombre del controlador pero con extensión xsl y en caso de encontrarlo use la librería de PHP libxslt[11] para formatear la salida.

11. Php.net, XSL (En línea) En: http://ar.php.net/manual/en/book.xsl.php

Estructura de Directorios

Se planteó la siguiente estructura de directorios tendiendo en cuenta la migración de clientes a esta arquitectura y cambios de versión del ZF

```
/plataforma/
    cgi-bin/
        includes/
            framework/
                Carpetas del Port del ZF (donde están todas las extensiones de clases para lograr la abstracción)
                ZF/1.5/Zend/  (framework original)
            models/ (todo lo relacionado al acceso de datos)
                packages/
                    pack_destacados.php
                dataobject/
                    pdo_capitulo.php
        apis/
            controllers/
            views/
                layout/
                scripts/
            index.php (Atiende el código de error HTTP 404 del directorio apis y deriva a ../index.php)
        fe/
            gadget/
                controllers/
                views/
                    layout/
                    scripts/
            plus/
                controllers/
                views/
                    layout/
                    scripts/

    index.php
        lang/
            idioma/
                modulo/
                    NombreControl+Controller.xml
                    NombreControl+Controller.Accion.xml
                    Global.xml
                Global.xml
```

Consideraciones detrás de la estructura de directorio planteada:

Las siguientes son algunas consideraciones al respecto de la estructura anterior.

Uso del Index.php dentro de cada carpeta principal:

La intención de que cada carpeta principal (ej: apis, fe, gadget, plus) tenga un index.php, es por el caso en donde sea necesario realizar cosas particulares para esa instancia, pero siempre terminará llamando al index.php principal ubicado en cgi-bin/index.php para mantener la idea de un FrontController y no uno por cada directorio, brindando de esta manera una interfaz de configuración unificada a todos.

Por otro lado la idea también resultó conveniente para poder fácilmente determinar cual es el comportamiento dentro de un site en base a la url de acceso:

Por ejemplo, fácilmente se pueden definir reglas en el Apache para que las siguientes invocaciones accedan a la carpeta correspondiente:

http://www.mixplay.tv/ Usa FE
http://www.mixplay.tv/apis/ Usa APIS
http://apis.mixplay.tv/ Usa APIS
Etc.

Finalmente para que el index.php sea nuestro FrontController, necesitamos que todos los request o peticiones a nuestro sitio pasen por él. Esto se puede solucionar usando el módulo Rewrite del Apache, quien simplemente toma las peticiones que recibe a través de una URL y las rescribe de modo que apunten al index.php.

Las reglas para el módulo Rewrite[12] pueden definirse en el archivo de configuración del Apache llamado httpd.conf o agregando el archivo .htaccess con las siguientes reglas:

```
RewriteEngine on
RewriteCond %{REQUEST_FILENAME} !-f
RewriteRule .* index.php
```

En este caso el archivo .htaccess activa el modulo de Rewrite y luego hace que todos los requests o peticiones queden mapeadas al index.php con excepción de aquellas que soliciten archivos que sí existen en el servidor, por ejemplo las imágenes, los css, los códigos en javascript, los xmls, los flashes, etc. El parámetro –f es el encargado de hacer este chequeo.

Por ejemplo:

La URL: http://plus.mixplay.tv/buscador/results/words/hola sería mapeado al index.php recibiendo /buscador/results/words/hola para que el Front Controller pueda hacer el despacho al controlador buscador con la acción results y los parámetros words con valor hola.

La URL: http://plus.mixplay.tv/imagenes/mixplay.jpg no haría uso del modulo Rewrite porque el archivo mixplay.jpg si existe en el servidor. Por lo tanto al URL nos devolvería directamente la imagen.

Uso de models dentro de la estructura planteada

La capa de Modelo (carpeta models) de la arquitectura MVC como podemos ver se encuentra separada dentro del nivel includes (/includes/models/) de la estructura de directorios planteada, o sea no existe una carpeta models por cada carpeta controller y view que existan. El objetivo de esto es que la misma pueda ser reutilizada por las N abstracciones de Vista+Control que existan (por ejemplo dentro de los módulos apis, gadget, plus). De este modo nos aseguramos que los modelos que implementan reglas de negocio puedan ser reutilizados por cualquier controlador sin la necesidad de duplicar el código.

División por módulos dentro de la estructura planteada

Dentro de la siguiente estructura: /plataformas/cgi-bin/ encontramos los siguientes directorios:

/apis
/fe y dentro del mismo /gadget y /plus

Denominamos módulos a las partes expuestas del framework es decir el módulo /apis se encuentra compuesto por los controllers que serán consultados por terceros y por las propias páginas que deban armarse utilizando sus servicios. Por defecto los controllers ubicados en el módulo apis siempre devuelven código XML.

12. Apache.org, Module mod_rewrite URL Rewriting Engine (En línea) En: http://httpd.apache.org/docs/1.3/mod/mod_rewrite.html

Los módulos /gadget y /plus, se crearon para implementar los dos clientes Sliverlight, los controllers ubicados en estos módulos deberán devolver formato XAML (lenguaje propio de Microsoft Silverlight) y HTML.

4.3 Implementación del Memcached

-Generar acceso para el uso del Memcached

Para implementar el memcached dentro del sistema se tuvieron en cuenta los siguientes puntos:

Memcached en PHP: desde la versión 4.3.3 de Php, memcached es soportado simplemente instalando el paquete PECL, no es necesario realizar ningún tipo de recompilación en el PHP.

Hardware para memcached: La filosofía subyacente en memcached es no crecer verticalmente sino horizontalmente. Se pueden generar "farms" de servers con memcached distribuidos. Los servidores pueden ser dedicados para memcached o compartidos con otros servicios. El principal consumo es la RAM ya que ahí es donde guarda los datos; con respecto al uso de CPU, las pruebas preliminares dan que 40 usuarios concurrentes consumen aproximadamente un máximo de 12% de CPU.

En memcached, se debe resolver todo desde la programación. Memcached provee un API muy sencilla que permite consultar o agregar información, que es accedida mediante una clave o "key" (si bien tiene APIs para diferentes lenguajes, solo me referiré a PHP que es la opción a utilizarse). La operación es la siguiente:

1. El script PHP conecta con el memcached y pregunta si esta cacheado el resultado.
2. Si efectivamente esta cacheado, lo devuelve y el PHP continua procesando como si el resultado lo hubiese obtenido del Oracle.
3. Si no esta cacheado (el memcached devuelve "false") es el propio PHP que se debe encargar de hacer la consulta a la base de datos de forma transparente como si el memcached no existiera.
4. Al recibir los datos del Oracle, se los almacena en el cache para su posterior reutilización. La siguiente consulta que necesite esos datos, ya los encontrará cacheados como se mencionó en el punto 2, con lo cual ya no es necesaria la conexión a base de datos.

Definiendo que se puede cachear en el Memcached:

Debido a la lógica del sistema y de las aplicaciones, no todas las llamadas a la base de datos deben quedar cacheadas, por ejemplo una autenticación de usuario, debería siempre consultarse a la base, ó aquellas consultas que puedan devolver por ejemplo los créditos disponibles de un usuario también.

Ejemplo:
1- Script PHP llama al memcached con clave: get_creditos_usuario-id-10
2- Respuesta del memcached false
3- Script PHP llama a la base de datos para obtener créditos del usuario con id 10
4- Respuesta del Oracle: usuario id 10 -> 100 créditos.
5- Script PHP realiza una inserción al memcached con clave: getcreditosusuario-id-10 -> 100 créditos

Suponiendo que en esta instancia el usuario gasta 50 créditos, cuando se vuelvan a consultar los créditos del usuario, sucederá lo siguiente:

1- Script PHP llama al memcached con Key: getcreditosusuario-id-10
2- Respuesta del memcached true -> 100 creditos (esto es incorrecto, el usuario en realidad tiene 50 créditos).

Para evitar este escenario de error, se definió un archivo de configuración, donde es posible disponer aquellos procedimientos que pueden ser cacheados en el memcached y también por cuanto tiempo el resultado de esos procedimientos son válidos.

Generación de claves para Memcached:

Cada conexión al Oracle por parte del sistema, produce siempre una llamada a un procedimiento. Una llamada a un procedimiento está compuesta, por el nombre del package que contiene el procedimiento, el nombre del procedimiento y los parámetros necesarios para su llamada. Con lo cual cada vez que se consulta a la base de datos, deberá codificarse una clave para que luego el resultado pueda ser guardado en el memcached y recuperado en una siguiente llamada cuando el procedimiento y los parámetros sean iguales, sin pasar por el Oracle.

Teniendo en cuenta esta premisa, se definió cada clave de la siguiente manera:

NOMBRE_PACKAGE-NOMBRE-PROCEDURE||PARAMETRO1||PARAMETRO2||..||PARAMETRO N

De esta manera nos aseguramos que cada llamada a los distintos procedimientos van a generar siempre la misma clave a la hora de guardarse en el memcahe.

Ejemplo:
Una llamada al procedimiento de una aplicación de Rankings sería:

```
  private $_packName=”PACK_RANKING_MVISTOS”;
   public function pr_consulta_mvistos (
                                $p_idioma,
                                $p_countrycode,
                                $p_nivel,
                                $p_presicion,
                                $p_minutos,
                                $p_horas,
                                $p_fecha_desde,
                                $p_fecha_hasta,
                                $p_cant_periodo,
                                $p_desde_fila,
                                $p_cant_filas
                                )
  {
     $par[]=Array(“NAME” => “P_ID_IDIOMA”           ,”VALUE” => $p_idioma );
     $par[]=Array(“NAME” => “P_COUNTRYCODE”         ,”VALUE” => $p_countrycode );
     $par[]=Array(“NAME” => “P_NIVEL”               ,”VALUE” => $p_nivel );
     $par[]=Array(“NAME” => “P_PRESICION”           ,”VALUE” => $p_presicion );
     $par[]=Array(“NAME” => “P_MINUTOS”             ,”VALUE” => $p_minutos );
     $par[]=Array(“NAME” => “P_HORAS”               ,”VALUE” => $p_horas );
     $par[]=Array(“NAME” => “P_FECHA_DESDE”         ,”VALUE” => $p_fecha_desde );
     $par[]=Array(“NAME” => “P_FECHA_HASTA”         ,”VALUE” => $p_fecha_hasta );
     $par[]=Array(“NAME” => “P_CANT_PERIODO”        ,”VALUE” => $p_cant_periodo );
     $par[]=Array(“NAME” => “P_DESDE_FILA”          ,”VALUE” => $p_desde_fila );
     $par[]=Array(“NAME” => “P_CANT_FILAS”          ,”VALUE” => $p_cant_filas );
     $par[]=Array(“NAME” => “P_CURSOR_OUT”          ,”TYPE” => ‘CURSOR’);
     $this->_db->execproc($this->_packName.’.PR_CONSULTA_MVISTOS’,$par);
     return($this->_db->getCursor(“P_CURSOR_OUT”));
```

En este caso la clave quedaría conformada de la siguiente forma:

PACK_RANKING_MVISTOS-
PR_CONSULTA_MVISTOS||ESP||AR||1||0||30||0||08/12/2008||08/12/2008||10||0||10

Donde ESP es el valor del parámetro P_ID_IDIOMA, AR es el valor de P_COUNTRYCODE, 1 es el valor de P_PRECISION y así sucesivamente con todos los parámetros del procedimiento.

5. Implementación de una API dentro del framework

Ejemplo de un controller

Las siguientes líneas son ejemplos de cómo implementar un controller.

```
<?   // cgi-bin/api/controllers/FrontpageController.php
     require_once("Controller/Action.php");
     class DestacadosController extends Controller_Action  {
     function indexAction($parametro1)  {
              $pack_destacados=$this->models->getPackage("pack_destacados");
              $rsCursor1=$pack_destacados->pr_destacados_obtener($parametro1);
              $rsCursor2=$pack_destacados->getRecordset("cursor_adicionalX");
              $this->view->resultados= $rsCursor1->toArray();
              $this->view->resultadosAdicionales= $rsCursor2->toArray();
     }
}
```

```
<? // cgi-bin/api/views/scripts/frontpage/index.phtml
     foreach($view->resultados as $row) print $row->nombre;
```

```
<? //ó cgi-bin/api/views/xsl/frontpage/index.xsl
     <xsl:template match="$x/resultados"><xsl:value-of select="./nombre"/></xsl:template>
```

Ejemplo de Llamada:

http://plus.mixplay.tv/apis/home/frontpage/manual
http://plus.mixplay.tv/apis/home/frontpage/manual/includeCommon/true/extended/true

6. Ambiente de Demostración

Una vez implementado el framework, se procederá a mostrar como se utiliza el mismo a través de dos casos: el primero es un ejemplo que muestra como se construyó un sitio que funciona haciendo uso del mismo y el segundo ejemplo trata de la exposición de APIS a un tercero para que pueda armar secciones dentro su propio sitio.

Caso 1:
Datos Generales:

Nombre del sitio: Mixplay Plus.
URL de acceso: http://plus.mixplay.tv

Características:

El cliente de este sitio se encuentra desarrollo en Microsoft Silverlight (nueva tecnología de Microsoft), de esta manera queda expuesto como el framework puede generar salidas en diferentes formatos. Para este sitio los formatos de salida deberán ser XAMLS (lenguaje de Silverlight), HTML y XML para los respuestas de los servicios.

Es un sitio cross-browser, es decir, debe funcionar en cualquier sistema operativo y navegador que soporte el objeto Silverlight[13].

13. Sistemas Operativos Soportados: Windows Vista and Windows XP Service Pack 2
Navegadores Soportados: Microsoft Internet Explorer 6, Windows Internet Explorer 7, Windows Internet Explorer 8, Mozilla Firefox 1.5.0.8, and Firefox 2.0.x, Firefox 3.0.x

El objetivo de este sitio es ofrecerles a los usuarios contenidos audios visuales a través de una interfaz amigable. Dentro de las funcionalidades del mismo, los usuarios podrán registrarse, votar a los videos y seleccionar los mismos a través de diferentes herramientas disponibles en el sitio, por ejemplo a través de un buscador, de contenidos relacionados al que se esté viendo ó a través navegador de contenidos que funciona como un catálogo, así mismo los contenidos se encuentran agrupados por categorías y canales.

Estructura de Directorios:

Para implementar este sitio dentro del framework se generó la siguiente estructura:

```
/plataforma/cgi-bin/fe/
                    plus/ index.php (redirecciona al index.php ubicado a la altura de cgi-bin quien
inicializa al Front Controller)
                        controllers/
                          views/
                              layout/
                                  scripts/
                    apis/
                          controllers/
                          views/
                                  layout/
                                   scripts/
```

Plus: es el nombre del módulo ó sitio. A partir de él se encuentran todos los archivos (controllers, views) que son propios de la página.

Apis: es el nombre del módulo utilizado para los servicios del framework. Los controllers ubicados en Apis son utilizados para alimentar a los controllers principales de una página, el formato por defecto de sus salidas es en XML.

Dentro de los helpers del framework se creó:

/cgi-bin/includes/framework/Controller/Action/Helper/PlusContext.php, este PHP, define los formatos por defecto que va aceptar el módulo plus. Los mismos serán xaml y html.

Archivos de Configuración:

Se crearon los archivos de configuración del sistema:

config.xml: Archivo general para la configuración del framework. En el se especifican las variables globales: idioma, identificador de site, nombre del site, etc., servidor de videos, ruta de los controllers según el módulo – apis, plus -, que adapter se usa para base de datos, datos de conexión –usuario, clave, servidor - , servidor del memcached, el layout por default, etc.)

cache.xml: Archivo donde se configuran que procedimientos deberán almacenarse en el memcached, por cuanto tiempo y si necesitan alguna clave especial.

A continuación se muestra un fragmento de este xml donde se ven algunos de los procedimientos que van a ser cacheados:

Macintosh Computer con Procesador Intel
Sistemas Operativos Soportados: Apple Mac OS X 10.4.8+
Navegadores Soportados: Firefox 1.5.0.8, Firefox 2.0.x, Firefox 3.0.x and Apple Safari 2.0.4+
Fuente: http://www.microsoft.com/silverlight/overview/faq.aspx

```
<!-- Memcached para Fichas -->
<pack_fichas.pr_obtener_grupov_ficha>
    <id></id>
    <lifetime>300</lifetime>
</pack_fichas.pr_obtener_grupov_ficha>

<pack_fichas.pr_obtener_contv_ficha>
    <id></id>
    <lifetime>300</lifetime>
</pack_fichas.pr_obtener_contv_ficha>
```

En este caso vemos dos procedimientos dentro del pack_fichas donde sus llamadas son cacheadas por 300 segundos ó 5 horas.

```
<!-- Memcached para Relacionados -->
<pack_player.pr_busqueda_relacionados>
    <id></id>
    <lifetime>3600</lifetime>
</pack_player.pr_busqueda_relacionados>
```

En este otro ejemplo vemos otro procedimiento cacheado encargado de traer contenidos relacionados donde su expiración es de 24 horas.

principal.xml: Archivo cuyo nombre se define en el config.xml y tiene la configuración del layout con todos los controllers y parámetros que deben llamarse. Tiene la estructura de la página.

```
<sections>
  <section name="seccionfondo" method="Invoke">
    <module></module>
    <controller>Estatico</controller>
    <action>fondo</action>
  </section>
  <section name="seccionfondoplayer" method="Invoke">
    <module></module>
    <controller>Estatico</controller>
    <action>fondoplayer</action>
  </section>
  <section name="seccionfrenteskin" method="Invoke">
    <module></module>
    <controller>Estatico</controller>
    <action>frenteskin</action>
  </section>
  <section name="seccionheader" method="Invoke">
    <module></module>
    <controller>Estatico</controller>
    <action>header</action>
  </section>
  <section name="seccionlogo" method="Invoke">
    <module></module>
    <controller>Estatico</controller>
```

```
      <action>logo</action>
    </section>
    <section name="seccionuserinfo" method="Invoke">
      <module></module>
      <controller>Usuario</controller>
      <action>userinfo</action>
    </section>
    <section name="seccionfooter" method="Invoke">
      <module></module>
      <controller>Estatico</controller>
      <action>footer</action>
    </section>
    <section name="seccionnav" method="Invoke">
      <module></module>
      <controller>Nav</controller>
     <action>index</action>
    </section>
    <section name="seccionbuscador" method="Invoke">
      <module></module>
      <controller>buscador</controller>
      <action>form</action>
    </section>
     <section name="secciontapas" method="Invoke" content="yes">
        <module></module>
        <controller>tapas</controller>
         <action>index</action>
    </section>
    <section name="seccionplayer" method="Invoke">
      <module></module>
      <controller>player</controller>
      <action>index</action>
    </section>
    <section name="seccionsilverlightobject" method="Invoke">
      <module></module>
      <controller>Silverlightobject</controller>
      <action>index</action>
    </section>
  </sections>
```

En este archivo xml vemos como se conforma el sitio. En el caso que se requiere agregar una nueva sección, es suficiente con agregarla en este archivo de configuración y en el archivo principal.xaml.phtml (este último se explica más adelante).

Como se puede apreciar, muchas secciones del sitio llaman al controlador estático tan solo modificando la acción, este controlador se utiliza para agregar imágenes y animaciones que no tengan necesidad de obtener datos de la base. (Por ejemplo: el logo, el fondo, etc.).

Controllers del Sitio:

Dentro de la carpeta plus/controllers se crearon las aplicaciones que conforman al sitio:

BannerController.php (para manejar la publicidad dentro del sitio)
BrowserController.php (para generar el navegador de contenidos)
BuscadorController.php (para generar el formulario de búsqueda o los resultados)
EstaticoController.php (para generar elementos fijos, como fondos ó imágenes)
FichaController.php (para generar las fichas de contenidos)

IndexController.php (este controller es llamado cuando no se especifica ningún otro controller)
LayoutController.php
NavController.php (para generar los menús del sitio)
PlayerController.php (para generar el reproductor de contenidos)
SilverlightobjectController.php (para instanciar al objeto silverlight)
SocialController.php (para generar el compartir videos)
TapasController.php (para generar las tapas del sitio)
UsuarioController.php (para generar el menú de ingreso, registración, modificación de datos de usuarios)

Dentro de la carpeta plus/views/scripts/ se crearon los PHPs que darán el formato de salida según cuáles sea necesaria (por ejemplo xamls, htmls, etc):

Por cada controller:

browser/ index.phtml (salida del browser en formato html)
index.xaml.phtml (salida del browser en formato xaml)

buscador/ form.xaml.phtml (acción que muestra al formulario de búsqueda en formato xaml)
results.xaml.phtml (acción que muestra el resultado de búsqueda en formato xaml)

ficha/ index.phtml (salida de una ficha en formato html)
index.xaml.phtml (salida de una ficha en formato xaml)

y así sucesivamente con cada controlador.

Dentro de la carpeta plus/views/layouts/ se crearon los siguientes PHPs
El siguiente archivo es el encargado de instanciar a todos los controladores que conforman al sitio.
principal.phtml (salida en formato html)
principal.xaml.phtml (salida en formato xaml)

El siguiente archivo se generó para poder llamar a un solo controlador, usado para pruebas.
modal.phtml (salida en formato html)
modal.xaml.phtml (salida en formato xaml)
Se podría utilizar el principal.phtml pasándole como parámetro extra el valor layoutoff, cuando el framework recibe este parámetro, solo renderea la aplicación principal y no continua con el resto de los controllers que son llamados en la vista, por lo tanto solo genera la salida de un solo controlador.

Dentro de la carpeta apis/controllers se crearon las APIS que servirán como servicios al sitio:

CardController.php (devuelve los datos de un contenido para una ficha)
FrontPageController.php (devuelve los datos de contenidos para una tapa)
NavController.php (devuelve los datos que conforman al menú de navegación)
PlayerController.php (devuelve los datos para reproducir un contenido)
RankingController.php (devuelve y recibe datos para votar contenidos y traer los rankings)
RelatedController.php (devuelve los datos de los contenidos relacionados a un contenido)
SearchController.php (devuelve los datos de búsqueda)
SocialnetworksController.php (devuelve los datos para recomendar un contenido)
UserController.php (devuele los datos de los usuarios)

En las siguientes imágenes se muestran que controladores son invocados en cada sección del site:


BuscadorController
EstaticoController
NavController
TapasController
EstaticoController
Lo Nuevo
Más Vistos


PlayerController action: history
PlayerController action: browser
PlayerController action: related
SocialController
Relacionados Dreamworks SKG


mixplay
BUSCAR
Todos
OK
Ingresar
Regístrate
Categorías
Canales
Home / Destacados
PlayerController
CALIDAD DEL VIDEO
BAJA
MEDIA
ALTA
VOLVER A VER
VER MÁS VIDEOS
RELACIONADOS
COMPARTIR
« Anterior
Madagascar: Escape 2 Africa - Trailer
DREAMWORKS
03/11/08
00:01:44
Votos: 5
110174 plays
VOLUMEN
00:00:26
Siguiente »
IAB


Categorías
Canales
Home / Destacados
BuscadorController
Resultado de búsqueda: 7121 resultados
1 2 3 4 5 Siguiente »
Virgen de Guadalupe
12/12/2008
129 plays
CNN en español
1 Capítulos
Veredicto en Gran Bretaña
13/12/2008
80 plays
CNN en español
1 Capítulos
EE.UU. congela fondos
12/12/2008
21 plays
CNN en español
1 Capítulos
Senador Christopher Dodd
13/12/2008
15 plays
CNN en español
1 Capítulos
CQC | 08-12-2008 | El balance de los famosos
13/12/2008
189 plays
CQC
1 Capítulos
CQC | 08-12-2008 | Madonna en la Argentina
13/12/2008
126 plays
CQC
1 Capítulos
CQC | 08-12-2008 | Cristina en un evento científico
13/12/2008
49 plays
CQC
1 Capítulos
CQC | 08-12-2008 | Top Five
13/12/2008
78 plays
CQC
1 Capítulos
« Anterior
1 2 3 4 5 Siguiente »
Madagascar: Escape 2 Africa - Trailer
DREAMWORKS
00:01:22
00:01:44
VOLUMEN
IAB

Ingreso al sitio y su interacción con el framework:

Introducidos los archivos de configuración, controllers y layouts que conforman o son utilizados por el sitio, se procede a explicar el funcionamiento del mismo:

El proceso de carga del sitio puede dividirse en tres etapas:

Etapa 1:

Cuando se inicializa el sitio desde un navegador, (introduciendo la URL http://plus.mixplay.tv), se ejecuta el index.php (nuestro front controller), debido a que en la URL no se especifica ningún controller o acción, se produce una llamada al layout principal.phtml, el cual posee configurado todos los controllers que se deben invocar para armar la página, la misma por defecto se generará en formato html. Dentro de esta salida del layout uno de los controladores que se ejecutan es el SilverlightobjectController el cual genera el código del object encargado de instanciar al objeto Silverlight dentro del navegador y pasándole como parámetro la URI ingresada (en este caso solo /, porque la URL era http://plus.mixplay.tv/) pero pidiendo el layout en formato xaml (/?format=xaml).

A continuación se muestra la parte del código que instancia al objeto Silverlight (generada por el SilverlightobjectController):

```
Silverlight.createObject(
    “/fe/plus/Mixplay.Plataforma.FePlus.v1_1.xap”, // Source property value.
    parentElement, // DOM reference to hosting DIV tag.
    “BaseApp”, // Unique plug-in ID value.
    { // Plug-in properties.
    width:’824’, // Width of rectangular region of plug-in in pixels.
    height:’618’, // Height of rectangular region of plug-in in pixels.
    background:’Transparent’, // Background color of plug-in.
    isWindowless:’true’, // Determines whether to display plug-in in windowless mode.
    version:’2.0.31005.0’, // Silverlight version.
    autoUpgrade:’true’,
    //alt: altHTML // Alternate HTML to display if Silveright is not installed.
    altInvoke: mostrarNoInstalado //Si el Silverlight no está instalado muestro el div con el proceso de instalacion.
     },
    {
    onError:’onSilverlightError’, // OnError property value -- event-handler function name.
    onLoad:null // OnLoad property value -- event-handler function name.
    },
    “domainFrontend=http://www.mixplay.tv/,nombreNodo=,skin=sk_mixplay,red=,id_contenido=,lang=esp,region=argentina,template=,starturl=/?format=xaml,regionalstarturl=,regionalbaseurl=http://plus.mixplay.tv/fe/plus,baseurl=http://plus.mixplay.tv/fe/plus,apibaseurl=http://plus.mixplay.tv/apis,sitename=mixplay,adsite=cl_mixplay”, // initParams -- user-settable string for information passing.
    null);
```

La parte más importante respecto al funcionamiento del framework se encuentra resaltada en gris y es la que dará origen a la etapa 2.

Etapa 2:

El objeto Silverlight ya instanciado, generará una nueva llamada al framework, solicitando http://plus.mixplay.tv/?format=xaml donde se producirá un ciclo similar al ya explicado en la etapa 1, pero ahora el layout será armado por el principal.xaml.phtml quien devuelve el código en el formato XAML para que pueda ser interpretado por el objeto Silverlight.

A continuación se muestra parte del XAML generado en esta etapa:

Sin entrar demasiado en detalle en el lenguaje XAML, en nuestro caso el mismo se encuentra compuesto por un Grid o Grilla que define el área o marco del objeto y dentro de esta Grilla figurará un Canvas por cada sección que conforme a la página. Por lo tanto nuestro XAML está compuesto por un Grid y n Canvas, cada uno representando una sección.

En este ejemplo vemos el Canvas correspondiente al Nav encargado de generar los menús de la página:

```
<Canvas xmlns:MarkupExtensions=”clr-namespace:Mixplay.Common.MarkupExtensions;assembly=Mixplay.Common” xmlns:Plataforma=”clr-namespace:Mixplay.Plataforma.FePlus;assembly=Mixplay.Plataforma.FePlus” xmlns:app=”clr-namespace:Mixplay.Aplicaciones.FePlus;assembly=Mixplay.Aplicaciones.FePlus” xmlns:controls=”clr-namespace:Mixplay.Controls;=assembly=Mixplay.Controls” xmlns=”http://schemas.microsoft.com/client/2007” xmlns:x=”http://schemas.microsoft.com/winfx/2006/xaml”>
<MarkupExtensions:DynamicLoaderControl XapName=”/Mixplay.DesignMixplay.FePlus.Nav.xap” AssemblyName=”Mixplay.DesignMixplay.FePlus.Nav.dll” ClassName=”Mixplay.DesignMixplay.FePlus.Nav.Index_default”>
<MarkupExtensions:DynamicLoaderControl.Actions>
<MarkupExtensions:ControllerAction Name=”ApiNavWeb” Action=”/apis/argentina/home/nav/index/extended/1/format/xml/flagtype/web/fromController/Nav/fromAction/index” />
```

<MarkupExtensions:ControllerAction Name="ApiNavChannel" Action="/apis/argentina/home/nav/index/extended/1/format/xml/flagtype/channel/fromController/Nav/fromAction/index" />
<MarkupExtensions:ControllerAction Name="ApiMostNews" Action="/apis/argentina/%7B%40node%7D/ranking/getfeatured/from/0/quantity/30/fromController/Nav/fromAction/index" />
<MarkupExtensions:ControllerAction Name="ApiMostSeen" Action="/apis/argentina/%7B%40node%7D/ranking/getmostviewed/from/0/quantity/30/fromController/Nav/fromAction/index" />
</MarkupExtensions:DynamicLoaderControl.Actions>
<MarkupExtensions:DynamicLoaderControl.Params>
<MarkupExtensions:ControllerParam Name="HomeNode" Value="home" />
</MarkupExtensions:DynamicLoaderControl.Params>
</MarkupExtensions:DynamicLoaderControl>
</Canvas>

En gris se encuentra destacada la parte más importante respecto a nuestro framework y que dará origen a la etapa 3.

Etapa 3:

El sitio cuenta con dos menús uno llamado Categorías y otro llamado Canales, si vemos los Actions de los tags marcados en gris de la etapa 2, vemos que el primero posee la siguiente acción:

"/apis/argentina/home/nav/index/extended/1/format/xml/flagtype/web/fromController/Nav/fromAction/index", si a este valor le anteponemos http://plus.mixplay.tv/ obtenemos una URL que llamará al módulo de APIS invocando el controller nav, con al acción index y el resto como pares de parámetro-valor.

Por lo tanto, en esta tercera etapa, cada sección que compone al Silverlight obtendrá los datos que debe representar invocando a las distintas APIS del framework, en el ejemplo del Canvas Nav, invocará a la API nav con una serie de parámetros para obtener el XML que represente al menú Categorías y nuevamente llamará a la API nav con otros parámetros para obtener el XML que represente al menú de Canales. El parámetro que maneja el nav controller para decidir el tipo de menú es el /flagtype/, pudiendo ser web o channel según el tipo de menú.

Al finalizar esta etapa cada Canvas habrá dibujado la sección que le corresponda utilizando las APIS del framework cuando fuese necesario obtener algún dato.

Navegación dentro del sitio y su interacción con el framework:

Una vez cargado el sitio por completo, la navegación del mismo se produce a través de la lógica implementada en el objeto Silverlight y sus llamadas a las diferentes APIS del framework para obtener los datos que necesite.

Por ejemplo un clic en uno de los nodos del menú dispara una llamada al Frontpage controller quien traerá los datos con la composición de la tapa solicitada.

Una búsqueda disparará una llamada al SearchController quien traerá los datos de la búsqueda ingresada.

A continuación se ejemplifican llamadas a las diferentes APIS:

Buscador
http://plus.mixplay.tv/apis/search/list/words/guitarra/from/0/quantity/5

Ficha:
http://plus.mixplay.tv/apis/card/video/group_id/55053
Relacionados:
http://plus.mixplay.tv/apis/argentina/related/tags/group_id/10704/quantity/10

Ranking Más Vistos:
http://plus.mixplay.tv/apis/ranking/getfeatured/level_id/G/days/365/init_row/0/row_count/100
http://plus.mixplay.tv/apis/argentina/modelos/ranking/getfeatured/from/0/quantity/30

Arbol Channel:
http://plus.mixplay.tv/apis/argentina/home/nav/index/extended/1/format/xml/flagtype/channel

A continuación se muestra el resultado de una de las llamadas a las APIS:

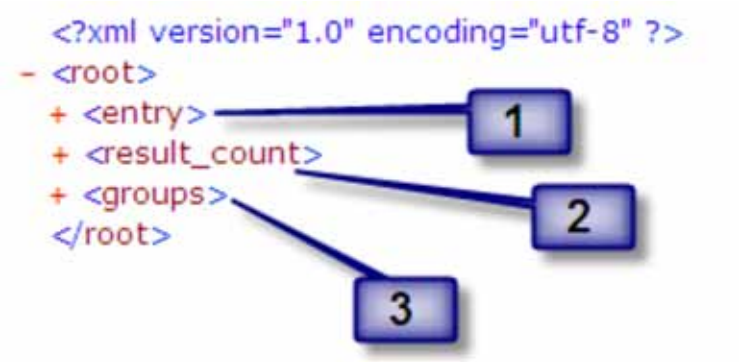
Llamada:
http://plus.mixplay.tv/apis/Ranking/getfeatured/level_id/G/days/365/from/0/quantity/5

En este ejemplo se llama al controlador Ranking con la acción getfeatured(Más Vistos), con los siguientes parámetros:

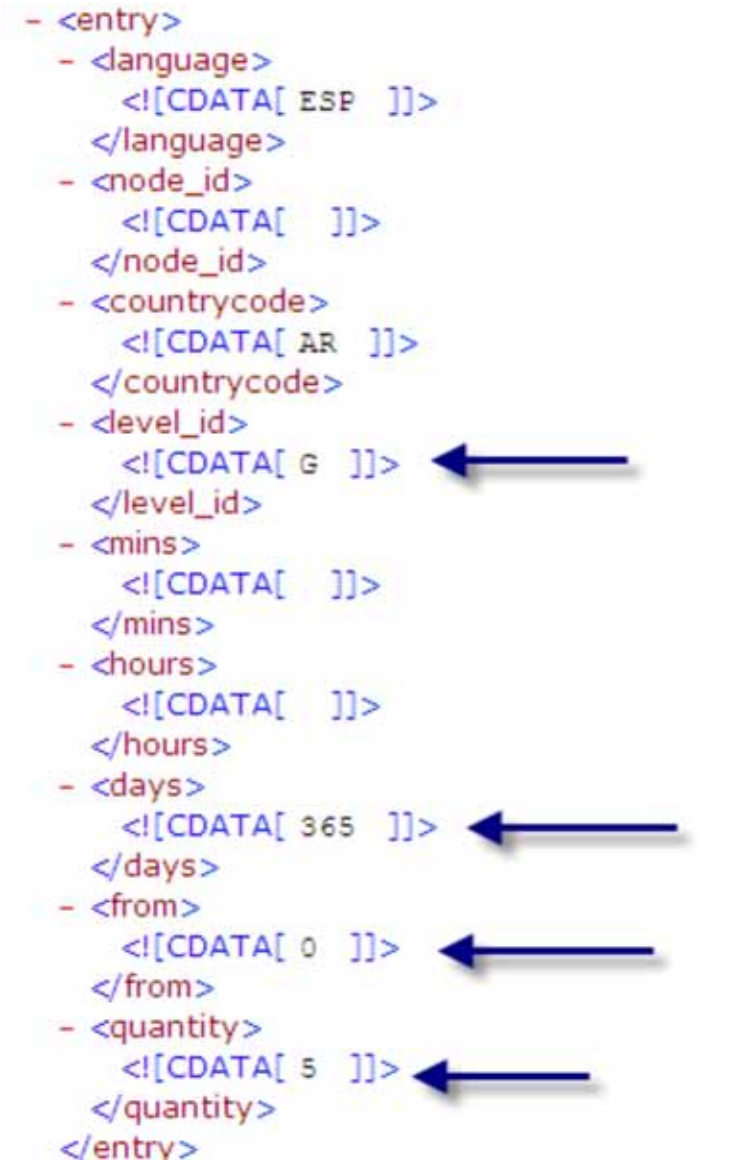
level_id -> G (se refiere a que traiga al material completo, con sus capítulos en caso de existir).
days -> 365 (los resultados que se traigan serán los más desde hoy a un año atrás).
from -> 0 (los resultados que se traigan empezarán desde el 1er elemento, este parámetro es utilizado por ejemplo para los paginados)
quantity -> 5 (se refiere a la cantidad de contenidos que debe traer)

El XML devuelto se divide en tres partes:

1- Parámetros de entrada, 2- Datos propios del controlador, 3- Datos de los contenidos.



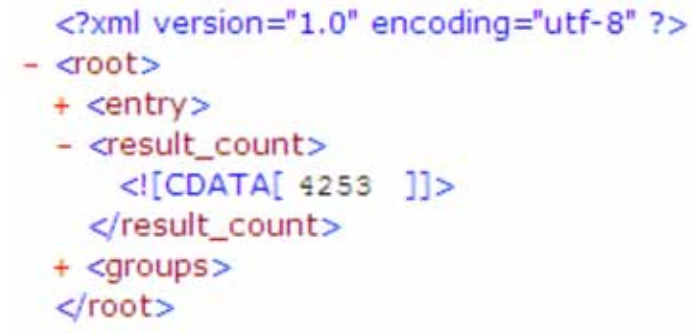
Parámetros de entrada:



En la imagen se destacan aquellos parámetros que se enviaron desde la URL, el resto de los parámetros, el controlador al no recibirlos toma valores por defecto.

Siempre que es invocada una API la misma devuelve los valores con la cual fue llamada. Esto se implementó para facilitar los testeos, identificar errores y facilitar los debugs.

Datos propios del controlador:



En este ejemplo al ser en definitiva un listado de contenidos, devuelve la cantidad total de resultados disponibles.

Datos de los Contenidos:

```
- <groups>
  - <group>
    - <links>
        <url_play href="http://www.mixplay.tv/videosonline/56895.php?playgroup&FWK[id_nodo]=&FWK[skin]=sk_mixplay" />
      </links>
      <common>
      - <position>
          <![CDATA[ 1 ]]>
        </position>
      - <id>
          <![CDATA[ 56895 ]]>
        </id>
      - <title>
          <![CDATA[ Veredicto en Gran Bretaña ]]>
        </title>
      - <description>
          <![CDATA[ Un jurado decide qué errores policiacos fueron los causantes de la muerte de un brasileño.
          Informe de CNN, Marta Altuna. (12 de diciembre) ]]>
        </description>
      - <copyright_id>
          <![CDATA[ 382 ]]>
        </copyright_id>
      - <copyright_image>
          <![CDATA[ /imagenes/fuente/esp_382_0_chica.gif ]]>
        </copyright_image>
      - <copyright_image_alt>
          <![CDATA[ CNN en español ]]>
        </copyright_image_alt>
      - <image_large>
          <![CDATA[ http://www.mixplay.tv/imagenes/video/materiales/esp_56895_grande.jpg ]]>
        </image_large>
      + <image_large_alt>
      + <image_medium>
      + <image_medium_alt>
      + <image_small>
      + <image_small_alt>
      - <duration>
          <![CDATA[ ]]>
        </duration>
      - <date>
          <![CDATA[ 13/12/2008 ]]>
        </date>
      - <media_type>
          <![CDATA[ 5 ]]>
        </media_type>
      - <ranking>
        - <views_count>
            <![CDATA[ 105 ]]>
          </views_count>
        - <votes_count>
            <![CDATA[ 0 ]]>
          </votes_count>
        - <average_votes>
            <![CDATA[ 0 ]]>
          </average_votes>
        </ranking>
      </common>
    - <media>
      - <artist_id>
          <![CDATA[ 0 ]]>
        </artist_id>
      </media>
    </group>
  + <group>
  + <group>
  + <group>
  + <group>
  </groups>
```


En esta sección del XML pueden destacarse dos partes: el área recuadrada, es denominada COMMON, y cada API que devuelva resultados que involucre contenidos de video, siempre debe traer estos datos, esto es para que cualquier aplicación que muestre un video ó un listado de los mismos, sin importar a que servicio invoque siempre pueda representarlos, al saber que siempre dispondrá del mismo conjunto de información. En definitiva una tapa podrá alimentarse con los rankings, al igual que un resultado del buscador, sin tener que generar ningún cambio más que la llamada a este controlador, debido a que las aplicaciones que dibujan la salida siempre dispondrán de los mismos elementos para renderizarse.

Por otro lado al final del XML pueden verse los otros 4 grupos que se pidieron en la consulta, si bien los mismos están cerrados tienen exactamente los mismos elementos que el grupo abierto.

Caso 2:

En este caso vamos a ver dos clientes diferentes que utilizando las APIS expuestas por el framework introducen contenidos en sus sitios.

Datos Generales:

Nombres del sitio: NOVULU – Videos en Español y RockRoad
URL de acceso: http://www.novulu.com/ y http://www.rockroadtv.com/videos.php

Características:

Si bien ambos sitios por lo que podemos apreciar al ingresar se encuentran desarrollados en PHP, desconocemos el tipo de implementación con lo que fueron armados, sin embargo podemos ver que la sección de videos de RockRoad y en la misma Home de Novulu se muestran listados ó rankings de videos que son obtenidos directamente de las APIS de nuestro framework.

Las mismas APIS que son consultadas por ambos sitios son también utilizadas por nuestro sitio expuesto en el caso número uno, por eso no es necesario realizar ningún agregado o configuración extra en el framework, es aquí donde se puede resaltar la sencillo que es reutilizar en este caso controllers ya desarrollados que devuelven su respuesta por ejemplo en formato XML. En caso que alguno de los clientes hubiese necesitado que el formato de salida fuera distinto, hubiese bastado con crear tan solo una nueva vista para el mismo componente con la salida deseada. De esta manera queda expuesta la separación entre las vistas y los controles y lo práctico de su uso.

Nota: En el anexo 2 es posible ver un documento que explica como llamar a la API de rankings para el caso de Novulu

7. Conclusiones Finales

A lo largo de este trabajo se mostró como es posible implementar un framework, donde la lógica de negocios, el acceso a datos y la capa de presentación queden separadas en diferentes capas sin afectar una a otra, utilizando patrones y tecnologías informáticas ya existentes, conocidas y probadas.

Se mostró como se seleccionó el Zend Framework para cumplir con este cometido y como fue posible adaptar el mismo a las necesidades de la organización extendiendo su funcionalidad, como ser el manejo de la base de datos, el uso del metalenguaje xsl para generar vistas o del memcached para el cacheo de las respuestas de la base de datos Oracle.

Se demostró la utilización del framework en casos reales con productos que hoy se encuentran en línea funcionando e innovando en el mercado con nuevas tecnologías como Microsoft Silverlight y que por su fisonomía requirieron el uso de diferentes formatos de salida para lograr el funcionamiento deseado.

Finalmente con el nuevo framework implementado, el cual posee una curva de aprendizaje corta para agregar nuevos módulos, de forma ordenada y nada restrictiva, permitirán a la compañía afrontar futuros desafíos sin tener que preocuparse por el alma que dará soporte a las nuevas implementaciones y de este modo podrá enfocarse más a los negocios y al cliente al sentirse respaldada por una estructura sólida que le permita crecer y extenderse.

8. Posibles Extensiones de la Tesina

- Agregarle al framework una metodología para casos de pruebas unitarios, por ejemplo una futura investigación podría analizar como agregarle al framework una herramienta como PhpUnit o alguna otra de características similares.

- Analizar la posibilidad de agregar un ORM manejado por el framework entre el modelo y la base de datos. ORM son las siglas de Object-Relational Mapping ó mapeo objeto-relacional y su función es abstraer la base de datos relacional, creando una base de datos virtual orientada a objetos.

9. Glosario

Apache: Es un servidor HTTP de código abierto para plataformas Unix, Windows y Macintosh, que implementa el protocolo HTTP/1.1

API: (Application Programming Interface - Interfaz de Programación de Aplicaciones), es el conjunto de funciones y procedimientos (o métodos si se refiere a programación orientada a objetos) que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.

Framework: Dentro del ámbito de sistemas Web, un framework es un software desarrollado con el propósito de facilitar y agilizar el desarrollo de sistemas Web, ofreciéndole a quien los use las herramientas necesarias para el desarrollo ágil de código. Generalmente los mismos se encuentran diseñados en diferentes patrones como por ejemplo el MVC.

Garbage Collector: Recoletor de Basura, es un mecanismo de gestión de memoria, generalmente encargado de liberar la memoria de los elementos no utilizados por una aplicación.

Helper [ZF]: Es una clase que hereda de la clase helper_abstract y esta pensada para poder extender funcionalidades a todos los controles, la cual puede ser solicitada a través de un objeto que implementa el patrón broker.

HTTP: El protocolo de transferencia de hipertexto (HTTP, HyperText Transfer Protocol) es el protocolo usado en cada transacción de la Web (WWW). Su desarrollo fue coordinado por la World Wide Web Consortium y la Internet Engineering Task Force.

include_path[PHP]: Especifica la lista de directorios donde deben buscar aquellas funciones que incluyen, abren o leen archivos.

MIME: (Multipurpose Internet Mail Extensions - Extensiones de Correo Internet Multipropósito), son una serie de convenciones o especificaciones dirigidas a que se puedan intercambiar a través de Internet todo tipo de archivos (texto, audio, vídeo, etc.) de forma transparente para el usuario.

Open Source: Es el término con el que se conoce al software distribuido y desarrollado libremente. Es un método de desarrollo de software que aprovecha al máximo el poder de la distribución, de revisión entre pares y transparencia del proceso, resultando en un software de mayor calidad, mayor fiabilidad, más flexibilidad y menor costo.

PHP: Es un lenguaje de programación interpretado, ampliamente usado y de propósito general pero especialmente utilizado en el desarrollo Web y puede se embebido dentro de código HTML.

Plugin [ZF]: Son objetos contemplados por el Zend Framework que mediante su uso permiten extender o modificar las funcionalidades del framework. Contemplan dos llamadas “preDispatch” y “postDispatch” y son instanciadas por el Front Controller, según un orden de prioridad configurable.

Silverlight, Microsoft: Es un complemento para navegadores de Internet basado en la plataforma Windows que agrega nuevas funciones multimedia como la reproducción de vídeos, gráficos vectoriales, animaciones y el mismo puede ser programado desde .net.

Singleton: Asegura que una clase solo tendrá una instancia, y provee un punto global de acceso a la misma.

Slab Allocator: Es un mecanismo ideado para el manejo más eficiente de memoria. Para este mecanismo la memoria se representa en tamaño de bloques fijos, de modo que si un objeto sale de memoria, la misma puede ser reutilizada por otro objeto del mismo tipo.

SOAP: (Simple Object Access Protocol – Protocolo Simple de Acceso a Objetos), es un protocolo para acceder a servicios Web a través de XML

SVN: Es un sistema de control de versionamiento de archivos, usado ampliamente para el desarrollo de software.

URI: (Uniform Resource Identifier - Identificador Uniforme de Recursos). Un URI es una cadena corta de caracteres que identifica inequívocamente un recurso (servicio, página, documento, dirección de correo electrónico, enciclopedia, etc.). Normalmente estos recursos son accesibles en una red o sistema.

URL: (Uniform Resource Locator - Localizador Uniforme de Recursos). Forma de organizar la información en la web. Es una secuencia de caracteres, de acuerdo a un formato estándar, que se usa para nombrar recursos, como documentos e imágenes en Internet, por su localización.

WSDL: (Web Services Description Language - Lenguaje para Describir Servicios Web), describe la interfaz pública a los servicios Web. Está basado en XML y describe la forma de comunicación, es decir, los requisitos del protocolo y los formatos de los mensajes necesarios para interactuar con los servicios listados en su catálogo. Las operaciones y mensajes que soporta se describen en abstracto y se ligan después al protocolo concreto de red y al formato del mensaje.

XAML: (Acrónimo pronunciado xammel del inglés eXtensible Application Markup Language - Lenguaje Extensible de Formato para Aplicaciones) es el lenguaje de formato para la interfaz de usuario para la Base de Presentación de Windows (WPF por sus siglas en inglés) y Silverlight (wpf/e), el cual es uno de los “pilares” de la interfaz de programación de aplicaciones .NET en su versión 3.0

10. Bibliografía

Patrones de Diseño Software.
En: LIBRO: GOF Erich Gamma, Richard Helm, Ralph Jonson y John Vlissides.

Patrón MVC en Sun [en línea]
En http://java.sun.com/blueprints/patterns/MVC-detailed.html [consulta: 15 de Noviembre 2008]

Patron MVC en Cunningham & Cunningham, Inc. [en línea]
En: http://c2.com/cgi/wiki y http://c2.com/cgi/wiki?ModelViewControllerAsAnAggregateDesignPattern [consulta: 20 de Noviembre 2008]

Wikipedia, definición Patrón MVC, [En línea]
En: http://en.wikipedia.org/wiki/Model-view-controller [consulta: 20 de Noviembre 2008]

Wikipedia, definición Patrón REST, [En línea]
En: http://en.wikipedia.org/wiki/REST [consulta: 3 de Diciembre 2008]

Patrón Rest en University of California [en línea]
En: http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm [consulta: 3 de Diciembre 2008]

Patrón Rest en Dos Ideas.com [en línea]
http://www.dosideas.com/java/314-introduccion-a-los-servicios-web-restful.html [consulta: 5 de Diciembre 2008]

Patrón Rest en O´Reilly Webservices [en línea]
http://webservices.xml.com/pub/a/ws/2002/02/20/rest.html [consulta: 5 de Diciembre 2008]

Danga Memcached [en línea]
En: http://www.danga.com/memcached/ [consulta: 6 de Diciembre 2008]

Zend Technologies Ltd, Zend Framework [en línea]
En: http://www.zend.com

Zend Technologies Ltd, Zend Front Controller [en línea]
http://framework.zend.com/manual/en/zend.controller.basics.html [consulta: 6 de Diciembre 2008]

Zend Developer Zone, Action Helpers[en línea]
En: http://devzone.zend.com/article/3350-Action-Helpers-in-Zend-Framework [consulta: 9 de Diciembre 2008]

Two Step View, Martin Fowler [en línea]
En: http://martinfowler.com/eaaCatalog/twoStepView.html [consulta: 11 de Diciembre 2008]

PHP Core php.ini directives, inlcude path [en línea]
En: http://ar.php.net/manual/en/ini.core.php#ini.include-path [consulta: 6 de Diciembre 2008]

Apache URL Rewriting Guide [en línea]
http://httpd.apache.org/docs/1.3/misc/rewriteguide.html [consulta: 12 de Diciembre 2008]

Wikipedía, La enciclopedia libre [en línea]
En: http://es.wikipedia.org/ y http://en.wikipedia.org/

Real Academia Española, Diccionario de la Real Academia Española [en línea]
En: http://www.rae.es

Silverlight 2
En LIBRO: Essential Silverlight 2 Up-to-Date – O´Reilly – Christian Wenz

Open source, The open source definiton [En línea]
En: http://www.opensource.org/

11. Anexos

Anexo I: Detalle del Front Controller en ZF:

A continuación se explica más detalladamente el funcionamiento del Front Controler dentro del Zend Framework, con los principales objetos que interviene en el proceso.

Punto de entrada: Server Request -> Zend_Controller_Front
-> (termina llamando al) Zend_Action_Controller.
El Zend_Controller_Front, orquesta el workflow completo del sistema Zend_Controller.
El Zend_Controller_Front procesa todos los pedidos recibidos por el servidor y su última responsabilidad es delegar estas peticiones al Action Controller.

Zend_Controller_Request_Abstract (el Request Object) quien por defecto utiliza Zend_Controller_Request_Http representa el entorno del pedido y provee lo métodos para setear u obtener el controlador, los nombres de las acciones y cualquier parámetro del pedido. Adicionalmente mantiene el registro de si las acciones que posee fueron o no despachadas por el Zend_Controller_Dispatcher

Zend_Controller_Router_Interface es usada para definir los routers. Es el proceso de examinar el entorno del pedido para determinar que controlador y acción deben recibir al pedido.
Este controlador, acción y parámetros opcionales son seteados en el object request para ser procesados por el Zend_Controller_Dispatcher_Standard.
El routing ocurre solamente una vez: cuando el pedido es inicialmente recibido y antes de que el primer controlador sea despachado.

El router por defecto, Zend_Controller_Router_Rewrite, toma la URI como especifica el Zend_Controller_Request_Http y la descompone en controlador, acción y parámetros.
Ej: http://localhost/ranking/masvistos/fecha_desde/X/fecha_hasta/Y va a ser decodificada para usar el controlador ranking, la acción masvistos, y los parámetros fecha_desde con valor X y fecha_hasta con valor Y.

Zend_Controller_Dispatcher_Interface define a los despachadores.
El despacho es el proceso de obtener o sacar el controlador y la acción del request object y mapearlo al archivo/clase del controlador y al método de la acción en la clase del controlador. Si el controlador o la acción no existe, se maneja determinando que controlador default y acción despachar.

A diferencia del routing que ocurre únicamente una vez, el dispatching ocurre en un ciclo.

Zend_Controller_Action es la acción base del componente del control. Cada controlador es un clase única que extiende a la clase Zend_Controller_Action y puede contener uno o más métodos de acción.

Zend_Controller_Response_Abstract define una clase base de respuestas (base response class) usada para colectar y retornar las respuestas de los controladores de acción. Colecta los headers y los contenidos del body

Anexo 2: Llamada a APIS para Novulu:

Códigos de Secciones. [SECCION].

Vamos a contar con una portada y cuatro secciones. Estos códigos deberán ser utilizados en las llamadas a las APIS para traer datos de las secciones disponibles.

Portada: novulu
Novelas: nvl_novelas
Películas: nvl_peliculas
Humor: nvl_humor
Música: nvl_musica

APIS:

Para obtener los Más Nuevos:

Con esta acción se obtendrá los contenidos más nuevos de acuerdo a un corte o rango de tiempo y otros parámetros que se detallan a continuación.

Parámetros:
Representan el rango de tiempo que se va a tener en cuenta en el resultado de la consulta. Estos parámetros son excluyentes, por lo tanto, no se deben utilizar combinados. Alguno de ellos en obligatorio en la consulta.

days: Es la cantidad de días hacia atrás, desde la fecha actual. El rango de valores que puede tomar es de 1 a 365.

hours: Es la cantidad de horas hacia atrás, desde la hora actual. El rango de valores que puede tomar es de 1 a 23.

mins: Es la cantidad de minutos en un rango de 5 a 59.

from: Del universo de contenidos que coincide con la consulta, este parámetro, indico a partir de que contenido quiero mostrar. El rango de valores que puede tomar son todos los enteros mayores o iguales que 0. Debería pasarse siempre 0, para obtener todos los que coinciden con la consulta.

quantity: Este parámetro es opcional y representa la cantidad de contenidos a mostrar, En caso de no recibir parámetro por defecto es 20.

level_id: Este parámetro es opcional y es utilizado para diferenciar si los elementos a devolver es un contenido o un grupo. Toma 2 valores C para contenido o G para grupo. Debería usarse el modo C, en el caso que se desee utilizar la funcionalidad de embed de MIXPLAY.

client: Es un número interno que maneja MIXPLAY, que debe ser recibido dentro de los parámetros.

Prototipo:

http://[dominio]/[SECCION]/Ranking/getfeatured/level_id/C/days/{valor}/from/{valor}/quantity/{valor}/client/10001
Ejemplos:
Los 100 más nuevos de los últimos 365 días de novulu:
http://[dominio]/novulu/Ranking/getfeatured/level_id/C/days/365/from/0/quantity/100/client/10001

Los 20 más nuevos de los últimos 7 días de novulu:
http://[dominio]/novulu/Ranking/getfeatured/level_id/C/days/7/from/0/quantity/20/client/10001

Los 20 más nuevos de los últimos 7 días de novulu Novelas:

http://[dominio]/nvl_novelas/Ranking/getfeatured/level_id/C/days/7/from/0/quantity/20/client/10001

Los 20 más nuevos de los últimos 7 días de novulu Películas:
http://[dominio]/nvl_peliculas/Ranking/getfeatured/level_id/C/days/7/from/0/quantity/20/client/10001

Los 20 más nuevos de los últimos 7 días de novulu Humor:
http://[dominio]/nvl_humor/Ranking/getfeatured/level_id/C/days/7/from/0/quantity/20/client/10001

Para obtener los Más Vistos:

Con esta acción se obtendrá los contenidos mas vistos de acuerdo a un corte o rango de tiempo y otros parámetros que se detallan a continuación.

Parámetros:

precision: Es un parámetro obligatorio e indica período a considerar

- MINUTO (precision='MI')
- HORA (precision='HO')
- DIA (precision='DI')
- MES (precision='ME')

La precisión ME (meses) ó DI (días) es un parámetro mandatario y acepta las siguientes combinaciones:

date_from: Es un parámetro opcional que indica la fecha de menor valor en el caso que se necesite un rango de fecha, el formato es del tipo 'DDMMYYYY'. Este parámetro será obligatorio solo para el caso que precision='DI' o 'ME'.

date_to: Es un parámetro opcional que indica la fecha de mayor valor en el caso que se necesite un rango de fecha, el formato es del tipo 'DDMMYYYY'. Este parámetro será obligatorio solo para el caso que precision='DI' o 'ME'.

quantity_period: Es un parámetro opcional que indica la cantidad de días o meses, a tener en cuenta en la consulta dependiendo de precisión sea DI o ME respectivamente. El valor no puede superar la diferencia de la fecha actual y 01/01/2004 en la unidad seleccionada (o sea, días o meses).

- Combinaciones validas para precisión = 'DI' o 'ME'
  - (date_from,date_to) à especifica período entre fechas.
  - (date_to,quantity_period) àhasta que fecha y la cantidad de días ó meses hacia atrás
  - (quantity_period) à desde ahora hacia atrás en días o meses.

mins: Es un parámetro opcional que indica la cantidad de minutos a considerar desde el ultimo minuto cerrado, el rango de valores que puede tomar es de 1 a 60. Este parámetro será obligatorio solo para el caso que precision='MI'.

hours: Es un parámetro opcional que indica la cantidad de horas a considerar desde la ultima hora cerrada, el rango de valores que puede tomar es de 1 a 23. Este parámetro será obligatorio solo para el caso que precision='HO'.

from: Del universo de contenidos que coincide con la consulta, este parámetro, indico a partir de que contenido quiero mostrar. El rango de valores que puede tomar son todos los enteros mayores o iguales que 0. Debería pasarse siempre 0, si quiero todos los que coinciden con la consulta.

quantity: Este parámetro es opcional y representa la cantidad de contenidos a mostrar, En caso de no recibir parámetro por defecto es 20.

level_id: Es un parámetro obligatorio utilizado para diferenciar si los elementos a devolver es un contenido o un grupo. Toma 2 valores C para contenido o G para grupo. Debería usarse el modo C, en el caso que se desee utilizar la funcionalidad de embed de MIXPLAY.

client: Es un número interno que maneja MIXPLAY, que debe ser recibido dentro de los parámetros.

Prototipo:
http://[dominio]//[SECCION]/Ranking/getmostviewed/level_id/C/precision/{valor}/quantity_period/{valor}/from/{valor}/quantity/{valor}/client/10001
Ejemplos:
Los 100 contenidos más vistos de los últimos 2 meses para Novulu:
http://[dominio]/novulu/Ranking/getmostviewed/level_id/C/precision/ME/quantity_period/2/from/0/quantity/100/client/10001

Los 100 contenidos más vistos de los últimos 2 días para Novulu:
http://[dominio]/novulu/Ranking/getmostviewed/level_id/C/precision/DI/quantity_period/2/from/0/quantity/100/client/10001

Los 100 contenidos más vistos de los últimos 2 días para Novulu Novelas:
http://[dominio]/nvl_novelas/Ranking/getmostviewed/level_id/C/precision/DI/quantity_period/2/from/0/quantity/100/client/10001

Los 100 contenidos más vistos de los últimos 2 días para Novulu Películas:
http://[dominio]/nvl_peliculas/Ranking/getmostviewed/level_id/C/precision/DI/quantity_period/2/from/0/quantity/100/client/10001

Los 100 contenidos más vistos de los últimos 2 días para Novulu Humor:
http://[dominio]/nvl_humor/Ranking/getmostviewed/level_id/C/precision/DI/quantity_period/2/from/0/quantity/100/client/10001

Los 10 contenidos más vistos de los últimos 2 días a partir del 31/08/2008 para Novulu:
http://[dominio]/novulu/Ranking/getmostviewed/level_id/C/precision/DI/quantity_period/2/date_to/31082008/from/0/quantity/10/client/10001

Los 20 contenidos mas vistos desde 10/01/2008 a 26/09/2008 para Novulu:
http://[dominio]/novulu/Ranking/getmostviewed/precision/DI/date_from/10012008/date_to/26092008/level_id/C/client/10001

Los 20 contenidos mas vistos 100 días para atrás desde el 26/09/2008 de Novulu:
http://[dominio]/novulu/Ranking/getmostviewed/precision/DI/date_to/26092008/quantity_period/100/level_id/C/client/10001

Los 20 contenidos mas vistos 10 meses para atrás desde la fecha actual para Novulu:
http://[dominio]/novulu/Ranking/getmostviewed/precision/ME/quantity_period/10/level_id/C/ client/10001

Ejemplo del XML de salida:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
- <root>
- <entry>
  <language>esp</language>
  <action>tops</action>
  <node>rockroad</node>
  <level>C</level>
  <days>0</days>
  <hours>4</hours>
  <minutes>0</minutes>
  <from>1</from>
  <quantity>20</quantity>
  <date>15/10/2007 14:30:32</date>
  </entry>
```

- <contents>
- <content>
- <links>
<url_play href="http://PAGE_DOMAIN/template/233345.php?playcontent" />
<url_embed href="http://PAGE_DOMAIN/template/233345.php?playext" />
</links>
- <common>
<position>1</position>
<id>233345</id>
- <title>
- <![CDATA[Gran expectativa por el duelo entre Brasil y Uruguay
]]>
</title>
- <description>
- <![CDATA[Uno de los clásicos sudamericanos se reedita este miércoles en San Pablo, por las Eliminatorias.
]]>
</description>
<copyright_id>123</copyright_id>
- <copyright_image>
- <![CDATA[http://PAGE_DOMAIN/imagenes/fuentes/foxsports.jpg
]]>
</copyright_image>
- <copyright_image_alt>
- <![CDATA[Rockroad
]]>
</copyright_image_alt>
- <image_large>
- <![CDATA[http://PAGE_DOMAIN/imagenes/video/materiales/esp_233345_grande.jpg
]]>
</image_large>
<image_large_alt />
- <image_medium>
- <![CDATA[http://PAGE_DOMAIN/imagenes/video/materiales/esp_233345_mediana.jpg
]]>
</image_medium>
<image_medium_alt />
- <image_small>
- <![CDATA[http://PAGE_DOMAIN/imagenes/video/materiales/esp_233345_chica.jpg
]]>
</image_small>
- <duration>
- <![CDATA[00:00:30
]]>
</duration>
- <date>
- <![CDATA[11/01/2007
]]>
</date>
- <media_type>
- <![CDATA[2
]]>
</media_type>
- <ranking>
- <views_count>
- <![CDATA[155
]]>
</views_count>

```
</ranking>
</common>
</content>
</contents>
</root>
```