Universidad de las Ciencias Informáticas

“Facultad 4”

Título: Desarrollo del Acceso a Datos para el Polo Sistemas Tributarios y de Aduanas.

Trabajo de Diploma para optar por el título de

Ingeniero en Ciencias Informáticas

Autor:

Dennis Vázquez Prida.

Tutor:

Msc. Julio César Díaz Vera.

“ Ciudad de la Habana, Junio del 2008 ”.

Declaración de autoría

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

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

Dennis Vázquez Prida	Msc. Julio C. Díaz Vera
_____	_____
Firma del Autor	Firma del Tutor

Datos de Contacto

Tutor: Msc. Julio C. Díaz Vera.

- Graduado de Ing. en Telecomunicaciones y Electrónica en el año 2003 en la Universidad Central de Las Villas.
- Se titula como Máster en Gestión de Proyectos en la Universidad de las Ciencias Informáticas. en el año 2007.
- Participó en el primer Taller de Minería de Datos Louisana – Cuba en el año 2005.
- Ha participado como ponente en las dos ediciones de UCIENCIA, publicando en las memorias de ambos eventos.

Agradecimientos

Al profe Julio por todas sus ideas y horas de revisión.

Dedicatoria

A mi papá, mi mamá y a mi hermano, por sentirme orgulloso de ellos.

A mis abuelos todos, por la sobrada naturalidad y fuerza desbordada en el cursar de sus vidas.

A mi familia y a mis amigos.

Resumen

La Aduana General de la República de Cuba tiene funcionando varios sistemas informáticos con el objetivo de controlar y gestionar sus procesos. Contando además con la necesidad del desarrollo de otros sistemas informáticos que se incorporen para facilitar la eficiencia y profesionalidad del trabajo.

Para ello se ha establecido una arquitectura de software estándar bajo la cual se desarrollarán todos los nuevos sistemas en desarrollo y se cambiarán de forma paulatina los ya establecidos, a estos mismos principios. Para esta nueva arquitectura se hace necesario el establecimiento de un modelo de acceso a datos que responda a la misma.

Este trabajo propone y valida el Modelo de Acceso a Datos a utilizar en los Sistemas Tributarios y de Aduanas desarrollados bajos los paradigmas de la nueva arquitectura. Basándose para ello en el estudio de los mecanismo de acceso a fuentes de datos para de esta forma establecer un mecanismo propio y la comprobación de la viabilidad del mismo.

Palabras Claves:

Modelo de Acceso a Datos, Arquitectura de software, Sistemas Tributarios y de Aduanas.

Índice

Introducción.

La Aduana General de la República de Cuba es una organización que controla el tráfico internacional de medios de transportes, mercancías y viajeros, tanto a la entrada como a la salida del país. Estas actividades están acompañadas de la profesionalidad y eficiencia de sus trabajadores. Impedir la entrada o salida de productos y artículos que atenten contra la seguridad de la sociedad cubana, facilitar el control del comercio legítimo, el procesamiento de sus estadísticas y la recaudación fiscal, están dentro de sus responsabilidades. A ellas se unen proteger al país del tráfico ilegal de armamentos, explosivos, drogas y sustancias químicas, objetos del patrimonio cultural, especies protegidas y otras, sin abandonar el cumplimiento de la política comercial, el enfrentamiento a la evasión fiscal y otros fraudes económicos.

En la actualidad la Aduana General de la República de Cuba (AGR) tiene funcionando diferentes sistemas informáticos que facilitan el control y la gestión de estos procesos. Estos sistemas difieren en el lenguaje de programación que fueron desarrollados, plataformas y sistemas de bases de datos que utilizan.

Con el objetivo de integrar todos estos sistemas de información la *AGR*[1] apoyado por el Centro de Automatización para la Información y la Dirección (CADI) comenzó el desarrollo del Sistema Único de Aduana (SUA) encargado de unificar en un solo sistema todos los demás sistemas ya instaurados, de forma tal que se garantice la comunicación de forma rápida, eficiente y segura entre ellos. Desde hace más de tres años a petición de la AGR se desarrolla el *SUA*[2] por parte del *CADI* [3] en colaboración con la Universidad de las Ciencias Informáticas, incluyendo además el desarrollo de nuevos sistemas según las necesidades de la AGR.

El SUA es un sistema que en un principio ha sido implementado en PHP de forma estructurada y sin hacer usos de buenas prácticas de programación, probadas como aceptadas, y formalizadas en patrones de diseño tanto arquitectónicos como a nivel de aplicación. Al no hacer uso de una arquitectura bien definida en capas existe una gran mezcla del código de la lógica del negocio tanto en la presentación como en la base de datos violando el principio de compartimentación, prácticamente

[1] AGR: Aduana General de la República de Cuba.

[2] SUA: Sistema Único de Aduana.

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

un tabú en el desarrollo de software en la actualidad. Aspectos estos que atentan contra la flexibilidad, usabilidad, mantenimiento y rendimiento del sistema actual.

La forma en que se accede a los datos en el sistema actual no se encuentra ajena a la presencia de problemas producto al no empleo de buenas prácticas de programación, pues el acceso a los datos se realiza mediante procedimientos almacenados, por lo que gran parte de la lógica del negocio se encuentra en la base de datos, no existe una capa de abstracción de los datos por lo que las consultas se mueven desde la lógica del negocio hasta la base de datos en formato SQL. Afectando de esta manera la seguridad y reusabilidad del sistema.

Es decir el sistema carece de un modelo de acceso a datos que facilite la utilización de buenas prácticas de programación basadas en estándares y que logre un alto nivel de abstracción entre las capas de negocio y la de datos.

La no definición de un Modelo de Acceso a Datos trae como consecuencia que los desarrolladores del sistema deban implementar en otras capas las funcionalidades del acceso a fuentes de datos, empleando en esta actividad tiempo de desarrollo y esfuerzo extra, la transparencia y flexibilidad del sistema también se ven afectada pues futuros cambios en la implementación implicaría el cambio de gran cantidad de código.

El entendimiento de la problemática anterior ha desencadenado un esfuerzo en aras de un desarrollo acorde con los paradigmas actuales de la programación para los nuevos sistemas del Polo Sistemas Tributarios y de Aduanas, acordando además ir sustituyendo poco a poco los ya existentes por versiones que respondan a estos mismos estándares. Empleando para ello la programación orientada a objetos *(POO[4]),* el uso del patrón Modelo-Vista-Controlador (MVC) y la definición de una capa de abstracción de los datos que permita abstraer al programador del negocio de la base de datos. Una vez establecido el marco arquitectónico de trabajo apoyado por el patrón MVC, se hace necesario el establecimiento de un Modelo de Acceso a Datos que facilite un alto nivel de abstracción entre las capas de negocio y la de datos, por lo que se espera que este trabajo facilite el establecimiento de un marco de trabajo de referencia para el acceso a datos en la arquitectura definida en el *PSTA[5].* Surgiendo de esta forma el siguiente problema:

¿Cómo modelar un mecanismo de acceso a datos para los sistemas del PSTA?

[4] POO: Programación Orientada a Objetos.

[5] Polo Sistemas Tributarios y de Aduanas.

Teniendo como **objeto de estudio:** Los Modelos de Acceso a Datos.

Campo de acción: El Acceso a Datos en los proyectos que conforman el PSTA.

Siendo el **objetivo general** del trabajo: Modelar un mecanismo de Acceso a Datos para el PSTA.

Para alcanzar el objetivo trazado se debe dar cumplimiento a un grupo de **tareas** esenciales, las cuales son:

- Realizar el estudio de los modelos teóricos de acceso a datos.
- Realizar el estudio de las implementaciones de los modelos de acceso a datos en plataformas *LAMP*[6].
- Escoger los patrones de acceso a datos que respondan a las necesidades del PSTA.
- Implementar el modelo escogido.

El trabajo está estructurado en tres capítulos, tal y como se describe a continuación:

Capítulo 1: Se enuncian los principales conceptos relacionados con los Modelos de Acceso a Datos. Se realiza un estudio de algunas soluciones existentes para acceder a las fuentes de datos. Concluyendo con la selección de la herramienta de ayuda para desarrollar el Acceso a los Datos.

Capítulo 2: Se propone el Modelo de Acceso a Datos a utilizar por el PSTA y se argumenta de forma teórica el por qué de la propuesta.

Capítulo 3: Se valida de forma práctica el Modelo de Acceso a Datos propuesto y se corrobora sus argumentos teóricos.

[6] LAMP: Linux, Apache, Mysql y PHP.

Capítulo 1: Fundamentación Teórica.

1.1 Introducción.

Portabilidad, eficiencia y amplias posibilidades para la automatización de los procesos son las necesidades imperantes en el mundo empresarial y por ende la demanda de este tipo software ha crecido considerablemente. Debido a las capacidades de comunicación a través de internet y la utilización de estándares abiertos de comunicación las tecnologías Web están acaparando la mayor parte de este mercado de desarrollo de software. Brindando a las empresas amplias ventajas para llevar a cabo los procesos de marketing, administración, prestación de servicios, gestión de información y publicidad entre otros.

En el desarrollo de las aplicaciones Web uno de los modelos de arquitecturas más usado, es el modelo de arquitectura en capas, el cual se ha convertido actualmente en el modelo por excelencia en el desarrollo de software para la gestión empresarial ya que en este modelo, cada capa compone un subsistema en el que se ubican clases con responsabilidades propias, donde sus objetivos son facilitar la reutilización de código y componentes gracias a la herencia y encapsulamiento, que brindan los lenguajes de POO. Esto brinda grandes beneficios a las aplicaciones que utilizan esta arquitectura, como los siguientes:

- Aislamiento de la lógica de aplicaciones en componentes independientes, susceptibles de ser reutilizados después en otros sistemas.
- Asignación de recursos a cada una de las capas por separado, brindando la posibilidad de desarrollarlas en paralelo.
- Independencia del acceso a los datos en relación con la lógica del negocio.

La principal ventaja que tiene esta arquitectura es que puede cambiarse la capa de presentación sin que cambie la capa del negocio. Además el acceso a datos también puede ser trasladado a una base de datos distinta, a otro modelo de base de datos o a otro servidor sin influir en la capa de negocio. La propia capa de negocio también puede a su vez ser reemplazada sin influir estos cambios

significativamente en las capas de datos y presentación. Dentro de esta arquitectura en capas, la capa de acceso a datos es fundamental ya que es la encargada de hacer persistir la información que se maneja en el negocio y realizar todas las operaciones demandadas con ella.

1.2 Tendencias del Acceso a Datos.

La necesidad de hacer persistir la información manipulada por un sistema de información es tan antigua como los sistemas de información en sí mismos. Los mecanismos para resolver esta problemática han sido disímiles a través de la historia, pasando desde tratamiento de ficheros simples hasta complejos modelos de bases de datos relacionales.

Los métodos para el acceso a los datos persistidos también han sido varios a través del tiempo desde el uso de funciones dentro del propio código del software para el trabajo con ficheros, el empleo de sentencias SQL de forma mezclada en el código hasta la tendencia actual del uso de framewoks de persistencia, estos dos últimos métodos para el trabajo con las bases de datos relacionales.

El empleo de ficheros como medio de almacenamiento y la manera en que se accede a ellos resuelve en alguna medida la persistencia de datos y el acceso a ellos pero atenta contra buenas prácticas desde el punto de vista de programación, además que los beneficios arrojados son muy limitados debido a muchas deficiencias presentadas dentro de las que se encuentran [14]:

- **Separación y aislamiento de los datos**. Cuando los datos se separan en distintos ficheros, es más complicado acceder a ellos, ya que el programador de las aplicaciones debe sincronizar el procesamiento de los distintos ficheros implicados para asegurar que se extraen los datos correctos.

- **Duplicación de datos.** La redundancia de datos existente en los sistemas de ficheros, hace que se desperdicie espacio de almacenamiento y lo que es más importante, puede llevar a que se pierda la consistencia de los datos. Se produce una inconsistencia cuando copias de los mismos datos no coinciden.

- **Dependencia de datos**. Ya que la estructura física de los datos (la definición de los ficheros y de los registros) se encuentra codificada en los programas de aplicación, cualquier cambio en dicha estructura es difícil de realizar. El programador debe identificar todos los programas afectados por este cambio, modificarlos y volverlos a probar, lo que cuesta mucho tiempo y está sujeto a que se produzcan errores. A este problema, tan característico de los sistemas de ficheros, se le denomina también falta de independencia de datos lógica-física.

- **Formatos de ficheros incompatibles**. Ya que la estructura de los ficheros se define en los programas de aplicación, es completamente dependiente del lenguaje de programación. La incompatibilidad entre ficheros generados por distintos lenguajes hace que los ficheros sean difíciles de procesar de modo conjunto.

- **Consultas fijas y proliferación de programas de aplicación**. Desde el punto de vista de los usuarios finales, los sistemas de ficheros fueron un gran avance comparados a los sistemas manuales. A consecuencia de esto, creció la necesidad de realizar distintos tipos de consultas de datos. Sin embargo, los sistemas de ficheros son muy dependientes del programador de aplicaciones, cualquier consulta o informe que se quiera realizar debe ser programado por él.

Como solución a estas deficiencias en el empleo de los ficheros y los problemas para el acceso a su información surgen los sistemas de bases de datos. No hay un momento concreto en que los sistemas de ficheros hayan cesado y hayan dado comienzo a los sistemas de bases de datos, de hecho todavía existen sistemas de ficheros en uso.

Se dice que los sistemas de bases de datos tienen sus raíces en el proyecto estadounidense Apolo de mandar al hombre a la luna, en los años sesenta. En aquella época, no había ningún sistema que permitiera gestionar la inmensa cantidad de información que requería el proyecto. La primera empresa encargada del proyecto, NAA (North American Aviation), desarrolló un software denominado GUAM (General Update Access Method) que estaba basado en el concepto de que varias piezas pequeñas se unen para formar una pieza más grande, y así sucesivamente hasta que el producto final está ensamblado. Esta estructura, que tiene la forma de un árbol, es lo que se denomina una estructura jerárquica. A mediados de los sesenta, IBM se unió a NAA para desarrollar GUAM en lo que ahora se conoce como IMS (Information Management System). El motivo por el cual IBM restringió IMS al

manejo de jerarquías de registros fue el de permitir el uso de dispositivos de almacenamiento serie, más exactamente las cintas magnéticas, ya que era un requisito del mercado por aquella época.

A mitad de los sesenta, se desarrolló IDS (Integrated Data Store), de General Electric. Este trabajo fue dirigido por uno de los pioneros en los sistemas de bases de datos, Charles Bachmann. IDS era un nuevo tipo de sistema de bases de datos conocido como sistema de red, que produjo un gran efecto sobre los sistemas de información de aquella generación. El sistema de red se desarrolló, en parte, para satisfacer la necesidad de representar relaciones de datos más complejas que las que se podían modelar con los sistemas jerárquicos, y, en parte, para imponer un estándar de bases de datos. Para ayudar a establecer dicho estándar, CODASYL (Conference on Data Systems Languages), formado por representantes del gobierno de EEUU y representantes del mundo empresarial, formaron un grupo denominado DBTG (Data Base Task Group), cuyo objetivo era definir especificaciones estándar que permitieran la creación de bases de datos y el manejo de los datos. El DBTG presentó su informe final en 1971 y aunque éste no fue formalmente aceptado por ANSI (American National Standards Institute), muchos sistemas se desarrollaron siguiendo la propuesta del DBTG. Estos sistemas son los que se conocen como sistemas de red, o sistemas CODASYL o DBTG. [14]

Los sistemas jerárquico y de red constituyen la primera generación de los *SGBD*[7]. Pero estos sistemas presentaban algunos inconvenientes:

- Era necesario escribir complejos programas de aplicación para responder a cualquier tipo de consulta de datos, por simple que esta sea.
- La independencia de datos era mínima.
- No tenían un fundamento teórico.

En 1970 Edgar Frank Codd, de los laboratorios de investigación de IBM, escribió un artículo presentando, el modelo relacional. En este artículo, presentaba también los inconvenientes de los sistemas previos, el jerárquico y el de red. Entonces se comenzaron a desarrollar muchos sistemas relacionales, apareciendo los primeros a finales de los setenta y principios de los ochenta. Uno de los primeros es System R, de IBM, que se desarrolló para probar la funcionalidad del modelo relacional,

[7] SGBD: Sistema Gestor de Base de Datos.

proporcionando una implementación de sus estructuras de datos y sus operaciones. Esto condujo a dos grandes desarrollos:

- El desarrollo de un lenguaje de consultas estructurado denominado *SQL*[8], que se ha convertido en el lenguaje estándar de los sistemas relacionales.
- La producción de varios SGBD relacionales durante los años ochenta, como DB2 y SLQ/DS de IBM, y ORACLE de ORACLE Corporation.

Hoy en día, existen varios SGBD relacionales, tanto para microordenadores como para sistemas multiusuario, aunque muchos no son completamente fieles al modelo relacional.

Otros sistemas relacionales multiusuario son INGRES de Computer Associates, Informix de Informix Software Inc. y Sybase de Sybase Inc. Ejemplos de sistemas relacionales de microordenadores son Paradox y dBase IV de Borland, Access de Microsoft, FoxPro y R: base de Microrim.

Los SGBD relacionales constituyen la segunda generación de los SGBD. Sin embargo, el modelo relacional también tiene sus fallos, siendo uno de ellos su limitada capacidad al modelar los datos. Se ha hecho mucha investigación desde entonces tratando de resolver este problema. En 1976, Peter Chen presentó el modelo entidad-relación, que es la técnica más utilizada en el diseño de bases de datos. En 1979, Edgar Frank Codd intentó subsanar algunas de las deficiencias de su modelo relacional con una versión extendida denominada RM/T (1979) y RM/V2 (1990). Los intentos de proporcionar un modelo de datos que represente al mundo real de un modo más fiel han dado lugar a los modelos de datos semánticos.

En un principio el acceso a los datos de estas bases de datos relacionales, al no existir un modelo arquitectónico de software, se hacia de manera mezclada en todo código del programa, conjuntamente con el código de la presentación y del negocio se encontraban las sentencias SQL encargadas del trabajo con los datos persistentes, no había una limitación evidente, solo se programaba con el objetivo de dar respuestas a necesidades pero paulatinamente estas prácticas van cambiando a medida que aumentan las complejidades de los sistemas, evolucionan las tecnologías tanto en el software como en el hardware, y la aparición de los lenguajes de programación de alto nivel dando paso al empleo

[8] SQL: Leguaje de Consulta Estructurado (Structured Query Language).

de la arquitectura en capas, donde ya el uso de estas sentencias SQL se encuentra encapsulado en una capa llamada capa de acceso a datos, es decir de una manera más organizada.

Dentro de la misma capa de acceso a datos surge una nueva capa, la capa de abstracción de la base de datos encargada de servir de interfaz entre la capa de acceso a datos y la base de datos que permite mediante varios drivers mantener la conexión con la base de datos independientemente de la que se use. La utilización de esta capa arroja como principal ventaja, poder migrar de gestor de base de datos sin traer consecuencia alguna, ejemplo de estas capas de abstracción son PDO (PHP Data Object) y Creole.

Es decir ya se podía programar el acceso a datos mediante sentencias SQL sin tener en cuenta para qué base de datos se está programando.

Todas estas soluciones ingeniosas van dando solución a los problemas y van facilitando en cuanto a tiempo, comodidad y organización de trabajo a los desarrolladores pero existe un nuevo problema, cuya solución implicaría la utilización de buenas prácticas de programación basadas en estándares donde el modelo relacional se pueda trabajar de forma orientada a objeto desde los lenguajes de programación de alto nivel y lograr un alto nivel de abstracción entre las capas de negocio y la de datos.

1.3 Diferencias entre el modelo relacional y el modelo de objeto.

El modelo relacional trata con relaciones y conjuntos por lo cual tiende a ser de carácter matemático, mientras que el modelo de la POO trata con objetos, su carácter y las asociaciones entre ellos, el problema entre estos dos modelos surge en el momento de querer persistir los objetos del negocio.

Asumiendo que para el desarrollo de las aplicaciones actuales se emplean lenguajes de programación ampliamente distribuido y basado en objetos, lo normal en ese ámbito sería abrir una conexión, crear una sentencia en SQL y copiar todos los valores de las propiedades del objeto en la sentencia, para posteriormente ejecutarla y así almacenar el objeto, esto puede resultar sencillo para un simple caso, pero qué pasa si el objeto posee muchas propiedades, o bien se necesita almacenar un objeto que a su vez posea una colección de otros elementos, el desarrollo de aplicaciones en este caso requerirá de mucho más código, aparte de un tedioso trabajo en cuanto a la creación de sentencias en SQL.

Este mismo problema pasa también en la etapa de carga del objeto, es decir, cuando se recupera de la base de datos, si se asume que se necesita cargar un objeto que posee una colección de objetos del mismo tipo, o de otro tipo, qué se debe hacer, ¿Cargar solamente el objeto?, ¿Cargar también la colección entera?, o bien ¿Cargarla en otro momento?, aún cuando lo más natural sería cargar todo el objeto y a su vez la colección ello implicaría un gran aumento en el código que permite recuperar al objeto de la base de datos donde ha sido persistido.

Este problema se conoce con el nombre de *"Impedancia Objeto-Relacional"* y se define como un conjunto de dificultades técnicas que surgen cuando una base de datos relacional se usa en conjunto con un programa escrito bajo el paradigma de la POO, un ejemplo claro de esta impedancia se observa en el hecho que en el ambiente de la POO, se tiene un claro sentido de la pertenencia, a cada objeto le pertenecen sus correspondientes atributos.

Así como lo anteriormente expuesto se evidencian otros problemas que surgen entre estos dos modelos:

- **Reglas de acceso**: En el modelo relacional los atributos pueden ser accedidos y/o modificados a través de operadores relacionales predefinidos, mientras que en el modelo orientado a objetos, se permite que cada clase defina la forma en que serán alterados los atributos.
- **Ataduras del esquema**: Los objetos del modelo de la POO, no deben seguir ningún esquema en cuanto a que atributos deben o pueden tener, puesto que son definidos por el programador, mientras que las tablas deben seguir el esquema entidad-relación.
- **Identificador único:** Las llaves primarias de una fila tienen generalmente una forma de poder representarse como texto visible, mientras que los objetos no requieren un identificador único externamente visible.
- **Estructura vs Comportamiento**: La orientación a objetos se concentra primordialmente en asegurar que la estructura del programa es razonable (entendible, extensible, reusable, segura), mientras que los sistemas relacionales ponen el énfasis en el tipo de comportamiento que el sistema tendrá una vez en producción (eficiencia, adaptabilidad, rapidez). Los métodos de la POO asumen que el principal usuario del código orientado a objetos y sus beneficios es el desarrollador de aplicaciones, mientras que el modelo relacional enfatiza que la forma en que los usuarios finales perciben el comportamiento del sistema es mucho más importante.

Es a raíz de todos estos problemas entre estos dos modelos, que surge dentro de esta misma capa de acceso a datos, otra nueva capa, la capa la capa de abstracción de los datos, por lo que las tendencias actuales de las capas de acceso a datos es poseer dentro de ellas dos capas más, la capa de abstracción de la base de datos y la capa de abstracción de los datos.

La tendencia actual para eliminar estas diferencias entre el modelo relacional y el modelo de objetos y a su vez implementar estas dos nuevas capas dentro de la capa de acceso a datos, logrando así el aprovechamiento del gran poder de abstracción que brinda la POO, es el empleo de los llamados, frameworks de persistencia, que implementan un componente llamado mapeo objeto relacional (Object-Relational Mapping), que es una técnica de programación que permite trabajar con los datos persistidos como si ellos fueran parte de una base de datos orientada a objetos (en este caso virtual). Debido a que lo estándar es trabajar con base de datos relacionales, se deben realizar operaciones que permitan transformar un registro en objeto y viceversa, es decir lo que hace es modelar las bases de datos relacionales como si fueran orientadas a objetos ya que todos los lenguajes de programación de alto nivel son orientados a objetos permitiendo la comunicación de forma sencilla con la base de datos, con esta nueva técnica, se logra que el desarrollador se independice y se abstraiga completamente de los datos con los que esta trabajando.

Esta tendencia tiene como principales ventajas:

- Trabajar las entidades del modelo relacional como objetos.
- Encapsular las acciones como Insertar, Actualizar, Borrar *(CRUD[9])* de una entidad del modelo.
- Conseguir un código más fácil de entender y mantener.
- Eliminar de forma casi total el empleo de sentencias SQL logrando así la independencia del motor de base de datos.

Como consecuencia de los grandes beneficios que implican el empleo de frameworks, tanto frameworks arquitectónicos como frameworks de persistencia, todo proyecto de desarrollo de software tiene como tendencia actual el empleo de los mismos.

[9] Create, Retrieve, Update, Delete: operaciones realizadas sobre las entidades de una Base de Datos.

Dentro de los frameworks de persistencia más usados en la actualidad se encuentra Propel para el lenguaje de programación PHP, Hibernate para el lenguaje JAVA y Nhibernate para la plataforma .NET.

1.4 Frameworks de Persistencia.

1.4.1 Hibernate.

Hibernate es un framework de persistencia objeto-relacional y un generador de sentencias SQL. Permite diseñar objetos persistentes que podrán incluir polimorfismo, relaciones, colecciones, y un gran número de tipos de datos. De una manera muy rápida y optimizada se puede generar bases de datos en cualquiera de los entornos soportados: Oracle, DB2, MySql. Y lo más importante de todo, es código abierto, lo que supone, entre otras cosas, que no hay que pagar nada por adquirirlo. Uno de los posibles procesos de desarrollo consiste en, una vez que se posea el diseño de datos realizado, mapear este a ficheros *XML*[10] siguiendo la *DTD* [11] de mapeo de Hibernate. Desde estos se podrá generar el código de los objetos persistentes en clases Java y también crear bases de datos independientemente del entorno escogido. Hibernate se integra en cualquier tipo de aplicación justo por encima del contenedor de datos.

Ventajas

- En las consultas se puede definir los campos que se quieren levantar de un objeto, no necesariamente todos.

[10] XML: Extensible Markup Language (Lenguaje de Marcas Extensible).

[11] DTD: Document Type Definition (la definición del tipo de documento).

- Es posible declarar si se quiere obtener los objetos relacionados al que se está obteniendo o no.

- Se puede especificar tamaño o límites de los objetos a levantar.

- Se puede hacer uso del llamado cache de segundo nivel y el cache de consultas.

- Se puede declarar consultas y después usar.

- Es muy cómodo y hace más ágil el desarrollo, e incluso hay herramientas que ya generan código para Hibernate, producto como AndroMDA ya genera código Hibernate y los archivos XML de Mapeo.

Desventajas

- Usa clases generadas en tiempo de ejecución para el levantado de los objetos de la base de datos, esto genera más sobrecarga que las consultas directas en SQL.

- Si la base de datos cambia ya sea agregar un campo o lo que sea, los objetos deben ser modificados por tanto también los XML y todo lo relativo a Hibernate.

1.4.2 NHibernate.

NHibernate es uno de los frameworks más usados. Se debe principalmente al poderoso lenguaje de consulta que trae consigo: HQL (Hibernate Query Language). Está basado en el proyecto Hibernate de Java, al igual que OJB.NET es un proyecto código abierto.

Nhibernate se compone de una sección de configuración (puede ser archivo App.config o Web.config según el tipo de proyecto que sea, Windows Forms o Web) y un conjunto de mapeos Objeto-Relacionales. Utilizando estos elementos, Nhibernate se comunicará con la base de datos y realizará las acciones requeridas por los objetos persistentes (inserción, actualización, borrado, selección). El mapeo objeto-relacional lo hace mediante un archivo de mapeo, este archivo está en formato XML y su nombre por convención es clasePersistente.hbm.xml.

Existen varias formas de configurar la comunicación entre NHibernate y la base de datos, sin embargo la más recomendable es utilizar un archivo App.config (configuración de proyecto) ya que permite cambiar la configuración de acceso sin cambiar el código de la aplicación en sí.

Un archivo de configuración de aplicaciones es un archivo XML que permite configurar algunas opciones específicas de la aplicación que se desarrolla.

NHibernate permite la implementación de varios modelos de diseño para la persistencia de objetos, por tanto se puede optar por utilizar objetos DAO, patrón AbstractFactory, entre otros.

Existe la posibilidad de manejar el acceso a la base de datos en base a transacciones, es decir los cambios en la base de datos se realizan de manera atómica controlados por una transacción. En caso de suceder alguna anomalía durante una transacción abierta, se puede indicar a la aplicación que aborte la transacción y esto anulará cualquier posible cambio en la base de datos.

NHibernate es un framework que pretende abstraer por completo al desarrollador de lo que es el manejo de persistencia en las aplicaciones. Al utilizar NHibernate el desarrollador sólo trabajará con objetos capaces de almacenar y recuperar estados, con todo lo que ello significa (manejo de relaciones entre objetos, jerarquía de objetos) [11].

Algunas de las características más importantes:

- Permite el mapeo de relaciones a tipos .NET específicos.
- Permite una buena interfaz para estructurar la consulta a partir de criterios.
- Las transacciones son administradas por objetos Session.
- La configuración del mapeo y demás se administra tanto por código como por fuera de él.
- Accede tanto a propiedades privadas, públicas o protegidas.
- Las asociaciones son mapeadas a objetos IList, Collection o Dictionary.

- Asocia las clases con sus correspondientes proxys.

- Soporta actualizaciones en cascada, con posibilidad de especificar que tipo de actualización está permitido: Insert, Update y Delete (Insertar, actualizar y borrar).

1.5 Frameworks para PHP.

La mayoría de los frameworks desarrollados para el lenguaje PHP para la realización del acceso a datos tienen como principio la implementación del patrón de diseño ActiveRecord, ejemplo de ello lo son el framework Kumbia y el framework CodeIgniter.

1.5.1 Como trabaja ActiveRecord.

ActiveRecord es un patrón de diseño que enfoca el problema de acceder a los datos de una base de datos. Una fila en la tabla de la base de datos (o vista) se envuelve en una clase, de manera que se asocian filas únicas de la base de datos con objetos del lenguaje de programación usado.

Encapsula el acceso y adiciona lógica de dominio en los datos manipulados. Sirve no solamente para acceder, sino que puede garantizar cierta lógica de dominio en ese acceso, posee métodos que responden no simplemente a la lectura/escritura sino que puede garantizar transformación de dominio sobre los datos.

Es decir la implementación de este patrón constituye la principal clase para la administración y funcionamiento de los modelos, proporciona la capa objeto-relacional que sigue rigurosamente el estándar *ORM*[12]*:* Tablas en Clases, Campos en Atributos y Registros en Objetos. Facilita el entendimiento del código asociado a la base de datos y encapsula la lógica específica haciéndola más fácil de usar para el programador.

[12] Object Relational Mapping: Mapeo Objeto - Relacional.

1.5.2 Ventajas del patrón ActiveRecord.

- Se trabajan las entidades del modelo más naturalmente como objetos.

- Las acciones como Insertar, Consultar, Actualizar, Borrar de una entidad del modelo están encapsuladas así que se reduce el código y se hace más fácil de mantener.

- Código más fácil de entender y mantener.

- Reducción del uso del SQL, con lo que se logra un alto porcentaje de independencia del motor de base de datos.

- Menos detalles, más practicidad y utilidad.

La utilización de ActiveRecord permite la realización de transacciones en motores de bases de datos que lo permitan como Oracle y PostgreSQL, la utilización de funciones de autollamada (CallBacks) en determinados eventos en las tablas que lo requieran, validación de los datos antes de ser enviados a la base de datos según el tipo de datos que sean, incluso con la utilización de expresiones regulares para validar direcciones de correo electrónico.

Otras ventajas que ofrece la implementación de ActiveRecord es el uso de datos persistentes en las clases que sean necesarias, el uso de trazas, debugs y logs, mostrar errores personalizados. Y no por último menos importante permite realizar operaciones en cascada con asociaciones entre clases de una manera elegante y orientada a objetos pero solo de manera sencilla.

1.5.3 Qué es complicado en ActiveRecord.

Sin embargo la utilización del ActiveRecord como único patrón de diseño para una capa de acceso a datos se torna un poco tedioso a la hora de tener que escribir todas las clases que se derivan del modelo de la base de datos, pues en sistemas demasiados complejos la generación de clases y constructores que heredarían la clase ActiveRecord se tornaría un trabajo monótono y afectaría la cantidad de tiempo en el proceso, además de que estaría propenso a numerosos errores de construcción.

Además de que en la construcción de algunas consultas que puedan resultar complicadas, necesariamente será inevitable el empleo de utilizar código SQL en las funciones de la capa de negocio, pues esta implementación no nos garantiza una gran flexibilidad en la construcción de consultas.

- No comprende del todo los modelos entidad-relación en lo que refiere a las relaciones débiles, la generalización y la especialización. En casos como este no es capaz de construir las clases correctamente, o solo lo hace de la tabla en cuestión, lo cual trae como consecuencia la omisión de atributos relevantes.

- No es capaz de realizar las operaciones de consultas con cierta complejidad, sobre todo en lo referente a la unión de campos por la funcionalidad JOIN de lenguaje SQL.

Presenta algunas funcionalidades que aún se encuentran en desarrollo por lo que son debilidades que aún presenta:

- No permite actualizar solo los campos que han sido cambiados.

- No exige realizar una sentencia **SELECT** anterior a la actualización **UPDATE** para comprobar que los datos no hayan sido cambiados.

- No Permite crear una entidad ActiveRecord de solo lectura que mapearía los resultados de un **SELECT** directamente a un objeto.

1.6 Capas de Persistencia para PHP.

Hay prácticamente un número de capas de objetos persistentes para Java (Torque, Hibernate, Castor, OJB entre otros); sin embargo, hay pocas soluciones que alguna vez proveen un mapeo básico objeto-relacional para PHP. Como PHP está siendo utilizado de forma creciente en despliegues de larga escala objeto-relacional, las herramientas para PHP se están volviendo más prevalecientes. Aquí hay un vistazo de algunos proyectos similares que existen actualmente en varios escenarios de desarrollo para PHP.

1.6.1 Propel.

Propel es un servicio de objeto persistente y de consulta lo que significa que Propel provee un sistema para almacenar objetos en una base de datos y un sistema para búsqueda y restauración de objetos desde una base de datos. Propel permite realizar consultas complejas y manipulación de bases de datos sin escribir una sola cláusula SQL. Propel hace más fácil la escritura de aplicaciones, más fácil de desplegar, y mucho más fácil para migrar si alguna vez la situación lo amerita.

Propel puede ser descrito como un mapeado objeto - relacional, una capa DAO, o una capa objeto persistente. Propel proporciona un inteligente y comprensivo servicio de manejo de datos con un mínimo costo de realización para su aplicación en PHP. Considerado hasta ahora la mejor capa de persistencia para PHP. [1]

Principales Ventajas:

- Independiente de la Base de Datos
- Permite realizar consultas sobre la base de datos de alta complejidad sin utilizar ningún código SQL.
- Implementa los patrones de diseño Data Row Gateway, Data Table Gateway, ActiveRecord.
- Genera completamente el ORM de la base de datos a partir de un fichero de configuración XML.
- Realiza todas las operaciones sobre la base de datos de manera transparente para la lógica del negocio y de manera orientada a objetos.
- Realiza operaciones sobre tablas que se relacionan de mucho a mucho.

1.6.2 ALYOOP.

ALYOOP es una simple estructura PHP5 DAO.

Algunas diferencias respecto a Propel:

- Un ambiente simple de ejecución, manejo de objetos consulta y persistencia.
- No es usada la descripción de archivos para la autogeneración (el generador de clases crea subclases vacías como punto de partida).
- No hay criterios basados en el sistema de consulta orientado a objetos, en su lugar son usados filtros los cuales representan adiciones a la cláusula WHERE de SQL.

1.6.3 DB_DataContainer.

DB_DataContainer es un paquete flexible PEAR que provee una estructura simple DAO.

Algunas diferencias con Propel:

- DB_DataContainer no genera clases PHP, sin embargo aparentemente genera métodos de acceso.
- Es similar en diseño (entrada de datos en columna) a DB_DataObject; sin embargo esta clase tiene un API más simple y no utiliza archivos INI.
- Se integra con DB_DataContainer_Form.
- No hay criterios basados en el sistema de consulta orientado a objeto.

1.6.4 DB_DataObject.

DB_DataObject es un paquete PEAR que hace una generación básica de código y posee un soporte básico para las relaciones.

Algunas diferencias con Propel:

- Utiliza sólo el patrón de entrada de datos en columna. Como su nombre lo dice, el diseño si es mucho más simple: hay esencialmente una “gran” clase de la cual todas las tablas de filas heredan.

- Requiere que la base de datos ya exista; luego crea los archivos INI que describen la base de datos para referencias internas.

- No usa las funciones getter/setter, pero expone las propiedades de la columna como públicas.

- No hay criterios basados en el sistema de consulta, sin embargo métodos como *whereAdd()* pueden construirse manualmente en SQL y las búsquedas pueden incluso ser desarrolladas usando un método *find()* que construye SQL basado en los valores del objeto actual.

- Posee integración con el paquete HTML_Quickform de PEAR.

1.6.5 DB_Table.

DB_Table es otro paquete PEAR que provee un nivel de tabla de abstracción de la base de datos.

Algunas diferencias con Propel:

- DB_Table provee solamente un patrón de entrada de datos

1.6.6 EasyORM.

La principal desventaja y diferencia es que es una solución para MySQL únicamente.

1.6.7 Metastorage.

Metastorage es probablemente la más completa en orden de soluciones además de la ya naciente Doctrine, y más bien similar a Propel en términos del uso de XML para el modelo de descripción y consultar una fase de construcción para crear clases PHP.

Algunas diferencias con Propel:

- Metastorage usa XML para la personalización de clases de comportamiento; Propel usa clases de extensión y métodos override. La personalización de Metastorage requiere algún conocimiento metalenguaje XML para crear código.

- Metastorage utiliza Metabase como una capa de abstracción de la base de datos, y como su nombre lo dice soporta un amplio arreglo de bases de datos.

- Teóricamente Metastorage puede crear objetos persistentes para cualquier lenguaje (Java), aunque en la práctica es solo generación de clases PHP. Propel puede en teoría inclusive ser extendido para crear objetos en otros lenguajes.

- Parece correcto decir que Metastorage no tiene ambiente de ejecución; todo es almacenado en los objetos generados. Propel utiliza algunas clases centrales para manejar la conexión a las bases de datos, y almacena código común para construir consultas.

- Metastorage no provee un criterio basado en un sistema de construcción de consulta, en su lugar genera métodos que encapsulan totalmente las consultas.

1.7 Propel vs Doctrine.

Actualmente viene estableciéndose con gran fuerza**,** Doctrine, el cual es un potente y completo sistema ORM, desarrollado para usar a partir de PHP 5.2 con un DBAL (database abstraction layer) incorporado, intentando imponerse ante el ya establecido

Propel, Doctrine presenta muchas facilidades al igual que Propel, superándolo en la rapidez ya que la capa de abstracción de la base de datos que presenta es PDO, siendo más ligera que la presentada por Propel, Creole.

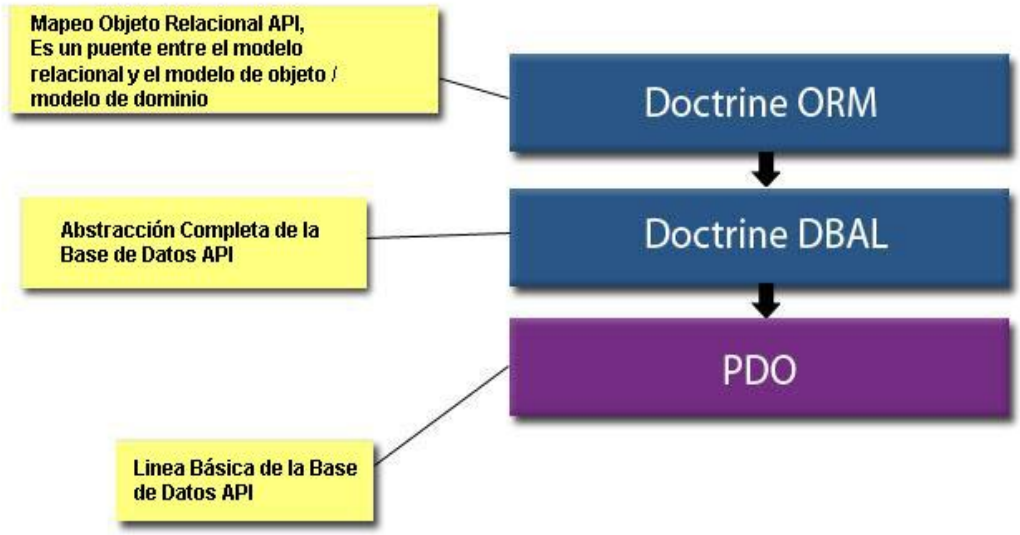

Fig.1.1 Estructura de Doctrine.

A pesar de ser visto Doctrine por muchos desarrolladores como más avanzado que Propel, Fabien Potencier el creador y máximo responsable de Symfony responde alto y claro ante cualquier duda:

- Propel está integrado a Symfony hace más de dos años. Doctrine lleva en forma de plugin más de un año.

- Mantener dos ORM diferentes cuesta mucho tiempo y esfuerzo, así que Symfony siempre va a elegir uno de los dos y ese va a ser el único que se incluya en el Framework.

- Symfony 1.1 al igual que la versión 1.0 ha elegido a Propel como su ORM por defecto.

- Doctrine no es lo suficientemente estable aún.

Dada la poca madurez del framework Doctrine, la necesidad del empleo de PHP 5.2.3+ para su buen desempeño, el poco empleo del mismo en la comunidad desarrolladora capaz de validar su aceptación en el desarrollo de aplicaciones complejas y con cierto de nivel de formalidad y seriedad hace que la balanza se incline por el empleo de Propel, un framework ya probado como aceptado y bastante estable. Siempre sin el descuido del constante estudio sobre ambas herramientas con el objetivo de emplear la más indicada según las condiciones lo permitan y de esta forma lograr la obtención de un producto de calidad.

1.8 Symfony y Propel.

Por las grandes facilidades que brinda el empleo de frameworks se decide realizar la arquitectura para el Polo Sistemas Tributarios y de Aduanas con el apoyo del framework Symfony que implementa el patrón arquitectónico MVC.

Symfony es un framework que simplifica el desarrollo de una aplicación mediante la automatización de algunos de los patrones utilizados para resolver las tareas comunes. Además proporciona una estructura al código fuente, forzando al desarrollador a crear código más legible y más fácil de mantener. Un framework facilita la programación de aplicaciones, ya que encapsula operaciones complejas en instrucciones sencillas. Symfony está completamente diseñado para optimizar, gracias a sus características, el desarrollo de las aplicaciones web. Proporciona varias herramientas y clases encaminadas a reducir el tiempo de desarrollo de una aplicación web compleja. Automatiza las tareas más comunes, permitiendo al desarrollador dedicarse por completo a los aspectos específicos de cada aplicación. Es compatible con la mayoría de gestores de bases de datos, como MySQL, PostgreSQL, Oracle y SQL Server de Microsoft. Se puede ejecutar tanto en plataformas *nix (Unix, Linux.) como en plataformas Windows.

Dentro de las principales características del framework se encuentran:

- Independiente del sistema gestor de bases de datos.

- Es lo suficientemente flexible como para adaptarse a los casos más complejos.

- Sigue la mayoría de las mejores prácticas y patrones de diseño para la web.

- Preparado para aplicaciones empresariales, y adaptable a las políticas y arquitecturas propias de cada empresa, además de ser lo suficientemente estable como para desarrollar aplicaciones a largo plazo.

- La gestión de la cache reduce el ancho de banda utilizado y la carga del servidor.

- La autenticación y la gestión de credenciales simplifican la creación de secciones restringidas y la gestión de la seguridad de usuario.

- Las interacciones con AJAX son muy fáciles de implementar mediante los *helpers* que permiten encapsular los efectos Javascript compatibles con todos los navegadores en una única línea de código.

- Las herramientas que generan automáticamente código han sido diseñadas para hacer prototipos de aplicaciones y para crear fácilmente la parte de gestión de las aplicaciones.

Symfony, continua teniendo muchas más ventajas que no se mencionan, y posiblemente la más importante sea la posibilidad que ofrece del empleo o no de frameworks de persistencia, o la utilización o no de un ORM. Usualmente hace uso del framework Propel para el desarrollo de la capa de acceso a datos ya que ambos trabajan con gran cohesión. Dada las comodidades del framework de persistencia Propel y el empleo de Symfony como framework para establecer la arquitectura se decide el empleo de Propel como apoyo para la modelación e implementación de la capa de acceso a datos para los sistemas del PSTA.

1.9 Conclusiones.

En este capítulo se ha podido apreciar como los mecanismos para el acceso a los datos se han visto involucrados en constantes cambios, cambios estos encaminados a lograr una mayor comodidad en el trabajo de los desarrolladores, logrando que ganen en tiempo, disminuyan el esfuerzo de trabajo y la obtención de un producto de mayor calidad. Es por ello que después de un análisis de los diferentes mecanismos de acceso a datos se decide el empleo de un framework de persistencia para el apoyo en la creación de un modelo de acceso a datos para los diferentes sistemas del PSTA en este caso del framework Propel.

Capítulo 2: Solución Propuesta.

2.1 Introducción.

En este capítulo se realiza una fundamentación del uso de Propel en la implementación de la capa de acceso a datos en los sistemas del PSTA. Serán expuestas además las adecuaciones necesarias para adaptar el framework a las necesidades del PSTA.

2.2 Como funciona Propel.

En el capítulo anterior se mostró Propel de forma general. Este capítulo presenta a Propel bajo un análisis más profundo.

Propel inicialmente implementa el patrón Data Row Gateway, como lo describe Martin Fowler, para representar la base de datos. Por citar a Fowler:

Una entrada de datos de fila (Data Row Gateway) le da objetos que lucen exactamente como el registro en su estructura de registros pero puede ser accedido con los mecanismos regulares de su lenguaje de programación habitual. Todos los detalles del acceso a las fuentes de datos están ocultos detrás de esta interfaz. [9]

Sin embargo, Propel también genera las clases para cada tabla que exhibe algunas de las propiedades de la tabla del patrón Table Data Gateway:

Una tabla de entrada de datos (Table Data Gateway) almacena todo el SQL para acceder a una sola tabla o vista: selecciones, inserciones, actualizaciones, y eliminaciones. Otro código llama los métodos para todas las interacciones con la base de datos.

En Propel las clases de tabla de entrada de datos (Table Data Gateway) son llamadas clasesPeer, mientras la clase de filas de entrada de datos (Data Row Gateway) es llamada entidad o clases objeto.

Propel tiene dos componentes principales:

- Un motor generador para construir sus clases y sentencias SQL.
- Un ambiente de ejecución que proporciona herramientas para construir SQL, ejecutando consultas compiladas, y herramientas para el manejo de conexiones para múltiples bases de datos.

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

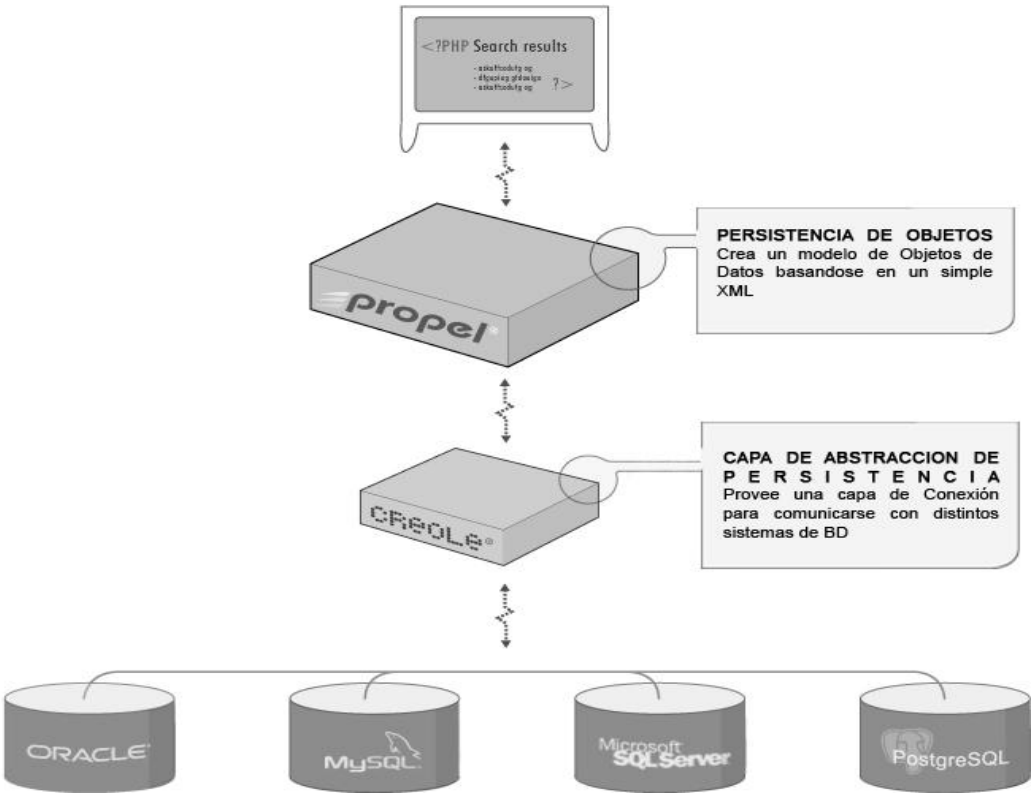

Fig.2.1 Estructura de Propel.

Para la creación de clases que sean capaces de representar las tablas de la base de datos y las relaciones entre ellas, se hace necesaria su representación en un archivo XML capaz de describir la

base de datos en cuestión. Este archivo .XML o .YML constituye lo que se le denomina esquema (schema).

Este archivo XML puede construirse de forma manual por el desarrollador para ajustarlo a sus necesidades, pero Propel ofrece la posibilidad que si ya la base de datos con la cual se trabajará ya está confeccionada, poder usar las clases metadato de Creole para que de forma automática construya el archivo XML, aunque la mayoría de las veces una vez generado este XML se deben observar algunos detalles, por ejemplo especificar autoincremento para las columnas, o personalizar los nombres en PHP para las columnas.

Una vez realizado el XML correctamente Propel genera, basado en este XML, sus clases y el SQL correspondiente.

2.2.1 Clases generadas y propósito.

Clase	Propósito
BaseTabla, Tabla	BaseTabla representa la clase base para una fila de la tabla Tabla. Tabla es la subclase vacía en donde las personalizaciones fueron adicionadas; las consultas retornarán un arreglo de objetos del tipo Tabla.
BaseTablaPeer, TablaPeer	La clase Peer es una clase que posee únicamente métodos estáticos que agiliza las consultas y manipulaciones contra la tabla Tabla. Todas las referencias deben ser para TablaPeer que es una subclase vacía (para personalización) de BaseTablaPeer.
TablaMap	Este contiene un mapa de la base de datos para la tabla Tabla. Para agilizar las bajas consultas a metadatos en tiempo de ejecución (ej. Conocer cuáles columnas son llaves primarias, llaves foráneas.), Propel compila una clase mapa que puede rápidamente retornar información relevante acerca de la estructura de la tabla.

[*2*]

Estas clases generadas a partir del esquema confeccionado en el XML permiten el trabajo del acceso a datos completamente orientado a objetos y logrando la abstracción de los datos.

2.3 Estructura del generador de Propel.

El generador de Propel propone una estructura para tener de una forma organizada y a la vez encapsulada los componentes que son generados por el framework.

```
Propel/generator
| -- classes
|     + --  propel
|             | --  engine
|             |       | --  database
|             |       |       | --  model
|             |       |       + --  transform
|             |       + -- sql
|             + -- phing
| -- dtd
| -- projects
|     + -- bookstore
| -- templates
+ -- test
       | -- classes
       |      + -- propel
       + -- etc
```

Brevemente se describe los contenidos de los mayores directorios:

Directorio	Descripción
Classes(clases)	Repositorio de todas las clases usadas por Propel. Estas incluyen las clases Creole, las cuales proveen una API ligera de bases de datos unificada.
Dtd	Almacena un simple archivo DTD para el esquema XML de validación de archivos de bases de datos.
Projects (Proyectos)	Este es un repositorio para todos los archivos de proyecto (un directorio por proyecto). Propel lee el esquema de los archivos de configuración desde este directorio y crea todas las salidas en este directorio.
Templates (Plantillas)	Hay plantillas PHP (capsula) que crea las clases PHP y archivos de descarga SQL basados en el modelo de datos.
Test(Pruebas)	PHPUnit2 casos de prueba.

2.4 Capa de abstracción de Propel.

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

La principal ventaja de la capa de abstracción de la base de datos es la portabilidad, porque hace posible el cambiar la aplicación a otra base de datos, incluso en mitad del desarrollo de un proyecto. Si se debe desarrollar rápidamente un prototipo de una aplicación y el cliente no ha decidido todavía la base de datos que mejor se ajusta a sus necesidades, se puede construir la aplicación utilizando SQLite y cuando el cliente haya tomado la decisión, cambiar fácilmente a MySQL, PostgreSQL u Oracle. Solamente es necesario cambiar una línea en un archivo de configuración y todo funciona correctamente.

Symfony utiliza Propel como ORM y Propel utiliza Creole como capa de abstracción de bases de datos. Estos dos componentes externos han sido desarrollados por el equipo de Propel, y están completamente integrados en Symfony, por lo que se pueden considerar una parte más del framework.

Creole es una capa de abstracción de la base de datos para PHP 5.* el cual se basa en la experiencia de una serie de paquetes anteriormente desarrollados para la abstracción de la base de datos en PHP, en particular PEAR::DB, PEAR::MDB, y ADOdb. Creole fue creado como un subproyecto de Propel para satisfacer necesidades específicas, que ninguna de las capas de abstracción disponibles hasta ese entonces era capaz de hacer frente de forma satisfactoria.

Creole presenta diferentes características dentro de las que se encuentran:

- Construido para PHP5: nuevo modelo de objetos, Excepciones.

- Totalmente orientado a objetos (API).

- ResultSet desplazamiento: siguiente, anterior, primero, último, en relación, y la fila absoluta de obtención.

- ResultSetIterator (SPL) proporciona un medio adicional para la iteración más de recordsets.

- Completa base de datos de metadatos (base de datos, tabla, columna completa, llaves primarias, llaves extranjeras, índices) utilizando simples API orientados a objetos.

- Completo sistema de tipo único (basado en JDBC Tipos).

- Manipulación de datos BLOB y CLOB.

- Tipo de métodos específicos para manejar cualquier necesidad para escapar de las conversiones, la inserción y recuperación de los valores.

- Amplia unidad de marco de pruebas de ensayo cada drivers utiliza bases de datos reales.

2.5 Symfony, Propel y los Plugins.

Symfony y Propel son frameworks muy flexibles, tienen una gran ventaja, la posibilidad que brindan del empleo de plugins pues en ocasiones, es necesario reutilizar una porción de código desarrollada para alguna aplicación Symfony. Si se puede encapsular ese código en una clase, tan sólo es necesario guardar la clase en algún directorio `lib/` para que otras aplicaciones puedan encontrarla. Sin embargo, si el código se encuentra desperdigado en varios archivos, como por ejemplo un tema para el generador de administraciones o una serie de archivos JavaScript y *helpers* que permiten utilizar fácilmente un efecto visual complejo, es muy complicado copiar todo este código en una clase.

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

Básicamente, un plugin es una extensión encapsulada para un proyecto. Los plugins permiten no solamente reutilizar código propio, sino que permiten aprovechar los desarrollos realizados por otros programadores y permiten añadir al núcleo de Symfony y Propel extensiones realizadas por otros desarrolladores.

Los plugins pueden jugar un papel fundamental en la seguridad de los sistemas pues tienen la utilidad de fortalecerla pues permiten la creación de componentes que permiten realizar comprobaciones de seguridad verticalmente sobre la aplicación.

Ejemplos de Plugins para Propel

- **sfPropelParanoidBehaviorPlugin**: deshabilita el borrado de los objetos y lo reemplaza por la actualización de una columna llamada deleted_at.

- **sfPropelOptimisticLockBehaviorPlugin**: implementa la estrategia *optimistic locking* para los objetos Propel.

- **sfPropelAlternativeSchemaPlugin**: amplía el modelo generador de Symfony sobre la base de Propel, de forma que será posible redefinir las propiedades de un esquema con otro esquema.

Ejemplos de Plugins para Symfony

- **sfThumbnailPlugin**: crea imágenes en miniatura, por ejemplo para las imágenes subidas por los usuarios.

- **sfMediaLibraryPlugin**: permite gestionar la subida de archivos multimedia, incluyendo una extensión para los editores avanzados de texto que permite incluir las imágenes dentro de los textos creados.

- **sfShoppingCartPlugin**: permite gestionar un carrito de compra.

2.5.1 Instalación.

El proceso de instalación de los plugins difiere del modo en que ellos vengan empaquetados. Para esto se debe leer el archivo léeme (readme) del plugin. No obstante siempre que se instale un plugin se debe limpiar la cache de la aplicación de Symfony para el correcto funcionamiento del mismo.
Los plugins pueden ser instalados a nivel de aplicación o de proyecto, los métodos de instalación descritos a continuación colocan los archivos de la instalación en el directorio `miproyecto/plugins/nombrePlugins`.
La manera más común de instalar un plugin es mediante el método PEAR. Este tiene tres formas fundamentales. Desde el sitio oficial de Symfony, desde un canal de PEAR y desde el disco local. Las tres formas utilizan el siguiente formato respectivamente:

```
> cd miproyecto

> php symfony plugin- install http://plugins.symfony-
project.com/nombrePlugins

> php symfony cc
```

```
> cd miproyecto

> php symfony plugin- install nombreCanal/nombrePlugins

> php symfony cc
```

```
> cd miproyecto

> php symfony plugin- install camino/del/plugins/nombrePlugins.tgz

> php symfony cc
```

La otra manera de realizar la instalación desde un archivo desde el disco local es de forma manual, esta requiere descomprimir el archivo del plugins en una carpeta `miproyecto/plugins/nombrePlugins`, y en caso de que el plugin contenga dentro de alguna carpeta Web, se copia para `miproyecto/web/nombrePlugins`.
Hay que tener en cuenta que a la hora de instalar un plugin hay algunas tareas que el comando `plugin-install` no realiza, por lo que hay que realizarlas a mano.

- El código de los plugins puede hacer uso de una configuración propia, pero no se pueden indicar los valores por defecto en un archivo de configuración `app.yml` dentro del directorio `config/` del plugin. Para trabajar con valores por defecto, se utilizan los segundos argumentos opcionales en las llamadas a los métodos `sfConfig::get()`. Las opciones de configuración se pueden redefinir en el nivel de la aplicación.

- Las reglas de enrutamiento propias se deben añadir manualmente en el archivo `routing.yml`.

- Los filtros propios también se deben añadir manualmente al archivo `filters.yml` de la aplicación.

- Las factorías propias se deben añadir manualmente al archivo `factories.yml` de la aplicación.

2.5.2 Activación.

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

```
all:

  .settings:

    enabled_modules:  [default, sfMiPlugin]
```

2.5.3 Desinstalación.

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

```
> cd miproyecto

> php symfony plugin-uninstall pear.symfony-project.com/sfMiPlugins

> php symfony cc
```

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

2.5.4 Estructura de un Plugin.

El directorio de un plugin se organiza de forma muy similar al directorio de un proyecto. Los archivos de un plugin se deben organizar de forma adecuada para que Symfony y Propel puedan cargarlos automáticamente cuando sea necesario.

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

```
      actions/
        actions.class.php
      config/
        module.yml
        view.yml
        security.yml
      templates/
       *.php
      validate/
        *.yml
web/
   * // Archivos estáticos
```

Fig.2.2 Estructura de Directorios de un Plugin.

Los plugins pueden contener numerosos elementos. Su contenido se tiene en consideración durante la ejecución de la aplicación y cuando se ejecutan tareas mediante la línea de comandos. Sin embargo, para que los plugins funcionen correctamente, es necesario seguir una serie de convenciones:

- Los esquemas de bases de datos los detectan las tareas `propel-`. Cuando se ejecuta la tarea `propel-build-model` para el proyecto, se reconstruye el modelo del proyecto y los modelos de todos los plugins que dispongan de un modelo. Los esquemas de los plugins siempre deben contener un atributo package que siga la notación `plugins.nombrePlugin. lib.model`, como se muestra en el ejemplo:

```
propel:

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

Fig.2.3 Atributo package en el archivo miPlugin/config/schema.yml

- La configuración del plugin se incluye en el script de inicio del plugin (`config.php`). Este archivo se ejecuta después de las configuraciones de la aplicación y del proyecto, por lo que Symfony ya se ha iniciado cuando se procesa esta configuración. Se puede utilizar este archivo por ejemplo para añadir nuevos directorios a la ruta de directorios incluidos por PHP o para extender las clases existentes con *mixins* .

- Los archivos de datos o *fixtures* del directorio `data/fixtures/` del plugin se procesan mediante la tarea `propel-load-data`.

- Las tareas del plugin están disponibles en la línea de comandos de Symfony tan pronto como se instala el plugin. Una buena práctica consiste en prefijar el nombre de la tarea con una palabra significativa, por ejemplo el nombre del plugin. Si se teclea simplemente Symfony en la línea de comandos, se puede ver la lista completa de tareas disponibles, incluyendo las tareas proporcionadas por todos los plugins instalados.

- Las clases propias se cargan automáticamente de la misma forma que las clases que se guardan en las carpetas `lib/` del proyecto.

- Cuando se realiza una llamada a `use_helper()` en las plantillas, se cargan automáticamente los *helpers* de los plugins. Estos *helpers* deben encontrarse en un subdirectorio llamado `helper/` dentro de cualquier directorio `lib/` del plugin.

- Las clases del modelo en el directorio `miplugin/lib/model/` se utilizan para especializar las clases del modelo generadas por Propel (en `miplugin/lib/model/om/` y `miplugin/lib/model/map/`). Todas estas clases también se cargan automáticamente. Las

clases del modelo generado para un plugin no se pueden redefinir en los directorios del proyecto.

- Los módulos proporcionan nuevas acciones, siempre que se declaren en la opción enabled_modules de la aplicación.

2.6 Clases en Propel.

Anteriormente se mostró de forma muy superficial algunas características de las clases que Propel genera para la manipulación de los datos, ahora se fundamentará de una forma más profunda el objetivo y función de cada una de ellas dejando bien claras las ventajas que son capaces de arrojar.
Las clases generadas por Propel se dividen en dos partes fundamentales:

- Las clases de los Objetos
- Las clases Peer

Estas son el resultado de dos patrones de acceso a datos fundamentales que Propel combina muy bien, `Tabla Data Gateway` y `Row Data Gateway`, estos se encargan de manejar las tablas y los registros de las base de datos respectivamente.

2.6.1 Clases Objetos y Peer.

Las instancias de las clases Objetos que se encuentran en los archivos `Objeto.php` y `BaseObjeto.php` representan registros de las tablas correspondientes. Se utilizan para manejar los datos de una instancia de las tablas, hacer cambios en el registro y guardarlos.
Las clases `Peer` se encuentran en los ficheros `ObjetoPeer.php` y `BaseObjetoPeer.php` y se encargan de manejar las funciones que controlan las distintas instancias de las clases Objeto. Clases que tienen métodos estáticos para trabajar con las tablas de la base de datos. Proporcionan los medios necesarios para obtener los registros de las tablas. Sus métodos devuelven normalmente un objeto o una colección de objetos de la clase `Objeto` relacionada.

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

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

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

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

2.6.2 Criteria.

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

La siguiente tabla ilustra el equivalente del código SQL como quedaría empleando la clase Criteria de Propel.

SQL	Criteria
WHERE columna = valor	->add(columna, valor);

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

De la misma forma que el lenguaje SQL es sencillo pero permite construir consultas muy complejas, el objeto de la clase Criteria permite manejar condiciones de cualquier nivel de complejidad.

2.7 Seguridad.

Una de las maneras más comunes para violar la seguridad de un sistema es haciendo uso de los llamados ataques por Inyección SQL, Propel posee un mecanismo para escapar de esta amenaza mediante el uso de la clase Criteria que se encuentra presente en el framework.

A pesar de que Propel permite el empleo de consultas SQL directamente realizadas por el desarrollador, es recomendable hacer uso de la clase Criteria pues las utilidades de esta clase permite la creación de consultas tan complejas como la situación lo requiera y todos los valores pasados son escapados para de esta forma evitar que exista algún tipo de brecha de seguridad que facilite el empleo de Inyecciones SQL como variante de ataque al sistema.

El uso de esta clase Criteria trae otra ventaja fundamental, la optimización del código SQL para la base de datos que se utiliza y la obtención de los resultados en un arreglo de objetos del tipo seleccionado.

2.8 Extender el modelo.

Las clases Propel representan como ya se ha observado la capa modelo del tradicional patrón arquitectónico MVC que implementa el framework Symfony. Los métodos del modelo que se generan automáticamente están muy bien, pero no siempre son suficientes. Si se incluye lógica de negocio propia, es necesario extender el modelo añadiendo nuevos métodos o redefiniendo algunos de los existentes. Siendo esta otra de las ventajas que brinda la flexibilidad de Propel.

2.8.1 Añadir nuevos métodos.

Los nuevos métodos se pueden añadir en las clases vacías del modelo que se generan en el directorio `lib/model/`. Se emplea $this para invocar a los métodos del objeto actual y self:: para invocar a los métodos estáticos de la clase actual. No se debe olvidar que las clases personalizadas heredan los

métodos de las clases Base del directorio `lib/model/om/`.También se pueden extender las clases *peer*.

Los nuevos métodos estarán disponibles de la misma forma que los métodos generados automáticamente.

2.8.2 Redefinir métodos existentes.

Si alguno de los métodos generados automáticamente en las clases Base no satisfacen las necesidades de la aplicación, se pueden redefinir en las clases personalizadas. Solamente es necesario mantener el mismo número de argumentos para cada método.

2.8.3 Uso de comportamientos en el modelo.

Algunas de las modificaciones que se realizan en el modelo son genéricas y por tanto se pueden reutilizar. Por ejemplo, los métodos que hacen que un objeto del modelo sea reordenable o un bloqueo de tipo *optimistic* en la base de datos para evitar conflictos cuando se guardan de forma concurrente los objetos se pueden considerar extensiones genéricas que se pueden añadir a muchas clases.

Symfony encapsula estas extensiones en *"comportamientos"* (*behaviors*). Los comportamientos son clases externas que proporcionan métodos extras a las clases del modelo. Las clases del modelo están definidas de forma que se puedan enganchar estas clases externas y Symfony extiende las clases del modelo mediante sfMixer.

Para habilitar los comportamientos en las clases del modelo, se debe modificar una opción del archivo `config/propel.ini`:

2.9 Patrones en Propel.

Los patrones de software son una forma de resolver los problemas de la ingeniería de software que ha emergido de la comunidad de la POO. Un patrón de diseño es una solución abstracta a un problema

determinado. Hoy en día existen diversas categorías de patrones a través de disciplinas técnicas y no técnicas.

Los patrones en los últimos años se han convertido en solución por excelencia, constituyéndose como una importante técnica para construir software orientado a objetos. El valor principal de estos se encuentra en el aprovechamiento de la experiencia de los expertos que los han identificado y documentado. El conocimiento de los patrones de software facilita la identificación de determinados problemas a resolver y a buscar una solución rápida y elegante; son una herramienta, la cual basándose en experiencias anteriores, resuelve ciertos problemas que son frecuentes permitiendo que los sistemas sean mucho más adaptables al cambio. Surgen en base a la arquitectura donde están presentes una serie de problemas de forma recurrente y la forma de atacarlos es similar.

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

Propel hace muy buen uso de estas técnicas pues combina muy bien algunos de estos patrones, en específico los patrones de acceso a datos dentro de los que se encuentran principalmente Data Table Gateway y Row Data Gateway, los cuales se encargan de:

Table Data Gateway:

Una tabla de entrada de datos almacena todo el SQL para acceder a una sola tabla o vista: selecciones, inserciones, actualizaciones, y eliminaciones. Otro código llama los métodos para todas las interacciones con la base de datos.

Row Data Gateway:

Una entrada de datos de fila le da objetos que lucen exactamente como el registro en su estructura de registros pero puede ser accedido con los mecanismos regulares de su lenguaje de programación habitual. Todos los detalles de acceso de fuentes de datos están ocultos detrás de esta interfaz.

Propel es considerado además como una capa DAO pues la forma de trabajo es muy fiel a la del patrón DAO pues un Data Access Object (DAO) es un componente de software que suministra una interfaz común entre la aplicación y uno o más dispositivos de almacenamiento de datos, tales como una base de datos o un archivo.

Los objetos de acceso a datos son considerados una buena práctica. La ventaja de usar objetos de acceso a datos es que cualquier objeto de negocio (el cual contiene detalles específicos de operación o aplicación) no requiere conocimiento directo del destino final de la información que manipula.

Los objetos de acceso a datos se usan generalmente para aislar a una aplicación de la tecnología de persistencia. Usando objetos de acceso de datos significa que la tecnología subyacente puede ser actualizada o cambiada sin cambiar otras partes de la aplicación.

Siendo el empleo de forma eficiente de estos patrones otro de los elementos que ponen en evidencia las ventajas del uso de Propel.

2.10 Modelación del Acceso a Datos.

El empleo de Symfony como framework arquitectónico de apoyo en el establecimiento de la arquitectura para los sistemas del PSTA, el empleo de buenas prácticas y estándares actuales en el desarrollo Web, hacen que Propel sea empleado como apoyo para el desarrollo del acceso a datos de los sistemas del PSTA.

Muchas son las ventajas que arroja el empleo de Propel, llegando a satisfacer muchas de las necesidades de los sistemas del PSTA pero existe un problema pues el generador de Propel genera las clases a partir de las tablas de la base de datos respondiendo de forma fiel, es decir, por cada tabla en la base de datos genera cinco clases, pero sucede que las clases del negocio en más de una ocasión van a depender de varias tablas de la base de datos por lo que van a depender de varias de las clases generadas para el manejo de los datos, si una tabla de la base de datos al generar las

clases respondiera fielmente a una clase del negocio fuera lo ideal pues esta clase sería la encargada del manejo de los datos de su correspondiente en el negocio pero esto solo sucede cuando la base de datos es orientada a objetos, mientras que las más usadas son las bases de datos relacionales, siendo además este tipo de base de datos las utilizadas por los sistemas del PSTA.

Para la solución de esta problemática se decide la creación de una capa intermedia entre la capa que contiene las clases que genera Propel y la capa del negocio, la cual se encargará de relacionarse con las clases generadas por el framework y de esta manera manipular los datos dejando así para las clases del negocio solo la lógica del mismo.

En la figura queda ilustrado como quedaría dicha capa:

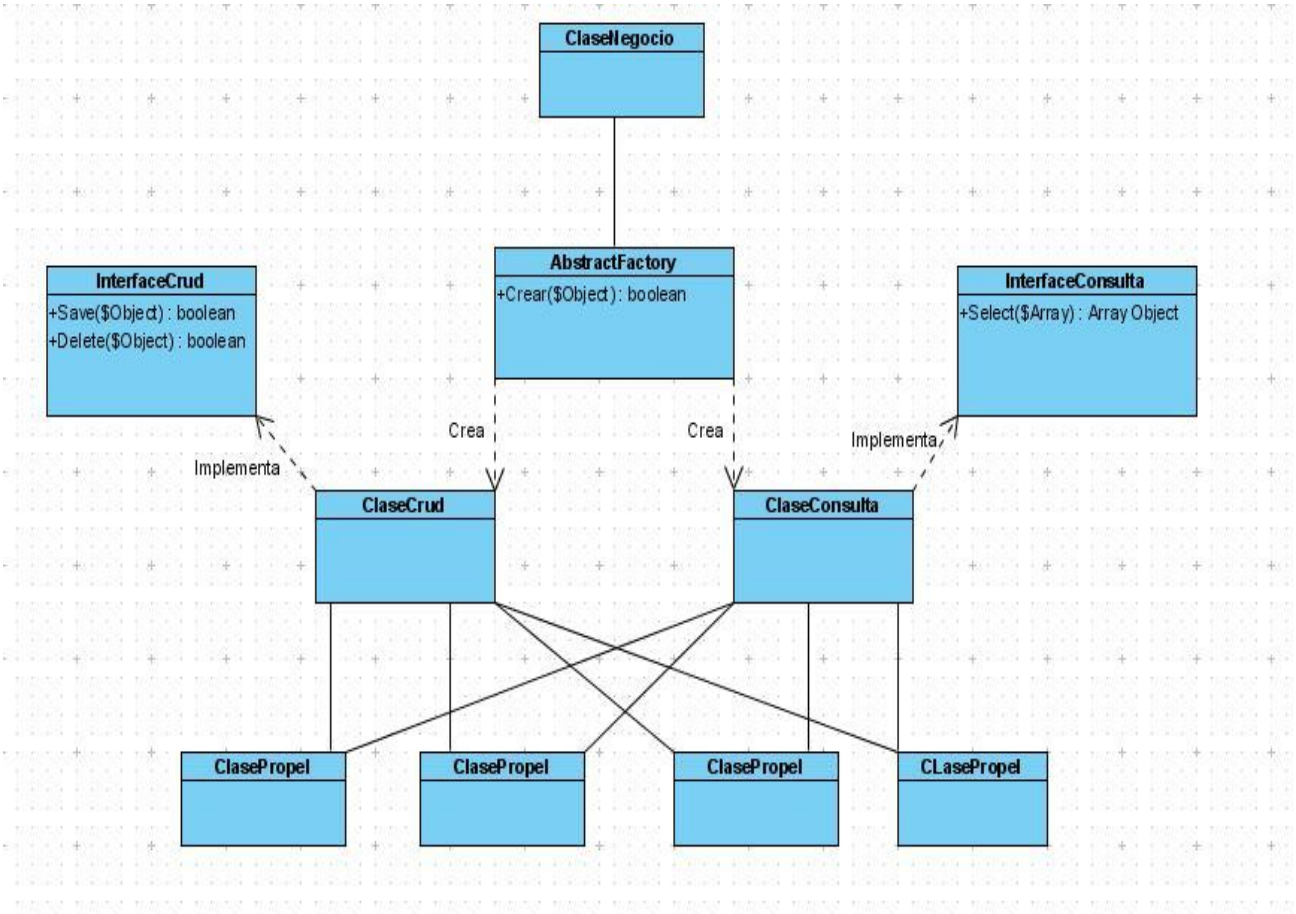

Fig.2.4 Modelo de Acceso a Datos.

2.10.1 Descripción de las clases:

***ClasePropel*:** Estas son todas las clases que genera el framework a partir del esquema de la base de datos por cada una de las tablas.

***ClaseCrud*:** Se encarga de realizar las operaciones básicas sobre la base de datos, adicionar y eliminar elementos en las tablas correspondientes siempre utilizando las clases generadas por Propel. Esta clase responde a la clase correspondiente en el negocio.

***ClaseConsulta*:** Se encarga de realizar las operaciones de búsquedas sobre la base de datos, siempre utilizando las clases generadas por Propel. Esta clase responde a la clase correspondiente en el negocio.

***InterfaceCrud*:** Esta clase contiene declarados las operaciones que van a realizar de forma común las clases **ClaseCrud**, encargándose estas últimas de implementarlos.

***InterfaceConsulta*:** Esta clase contiene declarados las operaciones que van a realizar de forma común las clases **ClaseConsulta**, encargándose estas últimas de implementarlos.

***AbstracFatory*:** Esta clase respondiendo al patrón de diseño del mismo nombre se encarga de la construcción de objetos de tipo **ClaseCrud** o **ClaseConsulta** en dependencia de la operación asignada.

***ClaseNegocio*:** Esta es la clase del negocio que se encargará de manipular toda la lógica. Su relación con las clases encargadas de hacerle sus operaciones básicas y consultas, es decir, su manipulación de los datos, será mediante la clase **AbstractFactory.**

2.10.2 Ventajas del Modelo.

La composición de esta capa intermedia permite el encapsulamiento de las operaciones sobre la base de datos en las clases especificadas para ello, permitiendo al programador del negocio trabajar sin preocuparse del acceso a los datos y enfocar su trabajo a la lógica del negocio. Siendo esta una gran ventaja, la abstracción entre las capas poniendo en marcha el buen uso de buenas prácticas de programación.

2.11 Conclusiones.

En este capítulo se pudo apreciar de forma objetiva como el framework Propel pone de manifiesto la mayoría de las buenas prácticas usadas actualmente para el desarrollo del acceso a datos en los sistemas de software. Quedó establecido además un Modelo de Acceso a Datos apoyado por el framework, pero solucionando algunas problemáticas, respondiendo de esta forma a las necesidades demandadas por la arquitectura establecida en el PSTA.

Capítulo 3: Validación de la Solución.

3.1 Introducción.

En este capítulo se podrá apreciar como la solución propuesta en la práctica es capaz de satisfacer las necesidades del PSTA y responder de forma eficiente a las mismas. Se demostrará además el nivel de integración a la arquitectura establecida en el PSTA por parte del Modelo de Acceso a Datos propuesto, reflejando así su poder de aplicabilidad de forma estándar en los sistemas del PSTA.

3.2 Estructura de las Aplicaciones del PSTA.

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

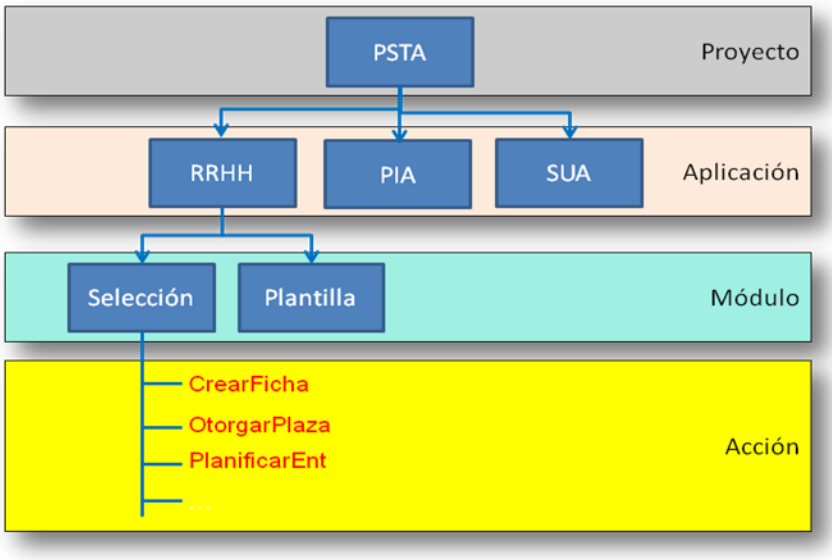

Fig.3.1 Ejemplo de la Estructura de las Aplicaciones en el PSTA.

Dentro de los proyectos que conforman el PSTA como se puede apreciar en la figura se encuentra el Proyecto Gestión Integral de Recursos Humanos (RRHH), el cual ya tiene establecido y probado el Modelo de Acceso a Datos propuesto. Proyecto este que será objeto de análisis para evidenciar la viabilidad del modelo propuesto.

3.3 Contenido del Proyecto.

Este es un proyecto que pretende proporcionar a la AGR un sistema que dé soporte a la gestión integral de los recursos humanos basándose en la gestión por competencias, llegando a definir el personal idóneo para un puesto determinado y llegar a definir estrategias de formación del personal según las carencias existentes. Para su primera versión se implementarán cuatro módulos:

Un módulo cuyo objetivo es la Selección de Candidatos que permitirá gestionar las fichas y guardar la información necesaria del candidato. Además de planificar los días del psicométrico y las fichas de los Candidatos a los que se le realizará las pruebas psicométricas en un día previamente planificado.

Un módulo para la gestión de la Estructura y Composición de la AGR que permitirá realizar las diferentes operaciones sobre la estructura de acuerdo con las reglas definidas en la AGR. Además de permitir la conversión, amortización e incrementos de plazas en algún grupo de trabajo ya creado. Además de generar todos los documentos legales del proceso.

Un módulo cuyo objetivo será el control de los Trabajadores de la AGR donde se realizará el proceso de ingreso de un trabajador a la AGR permitiendo generar el contrato de trabajo según el cargo para el que se ha procesado.

Un módulo para la gestión de las Nóminas de Pago que permitirá apoyar la actividad del pago a los trabajadores en la AGR. Llevar un registro de todas las incidencias cometidas por el trabajador durante todo el mes. Además de generar todos los documentos necesarios para el pago de los trabajadores.

Dentro de estos módulos correspondientes al proyecto se analizará de forma particular el módulo Selección al Ingreso.

3.3.1 Diagrama de Clases del Módulo Selección al Ingreso.

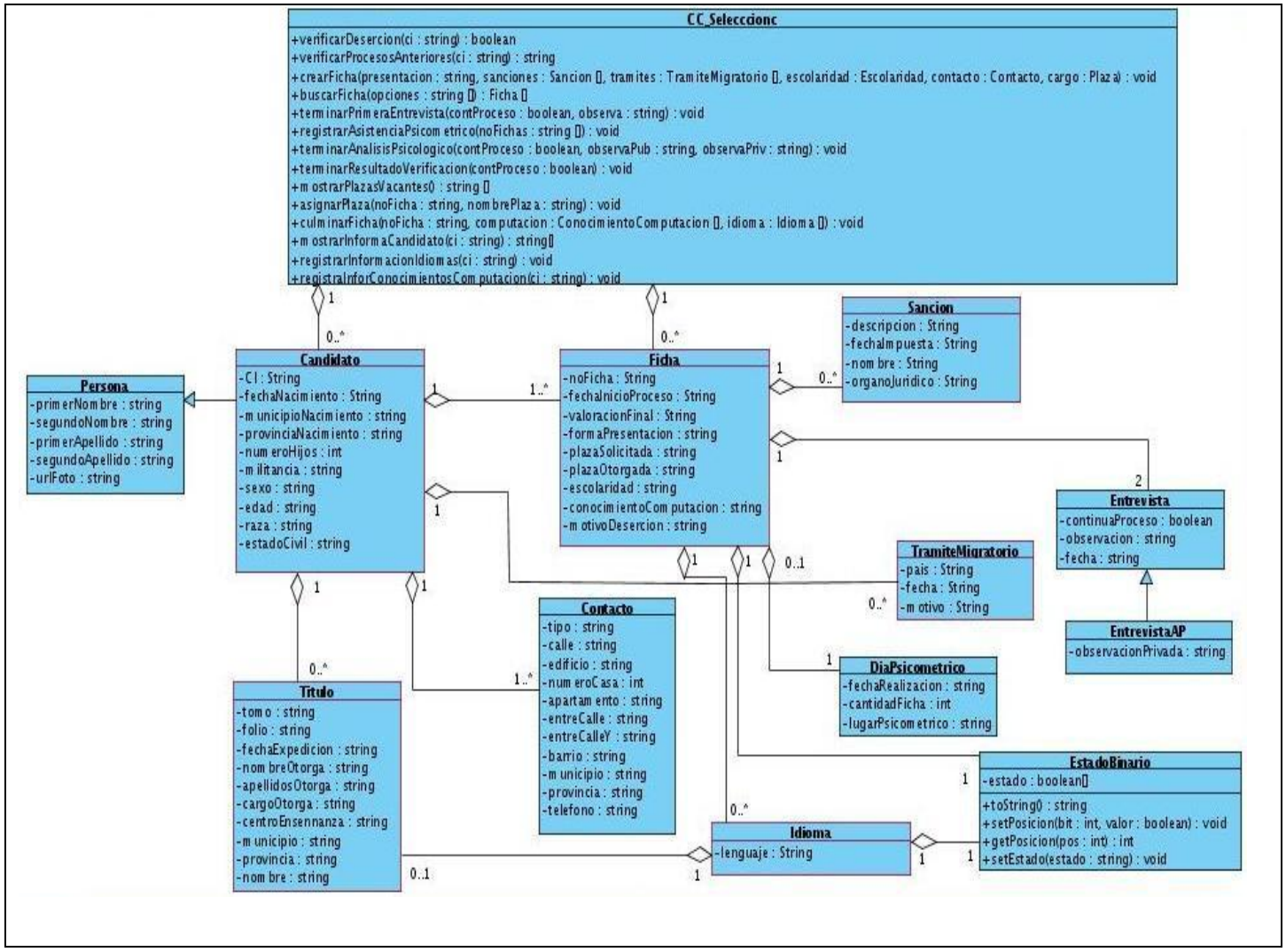

Fig.3.2 Diagrama de Clases del Módulo Selección al Ingreso.

En el diagrama se puede apreciar la relación entres las clases del negocio respondiendo a los nombres de EntrevistaAP, Entrevista, EstadoBinario, Idioma, DiaPsicometrico, TramiteMigratorio, Contacto, Titulo, Persona, Sancion ,Candidato, Ficha y finalmente CC_Seleccion siendo la Controladora de todas.

Muchas de ellas viéndose en la necesidad de la adquisición de datos pertenecientes a diferentes tablas dentro de la base de datos.

Objetos Negocio		**Entidades BD**	
		Tabla	**Campo**
Candidato	CI	rh_documento	numero
	FechaNacimiento	rh_datos_personales	fechaNac
	municipioNacimiento	tc_municipio	municipio
	provinciaNacimiento	tc_provincia	provincia
	numeroHijos	rh_datos_personales	cant_hijos
	raza	tc_raza	raza
	militancia	tc_asosiacion	asociacion
	urlFoto	rh_persona	foto
Persona	primerNombre	rh_persona	primer_nombre
	segundoNombre	rh_persona	segundo_nombre
	primerApellido	rh_persona	primer_apellido
	segundoApellido	rh_persona	segundo_apellido
	sexo	rh_datos_personales	sexo
	edad	rh_datos_personales	edad
Titulo	tomo	tc_titulo	tomo
	folio	tc_titulo	folio
	fechaExpedicion	tc_titulo	fecha
	nombreOtorga	tc_titulo	nombre_responsable
	apellidoOtorga	tc_titulo	apellido_responsable

	cargoOtorga	tc_titulo	cargo_responsable
	municipio	tc_municipio	municipio
	provincia	tc_provincia	provincia
	nombre	tc_tipo_formacion	grado
		tc_especialidad	especialidad
	centroEnsenanza	rh_formacion	nombre_institucion
Contacto	tipo	tc_localizacion	denominacion
	calle	rh_domicilio	calle
	edificio	rh_domicilio	numero
	numeroCasa	rh_domicilio	numero
	apartamento	rh_domicilio	apto
	entreCalle	rh_domicilio	entre_calle
	entreCalleY	rh_domicilio	entre_calle_y
	barrio	rh_domicilio	localidad
	municipio	tc_municipio	municipio
	provincia	tc_provincia	provincia
	telefono	rh_telefono	telefono
		rh_telefono	celular
		rh_telefono	fax
Ficha	noFicha	rh_ficha	numero
	fechaInicioProceso	rh_ficha	fecha_inicio

	valoracionFinal	rh_ficha	valoracion
	formaPresentacion	tc_forma_presentacion	forma
	plazaSolicitada	tc_cargo	nombre
	plazaOtorgada	tc_cargo	nombre
	escolaridad	tc_tipo_formacion	tipo_formacion
	conocimientoComputacion	rh_formacion	
Sancion	descripción	rh_sancion	descripcion
	fechaImpuesta	rh_sancion	fecha
	nombre	rh_sancion	motivo
	organoJuridico	tc_juridico	organo_juridico
Tramite Migratorio	país	tc_pais	país
	fecha	rh_tramite_migratorio	fecha
	motivo	rh_tramite_migratorio	motivo
Entrevista	continuarProceso	rh_ficha	estado
	observación	rh_comision	observacion
	fecha	rh_comision	fecha
EntrevistaAP	observacionPrivada	rh_comision	observacion_privada
DiaPsico-metrico	fechaRealizacion	rh_planificacion	fecha
	cantidadFicha	rh_planificacion	cantidad

	lugarPsicometrico	rh_planificacion	lugar
Idioma	lenguaje	tc_idioma	idioma

Fig.3.3 Tabla de Mapeo de Objetos.

En la tabla se puede apreciar el mapeo correspondiente de cada una de las clases del negocio con respecto a las tablas de la base de datos. Se analizará de forma especial la clase Candidato por ser una de las clases del negocio que más datos necesitados tiene diseminado en las tablas de la base de datos pues necesita datos de ocho tablas de la base de datos, es decir, apreciar como el Modelo de Acceso a Datos propuesto es funcional.

Respondiendo al modelo propuesto esta clase Candidato tiene destinada la clase CandidatoCrud y la clase CandidatoConsulta para realizar sus operaciones básicas y de consultas respectivamente sobre la base de datos, estas últimas utilizando a su vez las clases generadas por el framework. Estableciéndose la relación entre la clase Candidato y las clases CandidatoCrud y CandidatoConsulta mediante la clase AbstractFatory encargada de la creación de objetos de las dos últimas, siendo esta su única responsabilidad, además de servir como enlace entre esta capa intermedia y las clases de la capa de negocio.

```php
<?php

abstract class AbstractFactory{

    public static function Crear($clase){
        $clase.='CRUD';
        return new $clase();
    }

}

?>
```

Fig.3.4 Clase AbstracFactory.

Las clases Interface las cuales son implementadas por las clases Crud y Consultas en correspondencia a las clases del negocio.

```php
<?php

interface InterfaceCRUD{
    public function Save($Object);
    public function Delete($Object);

}

?>
```

Fig.3.5 Interface InterfaceCrud

```php
<?php

interface InterfaceConsulta{

    static public function Select($array);
}

?>
```

Fig.3.6 Interface InterfaceConsulta.

Estas clases garantizan el uso de buenas prácticas desde el punto de vista de diseño de clases al igual que las clases Crud y las clases Consultas. Se puede apreciar además (*Anexo 1*) como es que funcionan estas clases Crud y Consulta para la clase Candidato, observando además como es la relación de estas clases con las clases generadas por el framework.

Esta función Save (Anexo 1) pertenece a la clase CandidatoCrud, encargada de guardar todos los datos de la clase Candidato en las tablas correspondientes en la base de datos, demostrando de esta

forma la manipulación de las clases generadas por el framework por parte de la clase CandidatoCrud para ello. Encargándose de las consultas la clase CandidatoConsulta por ejemplo:

```
static public function obtenerCandidatoPorCI($carnet)
{
    $candidato = self::Select(array(RhDocumentoPeer::NUMERO => $carnet));
    if (count($candidato) !== 0) {
        $candidato = $candidato[0];
    }
    else{
        $candidato = array();
    }
    return $candidato;
}
```

Fig.3.7 Función consulta de la clase CandidatoConsulta.

Relacionándose la clase Candidato del negocio con las clases encargadas de su manejo de datos mediante la clase AbstractFactory como ya se ha mencionado, por ejemplo:

```
public function save(){
    $crud = AbstractFactory::Crear("Candidato");
    $crud->save($this);
}

public function delete(){
    $crud = AbstractFactory::Crear("Candidato");
    $crud->delete($this);
}
```

Fig.3.7 Funciones Adicionar y Borrar de la clase Candidato.

De esta forma en las clases Crud y Consultas de esta capa intermedia se encapsulan el manejo de los datos de las clases del negocio dejando así para estas clases solo la lógica del negocio por ejemplo:

```
private function generarEdad()
{
    $annoNacimiento = substr($this->fechaNacimiento, 0, 4);
    $mesNacimiento = substr($this->fechaNacimiento, 5, 2);
    $diaNacimiento = substr($this->fechaNacimiento, 8, 2);
    if (date('m') === $mesNacimiento) {
        if (date('d') < $diaNacimiento) {
            $this->edad = (date('Y') - 1) - $annoNacimiento;
        }
        else {
            $this->edad = date('Y') - $annoNacimiento;
        }
    }
    else if (date('m') > $mesNacimiento) {
        $this->edad = date('Y') - $annoNacimiento;
    }
    else {
        $this->edad = (date('Y') - 1) - $annoNacimiento;
    }
}
```

Fig.3.7 Función de la Lógica del negocio.

De esta forma se puede apreciar como es que funciona el mecanismo de forma real para el acceso a los datos según el modelo propuesto en los Sistemas Tributarios y de Aduanas. Teniendo como resultado en su empleo en este proyecto una buena capacidad de respuestas a las consultas solicitadas y un tiempo aceptable en su ejecución, ayudando con ello al rendimiento del sistema y a la satisfacción del cliente.

3.4 Conclusiones

En este capítulo se apreció como el modelo propuesto soluciona en la práctica los problemas que el framework no es capaz de solucionar por sí solo, sin afectar el rendimiento del sistema. Por lo que se puede decir que el mecanismo propuesto para el Acceso a Datos en el PSTA es válido y consecuente con los estándares establecidos por la arquitectura utilizada.

Conclusiones

Con la ayuda de los estudios realizados sobre los Modelos de Acceso a Datos se logró realizar la modelación de un mecanismo propio para el Acceso a Datos en el PSTA respondiendo a las necesidades de la nueva arquitectura bajo la cual se desarrollan sus sistemas, quedando de esta forma definido y validado dicho modelo, pues la implementación del mismo en el Proyecto Gestión Integral de los Recursos Humanos, fue sometida a pruebas de calidad por parte del departamento correspondiente en la Universidad, quedando de esta forma aprobado todos los requisitos no funcionales del sistema.

El trabajo permitió además conocer por estar así reflejado, cuan saludable es para el desarrollo de un software la necesidad de establecer un buen Modelo de Acceso a las fuentes de Datos.

Recomendaciones

Se recomienda el empleo del Modelo propuesto para sistemas que en su desarrollo empleen a Propel como framework de apoyo para la implementación del Acceso a Datos.

El constante estudio sobre las herramientas de Acceso a Datos como Doctrine y Propel, las más populares en la actualidad para el desarrollo en PHP, con el objetivo de mantenerse al margen de los beneficios que puedan arrojar el empleo indistintamente de cada una de ellas.

Bibliografía

- [1] Lellelid, Hans. 2004. "*Propel - Guía de usuario*".

- [2] Vesterinen Konsta. Doctrine Manual. Abril del 2008, 261p.

- [3] Gutierrez, Andrés F. 2007. *"Libro de Kumbia | Porque programar debería ser más fácil".* www.kumbiaphp.com

- [4] Ellis, Rick; Burdick, Paul. CodeIgniter User-Guide. 2008. Disponible en http://codeigniter.com/user_guide/toc.html

- [5] Wesley, Addison, Data Access Patterns**,** septiembre del 2003, 512 p.

- [6] Hernández, O. Acceso a datos de próxima generación. Disponible en: http://www.esasp.net/articulos-asp-net/1748.aspx.

- [7] Ercoli, Jorge. ¿Qué es un ORM (Object,relational,mapping)?, Octubre del 2007 Disponible en http://metodologiasdesistemas.blogspot.com/2007/10/que-es-un-orm-object-relational-pping.html

- [8] Cabrera, Martín. Introducción a NHibernate, Junio 2005 disponible en http://mcabrera.datacenter1.com/articles/dotNET/nhibernate/

- [9] Fowler, Martin. Patterns of Enterprise Application Architecture (P o EAA). 2003. 568 p.

- [10] Potencier, Fabier; Zaninotto, François. Symfony- La Guía definitiva. 2005. 588 p.

- [11] Daines, Francisco. Introducción a NHibernate. 2007. Disponible en http://fdaines.blogspot.com/2007/01/articulo-de-introduccin-nhibernate.html

- [12] Miranda P, Prudenza J, Segurola A, Calegari D, Corral J. Arquitectura de Acceso a Datos en Sistemas Orientados a Objetos: Clasificación y Descripción de Mecanismos de Persistencia. Instituto de computación, Facultad de Ingeniería, Universidad de la República, Montevideo, Uruguay, 2006.

- [13] Gallinas, A. F. G. Y. R. B. Acceso a Datos Relacionales Bajo un Entorno XML. Disponible en: http://vecpar.fe.up.pt/xata2005/papersfinal/48.pdf.

- [14] Marqués, Mercedes. Apuntes de Ficheros y Bases de Datos. Febrero del 2001.184 p. Disponible en http://www3.uji.es/%7Emmarques/f47/apun/apun.html

- [15] Jacobson, Ivar, Booch, Grady, and Rumbaugh, James. 1999. "*The Unified Software Development Process". Reading, MA: Addison-Wesley.*

- [16] Kaiser, S. H. 2005. "Software Paradigms." John Wiley & Sons, Inc.

- [17] Bass, L.; Clements, P. and Kazman, R. 2003. "*Software Architecture in Practice, 2nd Edition*". Reading, MA: Addison-Wesley.

Anexos

Anexo 1: Función Save (Guardar) perteneciente a la clase CandidatoCrud.

```
public function save($Object){

    $conexion = Propel::getConnection(RhPersonaPeer::DATABASE_NAME);

    try {

        $conexion->begin();

        $exist = false;

        $criteria = new Criteria();
        $criteria->add(RhDocumentoPeer::ID_TIPO_DOCUMENTO, 1);
        $criteria->add(RhDocumentoPeer::NUMERO, $Object->getCI());
        $personas = RhDocumentoPeer::doSelect($criteria);

        if (count($personas) != 0) {
            $exist = true;

            $persona = RhPersonaPeer::retrieveByPK($personas[0]->getIdPersona());
            $datosPersonales = RhDatosPersonalesPeer::retrieveByPK($personas[0]->getIdPersona());
            $fisico = RhFisicoPeer::retrieveByPK($personas[0]->getIdPersona());

            $criteria = new Criteria();
            $c1 = $criteria->getNewCriterion(RhAsociacionPeer::ID_PERSONA, $personas[0]->getIdPersona());
            $c2 = $criteria->getNewCriterion(RhAsociacionPeer::ID_TIPO_ASOCIACION, 1);
            $c3 = $criteria->getNewCriterion(RhAsociacionPeer::ID_TIPO_ASOCIACION, 2);
            $c2->addOr($c3);
            $c1->addAnd($c2);
            $criteria->add($c1);
```

```
        $asociaciones = RhAsociacionPeer::doSelect($criteria);
        if (count($asociaciones) != 0) {
            $asociacion=RhAsociacionPeer::retrieveByPK($asociaciones[0]->getIdAsociacion());
        }
    }
    else {
        $persona = new RhPersona();
        $datosPersonales = new RhDatosPersonales();
        $fisico = new RhFisico();
        $asociacion = new RhAsociacion();
        $documento = new RhDocumento();
        $persona->setIdPersona('0');
    }
    $persona->setPrimerNombre($Object->getPrimerNombre());
    $persona->setSegundoNombre($Object->getSegundoNombre());
    $persona->setPrimerApellido($Object->getPrimerApellido());
    $persona->setSegundoApellido($Object->getSegundoApellido());
    $persona->setFoto($Object->getUrlFoto());
    $persona->save($conexion);

    $datosPersonales->setIdPersona($persona->getIdPersona());
    $datosPersonales->setCantHijos($Object->getNumeroHijos());
    $datosPersonales->setEdad($Object->getEdad());
    $datosPersonales->setFechanac($Object->getFechaNacimiento());
    $datosPersonales->setIdMunicipio($Object->getMunicipioNacimiento());
    $datosPersonales->setIdEstadoCivil($Object->getEstadoCivil());
    $datosPersonales->setSexo($Object->getSexo());
    $datosPersonales->setIdPais(1);
    $datosPersonales->save($conexion);
```

```
            $fisico->setIdPersona($persona->getIdPersona());
            $fisico->setIdRaza($Object->getRaza());
            $fisico->save($conexion);

            if ($Object->getMilitancia() != 0) {
                $asociacion->setIdAsociacion('0');
                $asociacion->setIdTipoAsociacion($Object->getMilitancia());
                $asociacion->setIdPersona($persona->getIdPersona());
                $asociacion->save($conexion);
            }

            $contactos = $Object->getContactos();
            foreach ($contactos as $contact){
                $contact->save();
            }

            $titulos = $Object->getTitulos();
            foreach ($titulos as $titulo){
                $titulo->save();
            }

            $tramites = $Object->getTramitesMigratorios();
            foreach ($tramites as $tramite) {
                $tramite->save();
            }


             if (!$exist) {
                 $documento->setNumero($Object->getCI());
                 $criterial = new Criteria();
                 $criterial->add(TcDocumentoIdentificacionPeer::TIPO , 'CI');
                 $idTc = TcDocumentoIdentificacionPeer::doSelect($criterial);
                 $documento->setIdDocumento('0');
                 $documento->setIdTipoDocumento($idTc[0]->getIdTipoDocumento());
                 $documento->setIdPersona($persona->getIdPersona());
                 $documento->save($conexion);
             }

             $conexion->commit();

        }catch (PropelException $pe){
             $conexion->rollback();
             throw $pe;
        }
    }
```

Glosario de términos

Software: Programas o elementos lógicos que hacen funcionar un ordenador o una red, o que se ejecutan en ellos, en contraposición con los componentes físicos del ordenador o la red.

Base de Datos*:* Conjunto de datos interrelacionados, almacenados con carácter más o menos permanente en la computadora, puede ser considerada una colección de datos variables en el tiempo.

SGBD (Sistemas Gestores de Bases de Datos): Conjunto de herramientas que suministra a todos (administrador, analistas, programadores, usuarios) los medios necesarios para describir, recuperar y manipular los datos almacenados en la Base de datos, manteniendo la seguridad, integridad y confidencialidad de los mismo.

POO (Programación orientada a objeto): La Programación Orientada a Objetos es una metodología de diseño de software y un paradigma de programación que define los programas en términos de "clases de objetos", objetos que son entidades que combinan estado (es decir, datos) y comportamiento (esto es, procedimientos o métodos). La programación orientada a objetos expresa un programa como un conjunto de estos objetos, que se comunican entre ellos para realizar tareas.

Framework: Es una estructura de soporte definida en la cual otro proyecto de software, puede ser organizado y desarrollado. Típicamente, un framework puede incluir soporte de programas, bibliotecas y un lenguaje interpretado entre otros software para ayudar a desarrollar y unir los diferentes componentes de un proyecto. Representa una arquitectura de software que modela las relaciones generales de las entidades del dominio. Provee una estructura y una metodología de trabajo la cual extiende o utiliza las aplicaciones del dominio.

XML (Extensible Markup Language): es un metalenguaje extensible de etiquetas desarrollado por el World Wide Web Consortium (W3C). Es una simplificación y adaptación del SGML y permite definir la gramática de lenguajes específicos (de la misma manera que HTML es a su vez un lenguaje definido por SGML). Por lo tanto XML no es realmente un lenguaje en particular, sino una manera de definir lenguajes para diferentes necesidades.

SQL (Structured Query Language): Es un lenguaje declarativo de acceso a bases de datos relacionales que permite especificar diversos tipos de operaciones sobre las mismas. Usa características del álgebra y el cálculo relacional permitiendo lanzar consultas con el fin de recuperar información de interés de una base de datos, de una forma sencilla.

HQL (Hibernate Query Language): Es un lenguaje para el manejo de consultas a la base de datos. Este lenguaje es similar a SQL y es utilizado para obtener objetos de la base de datos. El uso de HQL nos permite usar un lenguaje intermedio que según la base de datos que usemos y el dialecto que especifiquemos será traducido al SQL dependiente de cada base de datos de forma automática y transparente.

Linux: Es un sistema operativo multi-usuario, multi-tarea que corre en muchas plataformas, incluyendo Intel. Linux tiene una buena interoperabilidad con otros sistemas operativos, incluyendo los de Apple, Microsoft y Novell. Soporta una variada gama de software, incluyendo el sistema de ventanas X y redes TCP/IP.

PEAR (PHP Extensión and Application Repository): Es un entorno de desarrollo y sistema de distribución para componentes de código PHP .Consiste en una lista bastante grande de bibliotecas de código PHP que permiten hacer ciertas tareas de manera más rápida y eficiente reutilizando código escrito previamente por otras personas. Generalmente las bibliotecas contienen clases en archivos PHP que luego se incluyen y usan sin muchas complicaciones.

API: Es una interfaz de programación de aplicaciones. Es un conjunto de especificaciones de comunicación entre componentes software. Se trata del conjunto de llamadas al sistema que ofrecen acceso a los servicios del sistema desde los procesos y representa un método para conseguir abstracción en la programación. Uno de sus principales propósitos consiste en proporcionar un conjunto de funciones de uso general. De esta forma, los programadores se benefician de las ventajas de la API haciendo uso de su funcionalidad, evitándose el trabajo de programar todo desde el principio.

DTD (Document Type Definition): La definición de tipo de documento (DTD) es una descripción de estructura y sintaxis de un documento XML o SGML. Su función básica es la descripción del formato de datos, para usar un formato común y mantener la consistencia entre todos los documentos que utilicen la misma DTD. De esta forma, dichos documentos, pueden ser validados, conocen la

estructura de los elementos y la descripción de los datos que trae consigo cada documento, y pueden además compartir la misma descripción y forma de validación dentro de un grupo de trabajo que usa el mismo tipo de información.

MVC (Modelo Vista Controlador): Es un patrón de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos. El patrón MVC se ve frecuentemente en aplicaciones web

AJAX (Asynchronous JavaScript And XML): Es una técnica de desarrollo web para crear aplicaciones interactivas .Éstas se ejecutan en el cliente, es decir, en el navegador de los usuarios y mantiene comunicación asíncrona con el servidor en segundo plano. De esta forma es posible realizar cambios sobre la misma página sin necesidad de recargarla. Esto significa aumentar la interactividad, velocidad y usabilidad en la misma.

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

Driver: Es un programa informático que permite al sistema operativo interactuar con un periférico, haciendo una abstracción del hardware y proporcionando una interfaz, posiblemente estandarizada, para usarlo. Se puede esquematizar como un manual de instrucciones que le indica cómo debe controlar y comunicarse con un dispositivo en particular. Por tanto, es una pieza esencial, sin la cual no se podría usar el hardware.

PHP: Es un lenguaje de programación interpretado, diseñado originalmente para la creación de páginas web dinámicas. Es usado principalmente en interpretación del lado del servidor (server-side scripting) pero actualmente puede ser utilizado desde una interfaz de línea de comandos o en la creación de otros tipos de programas incluyendo aplicaciones con interfaz gráfica.

YAML: Es un formato de serialización de datos inspirado en el lenguaje XML. Su propósito está centrado en los datos en lugar del marcado de documentos. Sin embargo, dado que se usa frecuentemente XML para serializar datos y XML es un auténtico lenguaje de marcado de documentos, es razonable considerar YAML como un lenguaje de marcado ligero.

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

Log: Herramienta que permite el monitoreo del comportamiento de la aplicación bajo distintas condiciones, que durante el proceso de desarrollo y prueba es muy difícil de simular.

Mixin: Es un grupo de métodos o funciones que se juntan en una clase para que otras clases hereden de ella.

Fixture: Conjunto de objetos preparados que se utilizan como punto de partida para las realizaciones de las pruebas.