UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

FACULTAD 6

Título: Interfaz de programación de aplicaciones para la transformación de esquemas de modelos de datos a Notación de Objetos JavaScript.

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

Autor: Sergio Jorge Hera.

Tutores: Msc. Asnay Guirola González.
Ing. Yanet Parra Infante.

La Habana, Junio del 2011.

DECLARACIÓN DE AUTORÍA

Declaro ser el autor del presente trabajo de diploma y reconozco a la Universidad de las Ciencias Informáticas (UCI) los derechos patrimoniales de la misma, con carácter exclusivo.

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

Sergio Jorge Hera

_____

Firma del autor

Ing. Yanet Parra Infante

_____

Firma de la tutora

Msc. Asnay Guirola González

_____

Firma del tutor

DATOS DE CONTACTO

Msc. Asnay Guirola González:
Graduado de Ingeniero en Ciencias Informáticas, se desempeña como profesor de Programación desde hace varios años, posee la categoría docente de instructor y el grado científico de Máster en Informática Aplicada. El mismo ha estado vinculado a varios proyectos productivos, en los cuales ha obtenido buenos resultados, además de participar en varios eventos nacionales e internacionales.

Email: aguirola@uci.cu

Ing. Yanet Parra Infante:
Graduada de Ingeniero en Ciencias Informáticas, se desempeña como profesora de Ingeniería y Gestión de Software desde hace tres años, tiene la categoría docente de Instructora. La misma ha estado vinculada a varios proyectos productivos, en los cuales ha obtenido buenos resultados.

Email: yinfante@uci.cu

AGRADECIMIENTOS

A los profesores que he tenido durante toda la carrera, a quienes admiro y respeto mucho.

A mis tutores Yanet y Asnay, por su paciencia, por su apoyo incondicional y por demostrarme en todo momento que yo podía lograrlo.

A Arieskien e Ismaray, por su apoyo en la realización de este trabajo y por ser excelentes amigos.

A mis amigos, por ser especiales para mí, por haber compartido tantas experiencias juntos y estar a mi lado en los momentos más difíciles.

A mi familia, que siempre ha estado pendiente de mi vida y mis estudios.

A mi novia Ana Niuska, por ser lo mejor que me ha pasado en este último año, por su inmenso amor, por estar a mi lado en los buenos y malos momentos, y por hacerme muy feliz.

A mis suegros, por su preocupación y por acogerme como un hijo más.

A todas las personas que de una forma u otra colaboraron con la realización de este trabajo.

DEDICATORIA

A mis padres Ileana y Ricardo, por ser lo más grande que tengo, por dedicarme su vida entera, por su amor, y sacrificio en cada instante de mi vida.

A mi hermano Ricardo, por su cariño, por haber sido siempre mi ejemplo a seguir y por su ayuda incondicional durante toda la carrera.

A mis abuelitas, por quererme tanto y enseñarme desde pequeño las cosas más importantes de la vida.

RESUMEN

Una Interfaz de Programación de Aplicaciones (acrónimo en inglés API) es el conjunto de funciones y procedimientos en la programación orientada a objetos que ofrece cierta biblioteca para ser utilizada por otro software como una capa de abstracción. Representa una interfaz de comunicación entre componentes de software que proporciona un conjunto de ventajas utilizadas por los programadores, los cuales hacen uso de sus funcionalidades, evitándose el trabajo de programar todo desde el principio.

El presente trabajo de diploma describe la realización de una Interfaz de Programación de Aplicaciones para la transformación de la descripción del modelo de datos asociado a Doctrine en notación de objetos JavaScript y viceversa. Se analizaron las diferentes herramientas que posibilitan el trabajo con modelos de datos en *Hypertext Pre-Processor* (PHP), centrando la investigación en la estructura del esquema del modelo de datos asociado a Doctrine en formato YAML utilizado para la generación de base de datos. Además, se realizó el análisis, diseño, implementación y prueba de la Interfaz de Programación de Aplicaciones.

El resultado fundamental se centra en la obtención de una Interfaz de Programación de Aplicaciones que permita transformar el esquema de un modelo de datos en formato JSON a formato YAML y viceversa, teniendo como valor agregado la transformación tanto de YAML o JSON a formato SQL.

PALABRAS CLAVE: API, JSON, SQL, YAML.

TABLA DE CONTENIDOS

ÍNDICE DE FIGURA

INTRODUCCIÓN

En el mundo existe un gran avance en las Tecnologías de la Informática y las Comunicaciones (TIC), dándole paso a la llamada *Era de la Informatización*. La inmersión de estas tecnologías para reducir los obstáculos tradicionales como el tiempo y la distancia, han posibilitado su uso en beneficio de millones de personas en todo el mundo, en diferentes esferas de la sociedad, lo cual ha propiciado, sin dudas, una revolución tecnológica. (1)

La información que se genera a cada segundo de forma instantánea, en todas las organizaciones, constituye la piedra angular de las mismas. Esto ha dado lugar a la necesidad de crear mejores aplicaciones informáticas, capaces de gestionar grandes cantidades de datos sin que se vea afectado su funcionamiento. El desempeño de estas aplicaciones requiere del uso de las Bases de Datos que permitan procesar la información necesaria.

Los desarrolladores de aplicaciones facilitan la gestión de la información utilizando la técnica de mapeo de objetos relacionales (ORM) que realiza el mapeo de los objetos de una base de datos relacional en clases de algún lenguaje orientado a objetos. Esto se realiza con el objetivo de ser usado directamente en código y evitar la interacción directa con las sentencias SQL, de esta forma se logra un ahorro significativo de tiempo y esfuerzo.

Cuba ha logrado alcanzar un gran avance en esta esfera de la informática, ahorrándole al país grandes sumas de dinero en cuestiones de compra de diferentes softwares, por lo que la mayoría de las instituciones cubanas se encuentran actualmente inmersas en un proceso de informatización de la sociedad. Ejemplo de ello es la creación en el año 2002 de la Universidad de las Ciencias Informáticas (UCI). Este centro universitario tiene entre sus objetivos informatizar el país y ampliar la industria cubana del software y su propósito es contribuir a la formación de los estudiantes desde la producción, lo que ha permitido que sean partícipes en el desarrollo de softwares, tanto para organismos cubanos como para otros países. Dicha universidad cuenta con una infraestructura productiva formada por varios centros de desarrollo, los cuales están asociados a las diferentes facultades. El Centro de Tecnologías de Gestión de Datos (DATEC) se especializa en el desarrollo de sistemas informáticos encaminados a satisfacer las necesidades de la gestión de la información.

Actualmente, Integración de Soluciones es uno de los cuatro departamentos que pertenecen a DATEC, en el mismo un equipo de trabajo se encuentra trabajando en la construcción de un Entorno de Desarrollo Integrado (IDE) llamado Lycan. Un IDE es una aplicación informática que consta de un

editor de código, un compilador, un depurador y un constructor de interfaz gráfica. Lycan surge por la necesidad de disponer de una herramienta para el desarrollo de componentes de presentación con el marco de trabajo ExtJS y actualmente cuenta con componentes tales como:

- **Área de Diseño:** Componente que se encarga de brindarle las funcionalidades necesarias para que el desarrollador pueda dibujar, modificar y seleccionar los componentes que se encuentran en la zona de diseño de forma visual.
- **Navegador de Objetos:** Muestra una vista en forma arbólica de los componentes que se encuentran en el área de diseño. Brinda las opciones de editar el componente a través de un menú contextual.
- **Editor de Propiedades:** Edita las propiedades y las opciones de configuración del componente seleccionado en el área de diseño.
- **Editor de Eventos:** A partir de los eventos del componente seleccionado, se pueden programar los eventos mismos.
- **Paleta de Componentes:** Muestra, clasificados por categorías, todos los componentes que contiene la plataforma e indica al IDE cuál fue el componente que se lanzó hacia el área de diseño.
- **Barra de Herramientas:** Brinda las opciones de crear, abrir, guardar, deshacer-rehacer.

En la línea Integración de Soluciones también se encuentran desarrollando un Diseñador de Modelo Entidad-Relación (MER) para incorporarlo al IDE Lycan. Este diseñador debe generar como salida la descripción de las entidades, sus atributos y relaciones, en un fichero en formato Notación de Objetos JavaScript (JSON) debido a que Lycan desarrolla los componentes de presentación con el marco de trabajo ExtJS[1], el cual utiliza el formato JSON para el intercambio de datos, por lo que es necesario crear un mecanismo capaz de generar la base de datos y el modelo de datos correspondiente al diseño realizado, así como también transformar la descripción que brinda como salida el diseñador a una descripción en formato YAML, ya que el mismo va a estar implementado sobre un Mapeador de Objetos Relacionales (ORM) que no es capaz de realizar sus funciones con una descripción en formato JSON. Por lo anteriormente planteado, surge como **problema a resolver:** ¿Cómo transformar el contenido del modelo de datos asociado a Doctrine en notación de objetos JavaScript y viceversa?

[1]Marco de trabajo de presentación que provee una biblioteca de JavaScript para el desarrollo de aplicaciones web.

Como **objeto de estudio se define:** Los Mapeadores de Objetos Relacionales existentes para el lenguaje php, enmarcado en el **campo de acción*:*** Proceso de transformación del modelo de datos asociado a doctrine en notación de objetos JavaScript.

Para darle solución al problema se plantea el siguiente **objetivo general:** Desarrollar una interfaz de programación de aplicaciones que permita la transformación de la descripción del modelo de datos asociado a Doctrine en notación de objetos JavaScript y viceversa.

Desglosándose en los siguientes **objetivos específicos:**

- Realizar un estudio del estado del arte referente a los Mapeadores de Objetos Relacionales existentes para el lenguaje php.
- Definir las tecnologías y herramientas necesarias para la implementación de la solución propuesta.
- Realizar análisis y diseño de la solución propuesta.
- Implementar la Interfaz de Programación de Aplicaciones.
- Evaluar la implementación mediante las pruebas a Nivel de Desarrollador.

Tareas de la Investigación:

- Realización de un estudio del estado del arte sobre las herramientas existentes que posibilitan el trabajo con modelos de datos.
- Realización de un análisis de la estructura del esquema del modelo de datos asociado a Doctrine en formato YAML, con el propósito de validar la estructura de los ficheros a transformar.
- Definición de las herramientas y tecnologías a utilizar en la solución.
- Realización del análisis con el objetivo de documentar el modelo de casos de uso del sistema.
- Identificación del diseño de clases de la Interfaz de Programación de Aplicaciones.
- Implementación de los componentes de la Interfaz de Programación de Aplicaciones.
- Evaluación de los resultados mediante la documentación de las pruebas.

Métodos científicos de la investigación

Analítico – Sintético: Se utiliza para analizar la teoría y los documentos relacionados con el estado del arte referente a los Mapeadores de Objetos Relacionales existentes para el lenguaje php.

Observación: Se utiliza para el análisis de la estructura del esquema del modelo de datos asociado a Doctrine en formato YAML**.**

Modelación: Permite la creación de los modelos que pertenecen al análisis, diseño e implementación de la Interfaz de Programación de Aplicaciones.

La presente investigación estará estructurada en tres capítulos:

Capítulo 1: Fundamentación Teórica.
Se presenta la definición del marco teórico de la investigación, realizando un estudio del estado del arte de las herramientas que posibilitan el trabajo con modelos de datos en PHP, centrando el análisis en la estructura del esquema del modelo de datos asociado a Doctrine en formato YAML; el cual es utilizado para la generación de base de datos. Además, se presenta la metodología, herramientas y tecnologías que serán utilizadas para el desarrollo de la solución.

Capítulo 2: Análisis y Diseño de la solución propuesta.
En este capítulo se realiza la descripción de la solución propuesta, se presentarán las estructuras que tendrán las descripciones del modelo, tanto en formato JSON como en YAML. Además, de la realización del modelo de dominio, definición y especificación de requisitos (funcionales y no funcionales), diagrama de casos de uso, diagrama de clases del diseño, diagramas de secuencia y descripción de estilos arquitectónicos y patrones de diseño utilizados.

Capítulo 3: Implementación y prueba.
En este capítulo se realiza la implementación de la Interfaz de Programación de Aplicaciones, Diagramas de componentes y la evaluación de la solución mediante las pruebas a nivel de desarrollador.

CAPÍTULO I. FUNDAMENTACIÓN TEÓRICA

En este capítulo se realiza un estudio del estado del arte referente a las herramientas que posibilitan el trabajo con modelos de datos en lenguaje PHP, centrando el análisis en la estructura del esquema del modelo de datos asociado a Doctrine en formato YAML, así como también de los elementos relacionados con la Interfaz de Programación de Aplicaciones a desarrollar. Se definen las herramientas, lenguajes y tecnologías que conformarán el ambiente de desarrollo a utilizar para la solución.

1.1 Conceptos relacionados con la investigación

Modelo de datos

Permite describir las estructuras de la base de datos, las restricciones de integridad (condiciones que deben ser cumplidas por los datos) y las operaciones de manipulación de los datos. De manera más genérica se puede expresar que permite describir aquellos elementos que interactúan en una realidad o en un problema dado y la forma en que se relacionan dichos elementos entre sí. Con frecuencia, los modelos de datos presentan dos sub-lenguajes:

- Lenguaje de Definición de Datos o *Data Definition Language* (por sus siglas en inglés DDL): su función fundamental es la de describir, de manera abstracta, aquellas estructuras de datos y sus restricciones de integridad.
- Lenguaje de Manipulación de Datos o *Data Manipulation Language* (por sus siglas en inglés DML): se encuentra orientado a describir las operaciones de manipulación de los datos. Este presenta una sección centrada en la recuperación de datos, comúnmente conocida como Lenguaje de Consulta o *Query Language*.

Los modelos de datos pueden estar separados en tres grupos: los lógicos basados en objetos, los lógicos basados en registros y los físicos de datos. Los lógicos basados en registros describen los datos en los niveles conceptual y físico, utilizando registros e instancias para representar la realidad, así como también las relaciones que puedan tener los registros o apuntadores. Los tres modelos de datos más aceptados son: Relacional, de Red y Jerárquico. Los físicos de datos, por su parte, se encargan de la representación de los datos en el nivel más bajo, básicamente captan aspectos de la implementación de los sistemas de base de datos. El Modelo unificador y la Memoria de elementos, son las dos clasificaciones existentes de este tipo.

Específicamente, los modelos lógicos basados en objetos son utilizados para realizar la descripción de datos en los niveles conceptuales y de visión. Mediante su utilización se representan los datos de la forma más cercana a la realidad, presentando gran capacidad de estructuración flexible que permite la especificación de restricciones de datos de manera explícita. Dentro de esta clasificación existen diferentes modelos, pero el más utilizado por su sencillez y eficiencia es Entidad-Relación. (2)

Mapeo relacional de objetos (ORM) (3)

Los ORM permiten mapear los objetos de una base de datos relacional en clases de algún lenguaje orientado a objetos, para ser usados directamente en código y evitar la interacción directa con las sentencias SQL. La utilización de estos permite:

- Abstracción de la base de datos: brinda la posibilidad, en un futuro, de cambiar el Sistema Gestor de Base de Datos (SGBD) con el que se está trabajando por otro, dando la seguridad de que este cambio no afectará en gran medida al sistema en desarrollo, siendo el cambio más simple.
- Reutilización: permite la utilización de los métodos propios de un objeto de datos desde distintas zonas de la aplicación, incluso, desde aplicaciones distintas.
- Mantenimiento del código: facilita el mantenimiento del código, debido al correcto orden de la capa de datos, haciendo que el mantenimiento del código sea mucho más sencillo.
- Lenguaje propio para realizar las consultas: incorporan su propio lenguaje para realizar las consultas, lo que hace que las sentencias SQL se vuelvan casi obsoletas para el usuario y comience a emplear el lenguaje propio de cada ORM.

La mayoría de los lenguajes de programación modernos son orientados a objetos, estos utilizan ORMs para mapear los objetos de una base de datos relacional en clases, algunos ejemplos de estos lenguajes son los siguientes (Ver Tabla # 1):

Lenguaje	ORMs asociados
Java	Hibernate, ORMlite, QuickDB ORM, JPMapper.
Ruby	Datamapper, ActiveRecord, Sequel.
PHP	Syrius, Doctrine, Sphorm, Propel.
C#	Nhibernate, ObjectMapper.NET, OpenAccess ORM.

Tabla # 1: Ejemplos de ORMs asociados a lenguajes de programación

Notación de objetos de JavaScript (JSON)

La Notación de objetos de JavaScript es un formato ligero de intercambio de datos. Leerlo y escribirlo es simple para humanos, mientras que para las máquinas es simple interpretarlo y generarlo. Está basado en un subconjunto del lenguaje de programación JavaScript. Este es un formato de texto completamente independiente del lenguaje, pero utiliza convenciones que son ampliamente conocidos por los programadores de la familia de lenguajes C, incluyendo C, C++, C#, Java, JavaScript, Perl, Python. Estas propiedades hacen que sea un lenguaje muy empleado y factible para el intercambio de datos. Una de sus principales ventajas es que representa mejor la estructura de los datos y requiere menos codificación y procesamiento. (4)

YAML

El formato YAML por sus siglas en inglés (*YAML Ain't Another Markup Language*) y en castellano *YAML* no es otro lenguaje de marcado, permite especificar estructuras de forma sencilla y con menos caracteres que en formato XML[2] (ver figura # 1). YAML es usado en frameworks de PHP, tales como Symfony, dada su facilidad de uso que puede resultar su edición y lectura para los seres humanos. Para la utilización de YAML no es necesario el uso de etiquetas de apertura y cierre, siendo más rápido de escribir y de leer que los ficheros de formato XML. Este es utilizado en varios lenguajes como JavaScript, Perl, PHP, Python, Ruby, Java, C#, C/C++, OCaml y Haskell.

```yaml
## YAML Template.
default language: es
other languages: [en, it]
default controller:
   web: home
   admin: login
controller alias:
   es:
     acerca: about
     contacto: contact
     busqueda: search
```

Figura # 1: Ejemplo de código YAML

La estructura de este lenguaje puede ser observada a través de la sangría, la secuencia de elementos se denota por un guión, y los pares clave/valor están separados por dos puntos. (5)

[2] Lenguaje de Marcado Extensible que se basa en un conjunto de reglas para definir etiquetas semánticas que organizan un documento en diferentes partes.

1.2 Antecedentes de la investigación

En la actualidad, la mayoría de las bases de datos siguen una estructura relacional, básicamente esto se traduce en un conjunto de tablas relacionadas, compuestas por valores escalares. Sin embargo, la Programación Orientada a Objetos (POO), muy difundida en los días de hoy, resultó en su desarrollo, posterior al surgimiento de la estructura relacional, lo que conlleva a que no sea factible el manejo de la estructura de esas bases de datos siguiendo una lógica de negocio orientada a objetos y sea necesario convertir este conjunto de tablas y relaciones en clases y objetos válidos para la programación. Como solución a dicha problemática surgen los ORM. A continuación se presentan algunos de los ORM existentes para el lenguaje PHP.

Syrius
Proyecto desarrollado en PHP 5.2.3 que añade potencia y flexibilidad para aplicaciones web mediante la introducción de las funciones de un ORM. Puede gestionar relaciones con objetos mediante la conexión de la base de datos a través de una capa de abstracción portátil DBAL (siglas en inglés de Capa de Abstracción de Bases de Datos) y sin ningún tipo de generación de código. Este puede ser visto como una implementación que permite introducir cambios adicionales en términos de patrones de diseño para mejorar la gestión de los objetos. (6)

Sirio se encuentra en su estado inicial de desarrollo, el mismo es usado por algunos sitios Web en la actualidad con el objetivo de hacerle una depuración intensiva. Su primera versión fue en marzo 2011, de ahí que se cuente con poca documentación del mismo y no sea muy empleado aun en el desarrollo de aplicaciones Web.

Generador de Objetos PHP (POG)
Generador de código PHP de código abierto que genera automáticamente el código limpio, probado y orientado a objetos para una aplicación PHP4/PHP5, mediante la generación de objetos de PHP con los métodos Crear, Obtener, Actualizar y Borrar integrados. Dentro de sus principales ventajas se aprecia que es rápido y escalable, genera código limpio, de prueba, y ficheros de instalación. Es extensible a través de plugins, compatible con PHP4 y PHP5, y con Objetos de Datos de PHP (DOP). Este es gratis para uso personal y comercial. Aunque no es muy empleado por la comunidad de desarrollo de aplicaciones Web, debido principalmente a que existe poca bibliografía del mismo y no se encuentra integrado a los marcos de trabajos más empleados. (7)

ADOdb Active Record
Implementado utilizando PHP y diseñado en los principios del modelo de ActiveRecord, diseño que fue descrito por primera vez por Martin Fowler. En el mismo, las tablas y las filas de la base de datos se resumieron en objetos nativos de PHP, permitiéndole al programador centrarse más en la manipulación de los datos y menos en la escritura de consultas SQL. Este es de código abierto, funciona con PHP4 y PHP5, proporcionando una funcionalidad equivalente en ambas versiones de PHP. Presenta una documentación pobre y no se encuentra integrado a los marcos de trabajos más empleados en la actualidad. (9)

Php-ActiveRecord
Librería de código abierto, cuyo objetivo es simplificar las interacciones con la base de datos y eliminar la tarea de SQL escrita a mano para operaciones comunes. No es necesario usar generadores de código ni mantener ficheros de asignación para las tablas. Esta colección está inspirada en la implementación de Ruby on Rails, por lo tanto toma muchas de sus convenciones e ideas. Solamente trabaja bien cuando el esquema coincide con la estructura del objeto. Es algo isomórfico y el acoplamiento a la base de datos es bastante alto. Esto puede ser una desventaja y puede inducir problemas en el futuro a la hora de refactorizar.(10)

Propel
Realizado bajo código abierto para PHP5, facilita la labor de desarrollo de aplicaciones web, gracias a la capa que transforma el tratamiento de la base de datos mediante objetos, proporcionando una Interfaz de Programación de Aplicaciones sencilla para almacenar, modificar y recuperar datos. Ofrece al desarrollador web las herramientas para gestionar bases de datos, de la misma manera que se trabaja con otras clases y objetos en PHP. Propel genera código para eliminar la carga de la introspección en tiempo de ejecución, ganando en rapidez, dando la opción, además, de no interferir en la realización de consultas personalizadas. Provee un soporte básico para implementar herencia orientada a objetos (subclases). Hay varias opciones de implementación para mapear clases entidades y subclases para las tablas de las bases de datos. Es muy empleado en la actualidad en el desarrollo de aplicaciones Web y se encuentra integrado a varios marcos de trabajo para PHP. (12)

Doctrine
Librería para el trabajo con PHP que permite interactuar con un esquema de base de datos como si se tratase de un conjunto de objetos, y no de tablas y registros. Actualmente goza de una amplia comunidad de desarrollo y documentación disponible superior y más eficiente que la de muchos otros frameworks de persistencia, que ha ido en aumento en los últimos tiempos y lo convierte en uno de los

más potentes para PHP. Es muy empleado en la actualidad en el desarrollo de aplicaciones Web y se encuentra integrado a varios marcos de trabajo para PHP. Presentando como ventaja principal, que ofrece la posibilidad de escribir consultas de base de datos en un lenguaje propio, llamado Doctrine Query Language (DQL) inspirado en el HQL (Lenguaje de Consultas Hibernate) de Hibernate, aunque se puede destacar, además, que permite generación automática del modelo y posibilidad de trabajar con YAML. (13)

Después de haber realizado un estudio del los principales ORM existentes para el lenguaje PHP, viendo sus características esenciales, ventajas y desventajas, se puede concluir que en la actualidad los más utilizados son Propel y Doctrine. Teniendo esto en cuneta se estableció una comparación entre ellos, con el objetivo de determinar el que se utilizará para analizar la estructura del esquema del modelo de datos, en formato YAML, y determinar los elementos y características necesarios a tener en cuenta para validar el fichero que la API a desarrollar debe transformar. Además es validos señalar que estos dos ORM dadas las características que presentan son muy empleados no solo por comunidades de desarrollo Web sino también en la universidad por muchos de sus proyectos productivos.

Comparación entre Doctrine y Propel (14)

Ambos ORMs tienen numerosas características similares, ya que soportan cualquier operación usual en un CRUD (Crear, Obtener, Actualizar y Borrar del inglés *Create, Retrieve, Update and Delete*), ya sea desde crear un nuevo registro o actualizar los existentes. Además, los dos pueden generar las clases PHP del modelo, Propel basado en XML y Doctrine en YAML. También, ambos soportan varios motores de bases de datos, validación de data en los modelos, relaciones entre modelos y herencia simple, aunque en Doctrine es conocida como una herencia concreta. Doctrine soporta otros dos tipos de herencia: Simple, donde todas las clases tienen las mismas columnas, y de Agregación. En la cual se almacena un valor adicional en la tabla, permitiendo la instanciación automática del tipo de modelo correcto cuando se realiza una consulta.

Características que solo Doctrine posee:

Documentación

Sin una buena documentación se hace difícil utilizar cualquier librería, por eso es una de las cuestiones más importantes a tener en cuenta. La documentación de Propel ha sido uno de sus principales problemas, aunque ha ido mejorando aún le falta. Por el contrario, el equipo de Doctrine ha estado perfeccionando constantemente su documentación, que ya es superior, y se está trabajando en la creación de un libro.

Usando las librerías

Lo primero que hacen ambos ORMs es crear las clases del modelo. Doctrine permite escribir un simple archivo YAML, o bien código PHP, si se usa YAML, Doctrine tiene algunos métodos que se pueden llamar en el propio código, o descargar una interface de línea de comandos para construir modelos. La propuesta de Propel para crear modelos requiere escribir un XML.

Operaciones con la base de datos

Las operaciones básicas de un CRUD son muy similares en ambos ORMs. Sin embargo, existe una gran diferencia en la manera en que son hechas con más precisión.

Propel utiliza una propuesta *Criteria/Peer.* A continuación se muestra un ejemplo (Figura # 2):

```php
<?php

  $c = new Criteria();

  $c->add(UserPeer::ID, 10);


  //SELECT all "User" models which have 10 as their ID and join all foreign

  $users = UserPeer::doSelectJoinFoobar($c);
?>
```

Figura # 2: Ejemplo de una consulta utilizando Propel

La propuesta de Doctrine es usar *Doctrine_Query* y un lenguaje SQL personalizado llamado DQL (*Doctrine Query Languaje*). A continuación se muestra un ejemplo (Figura # 3)::

```php
<?php

  $items = Doctrine_Query::create()

    ->from('User u')

    ->leftJoin('u.Foobar')

    ->where('u.id = ?', 10)

    ->execute();
?>
```

Figura # 3: Ejemplo de una consulta utilizando Doctrine

Según el estudio realizado a los principales ORM para PHP existentes, y la comparación efectuada entre Doctrine y Propel que son, sin lugar a dudas, los más conocidos y utilizados se seleccionó Doctrine para realizar el análisis de la estructura del esquema del modelo de datos en formato YAML,

con el objetivo de obtener los elementos necesarios para validar la estructura del fichero a transformar a través de la API.

1.3 Análisis de la estructura del esquema del modelo de datos

1.3.1 Estructura del esquema del modelo de datos asociado a Doctrine en formato YAML

El formato YAML para la configuración, en vez de los tradicionales formatos XML e INI es muy ventajoso y empleado en la actualidad por muchos de los marcos de trabajo para PHP. Este formato YAML indica su estructura mediante la tabulación y es muy rápido de escribir. Se deben tener presentes algunas convenciones para trabajar con ficheros YAML. A continuación se mencionan las más importantes:

Sintaxis YAML (15)

En el formato de los ficheros YAML no se deben utilizar tabulaciones, sino que siempre se deben utilizar espacios en blanco. Los sistemas que procesan YAML no son capaces de tratar con las tabulaciones, estas se deben crear con espacios en blanco como se muestra en la figura # 4.

Norma 1- Los ficheros YAML no permiten los tabuladores

```
3   # No utilizar tabuladores
4   all:
5   -> mail:
6   -> -> webmaster: webmaster@ejemplo.com
7   # Utilizar espacios en blanco
8   all:
9   mail:
10  webmaster: webmaster@ejemplo.com
```

Figura # 4: Ejemplo norma 1

Si los parámetros son cadenas de texto que contienen espacios en blanco al principio o al final, se debe encerrar la cadena entera entre comillas simples. Si la cadena de texto contiene caracteres especiales, también se encierran con comillas simples, como se muestra en la figura # 5.

Norma 2- Las cadenas de texto especiales deben encerrarse entre comillas simples

```
3   error1: Este campo es obligatorio
4   error2: ' Este campo es obligatorio '
5   # Las comillas simples que aparecen dentro de las cadenas de
6   # texto, se deben escribir dos veces
7   error3: 'Este <nowiki>''campo''</nowiki> es obligatorio'
```

Figura # 5: Ejemplo norma 2

Se pueden escribir cadenas de texto muy largas en varias líneas, además de juntar cadenas escritas en varias líneas. En este último caso, se debe utilizar un caracter especial para indicar que se van a escribir varias líneas (se puede utilizar > o |) y se debe añadir una pequeña tabulación (dos espacios en blanco) a cada línea del grupo de cadenas de texto. La figura # 6 muestra este caso.

Norma 3 - Definir cadenas de texto muy largas y cadenas de texto multi-línea

```
# Las cadenas de texto muy largas se pueden escribir en
# varias líneas utilizando el carácter >
# Posteriormente, cada nueva línea se transforma en un
# espacio en blanco para formar la cadena de texto original.
# De esta forma, el archivo YAML es más fácil de leer
frase_para_recordar: >
Vive como si fueras a morir mañana y
aprende como si fueras a vivir para siempre.
# Si un texto está formado por varias líneas, se utiliza
# el carácter | para separar cada nueva línea. Los espacios
# en blanco utilizados para tabular las líneas no se tienen
# en cuenta.
direccion: |
calle, número X
Nombre de ciudad
CP XXXXX
```

Figura # 6: Ejemplo norma 3

Los arreglos se definen mediante corchetes que encierran a los elementos o mediante la sintaxis expandida que utiliza guiones medios para cada elemento del arreglo, como muestra en la figura # 7.

Norma 4- Sintaxis de YAML para incluir arreglos

```
# Sintaxis abreviada para los arrays
idiomas: [ Alemán, Francés, Inglés, Italiano ]
# Sintaxis expandida para los arrays
idiomas:
 - Alemán
 - Francés
 - Inglés
 - Italiano
```

Figura # 7: Ejemplo norma 4

Para definir arreglos asociativos, se deben encerrar los elementos mediante llaves ({ y }) y siempre se debe insertar un espacio en blanco entre la clave y el valor de cada par clave: valor. También existe una sintaxis expandida que requiere indicar cada par clave: valor en una nueva línea y con una tabulación (es decir, con 2 espacios en blanco delante) como se muestra en la figura # 8.

Norma 5- Sintaxis de YAML para incluir arreglos asociativos

```
# Sintaxis incorrecta, falta un espacio después de los 2 puntos
mail: {webmaster:webmaster@ejemplo.com,contacto:contacto@ejemplo.com}
# Sintaxis abreviada correcta para los array asociativos
mail: { webmaster: webmaster@ejemplo.com, contacto: contacto@ejemplo.com }
# Sintaxis expandida para los arrays asociativos
mail:
webmaster: webmaster@ejemplo.com
```

Figura # 8: Ejemplo norma 5

Para los parámetros booleanos, se pueden utilizar los valores *on*, 1 o true para los valores verdaderos y *off*, 0 o false para los valores falsos. La figura # 9 muestra los posibles valores booleanos.

Norma 6- Sintaxis de YAML para los valores booleanos

```
valores_verdaderos: [ on, 1, true ]
valores_falsos: [ off, 0, false ]
```

Figura # 9: Ejemplo norma 6

Es recomendable añadir comentarios (que se definen mediante el caracter #) y todos los espacios en blanco adicionales que hagan falta para hacer más fácil la lectura de los ficheros YAML, como se muestra en la figura # 10.

Norma 7- Comentarios en YAML y espacios adicionales para alinear valores

```
# Esta línea es un comentario
mail:
webmaster: webmaster@ejemplo.com
contacto: contacto@ejemplo.com
admin: admin@ejemplo.com # espacios en blanco para alinear los valores
```

Figura # 10: Ejemplo norma 7

Las líneas que empiezan por # se consideran comentarios y se ignoran, se emplean a lo largo de los ficheros de configuración de cada aplicación, con el único objetivo de informar al desarrollador. De esta forma, para modificar esa opción de configuración, solamente es necesario eliminar el carácter de los comentarios y establecer su nuevo valor. La figura # 11 muestra un ejemplo.

Norma 8 - La configuración por defecto se muestra en forma de comentarios

```
# Por defecto la cache está desactivada
settings:
# cache: off
# Para modificar esta opción, se debe descomentar la línea
settings:
cache: on
```

Figura # 11: Ejemplo norma 8

Todas las opciones que pertenecen a una categoría se muestran tabuladas y bajo el nombre de esa categoría. La configuración es más sencilla de leer si se agrupan las listas largas de pares clave: valor. Los nombres de las categorías comienzan siempre con un punto (.), la figura # 12 muestra un ejemplo de uso de categorías.

Norma 9 - Los nombres de categorías son como los nombres de las claves, pero empiezan con un punto

```
4   all:
5   .general:
6   impuesto: 19.6
7   mail:
8   webmaster: webmaster@ejemplo.com
```

Figura # 12: Ejemplo norma 9

En el ejemplo anterior, mail es una clave y general solo es el nombre de la categoría. En realidad, el fichero YAML se procesa como si no existiera el nombre de la categoría, es decir, como se muestra en la figura # 13. El parámetro impuesto realmente es descendiente directo de la clave *all*.

Norma 10- Los nombres de categorías solo se utilizan para hacer más fácil de leer los ficheros YAML y la aplicación los ignora

```
4   all:
5   impuestos: 19.6
6   mail:
7   webmaster: webmaster@ejemplo.com
```

Figura # 13: Ejemplo norma 10

1.3.2 Análisis de la estructura del esquema del modelo de datos asociado a Doctrine en formato JSON

JSON está constituido por dos estructuras principales, la primera es una colección de pares de nombre/valor. En varios lenguajes esto es conocidos como un objeto, registro, estructura, diccionario, tabla hash, lista de claves o un arreglo asociativo. La segunda estructura es una lista ordenada de valores. En la mayoría de los lenguajes, esto se implementa como arreglos, vectores, listas o secuencias.

Estas son estructuras universales; virtualmente todos los lenguajes de programación las soportan de una forma u otra. Es razonable que un formato de intercambio de datos que es independiente del lenguaje de programación se base en estas estructuras.

Las estructuras del esquema del modelo de datos asociado a Doctrine puede tener diferentes formatos como quedo evidenciado y la transformación entre ellos no es un proceso automático que lleven a cabo los ORM. Es por esto que cualquier transformación entre ellos debe hacerse de forma manual o con el empleo de una herramienta externa dedicada a este fin, teniendo en cuenta las particularidades de cada formato.

1.4 Ambiente de desarrollo

Por investigaciones realizadas anteriormente a nivel de proyecto, ya se encontraban definidas las Tecnologías que formarán parte del ambiente de desarrollo a utilizar para dar solución al problema en cuanto a Metodología y Herramientas, Lenguajes y Frameworks.

Metodología de desarrollo OpenUP

El uso de una metodología es importante para controlar y lograr el ciclo normal de vida de un software. La metodología es el proceso que define Quién debe hacer, Qué, Cuándo y Cómo debe hacerlo, guía a los desarrolladores en la realización de las actividades, durante todo el proceso de creación de un producto.

No existe una metodología de software universal, las particulares de cada proyecto o equipo de desarrollo exigen que el proceso sea configurable. Por políticas del proyecto la metodología que se utilizará para el desarrollo de la API es OpenUp por las siguientes características:

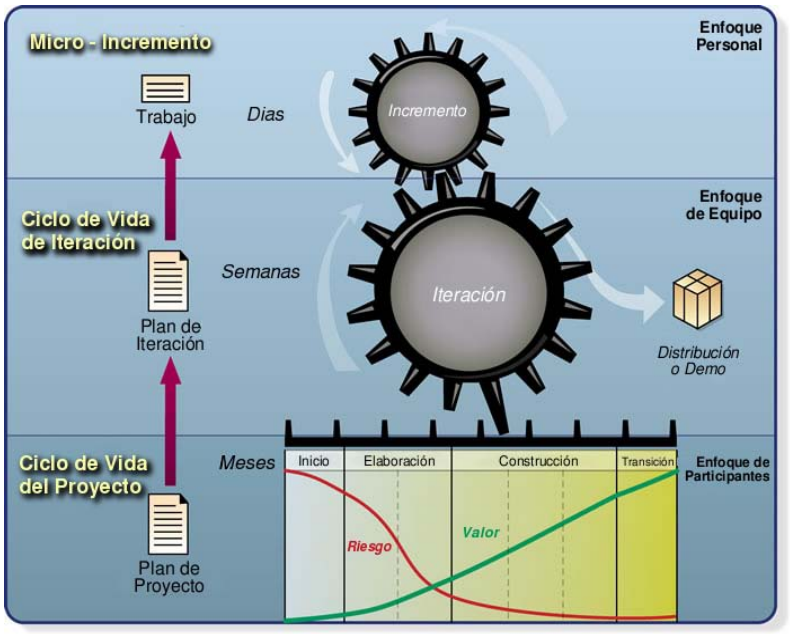


Figura # 14: Las tres capas del OpenUP.

- Es un proceso de desarrollo de software completo en el sentido de que puede ser manifestado como todo el proceso para construir un sistema.
- Es extensible, ya que en el proceso se puede agregar o adaptar según lo vaya requiriendo el sistema. Es un proceso ágil.
- Es ligero y proporciona una comprensión detallada del proyecto, beneficiando a clientes y desarrolladores sobre el producto a entregar y su formalidad.
- Se centra en una arquitectura temprana para reducir al mínimo los riesgos y organizar el desarrollo.
- OpenUP como metodología de desarrollo es conducida por el principio de colaboración para alinear intereses y compartir su comprensión.
- Conserva las características principales del modelo de desarrollo RUP, incluye el desarrollo iterativo, permite identificar los requisitos operacionales del sistema, prevé las interacciones con los usuarios y previene los posibles riesgos en el desarrollo del sistema.
- Preserva la esencia del Proceso Unificado (Desarrollo iterativo e incremental, dirigido por casos de uso, centrado en la arquitectura). Solo lo fundamental está incluido, sin dejar de ser completo y extensible (menos de 20 artefactos).
- OpenUp es la metodología utilizada por desarrolladores de alto nivel en casi todo el mundo por sus altas cualidades administrativas. (15)

Herramienta de Modelado Visual Paradigm 6.4

Visual Paradigm es una herramienta profesional que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. El software de modelado UML ayuda a una más rápida construcción de aplicaciones de calidad, mejores y a un menor costo. Permite dibujar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación, es fácil de usar, soporta la última notación UML 2.1, ingeniería inversa, generación de código, importación desde *Rational Rose*, exportación/importación XMI, generador de informes, editor de figuras, integración con MS Visio, plug-in, integración IDE con Visual Studio, Eclipse, NetBeans. Entre sus nuevas características se incluyen el modelado colaborativo con CVS[3] y *Subversion*, interoperabilidad con modelos UML 2 a través de XMI, abundantes tutoriales y demostraciones interactivas de UML y proyectos UML. (16)

[3] Sistema de Versiones Concurrentes es una aplicación informática que implementa un sistema de control de versiones.

Lenguaje de Modelado UML 2.0

El lenguaje de modelado UML es el estándar de propósito general más utilizado para especificar y documentar cualquier sistema de forma precisa. Proporciona una gran flexibilidad y expresividad a la hora de modelar sistemas. UML es una consolidación de muchas de las notaciones y conceptos más usados orientados a objetos. Empezó como una consolidación del trabajo de Grade Booch, James Rumbaugh, e Ivar Jacobson, creadores de tres de las metodologías orientadas a objetos más populares. UML se puede usar para modelar distintos tipos de sistemas: software, hardware y organizaciones del mundo real. (17)

Lenguaje de programación del lado del servidor PHP 5

PHP, es un lenguaje de script incrustado dentro del HTML, interpretado del lado del servidor generando páginas web una vez compilado. La mayor parte de su sintaxis ha sido tomada de C, Java y Perl, con algunas características específicas de sí mismo. Puede ser utilizado en la mayoría de los sistemas operativos existentes, permitiendo migrar las aplicaciones de un sistema a otro sin necesidad de realizar cambios en el código. Se integra perfectamente a la mayoría de los sistemas gestores de bases de datos. Su rapidez en la ejecución y los bajos requerimientos de consumo en los sistemas donde es desplegado lo hacen uno de los preferidos por los desarrolladores. Su mayor ventaja radica en ser un lenguaje libre, por lo que se convierte en una alternativa de muy fácil acceso, además de poseer una amplia comunidad de desarrolladores. De esta forma, ante cualquier duda, es muy fácil obtener documentación para darle solución de forma rápida y sin costo alguno. (18)

Lenguaje de programación del lado del cliente JavaScript

Al igual que VisualBasic y Perl, JavaScript es un lenguaje interpretado, característica que lo hace especialmente idóneo para trabajar en la Web, son los navegadores que se utilizan para viajar por ella los que interpretan y, por tanto, ejecutan los programas escritos en JavaScript. De esta forma, se envían documentos a través de la Web que llevan incorporados el código fuente de programas, convirtiéndose de esta forma en documentos dinámicos, dejando de ser simples fuentes de información estática. JavaScript comparte muchos elementos con otros lenguajes de alto nivel. (19)

Permite incluir macros en páginas Web, estas macros se ejecutan en el ordenador del visitante de las páginas, y no en el servidor. Hay que tener en cuenta que este lenguaje es muy semejante a otros como C, Java o PHP, tanto en su formato como en su sintaxis, aunque por supuesto tiene sus propias características definitorias. (20)

Lenguaje basado en objetos: es decir, el paradigma de programación es básicamente el de la programación dirigida a objetos, pero con menos restricciones).

Lenguaje orientado a eventos: debido al tipo de entornos en los que se utiliza (Windows y sistemas X-Windows). Gran parte de la programación se centra en describir objetos (con sus variables de instancia y métodos) y escribir funciones que respondan a movimientos del ratón, pulsación de teclas, apertura y cerrado de ventanas o carga de una página, entre otros eventos.

JavaScript proporciona los medios para:

- Controlar las ventanas del navegador y el contenido que muestran.
- Evitar depender del servidor Web para cálculos sencillos.
- Capturar los eventos generados por el usuario y responder a ellos sin salir a Internet.
- Simular el comportamiento de las macros CGI cuando no es posible usarlas.
- Comprobar los datos que el usuario introduce en un formulario antes de enviarlos.
- Comunicarse con el usuario mediante diversos métodos.

Framework de desarrollo Symfony 1.4.8

Framework de desarrollo de aplicaciones con PHP e implementado completamente en este, hace más simple trabajar con aplicaciones Web, dando respuesta de manera rápida a las tareas más frecuentes. Hace uso de los mecanismos de Programación Orientada a Objetos (POO) disponibles en el lenguaje PHP 5 y está enfocado al desarrollo en el mismo.

En busca de mantener su enfoque orientado a objetos, hace uso de un mapeo de objetos a bases de datos empleando los ORM Propel y Doctrine, logrando que el programador se evite realizar grandes consultas SQL. Implementa el patrón arquitectónico Modelo Vista Controlador (MVC), el cual brinda una estructura que hace más fácil realizar cambios en la aplicación. (21)

Framework de presentación ExtJS 3.3.1

ExtJS fue creado inicialmente por Jack Slocum. Empezó siendo un conjunto de librerías y extensiones para Yahoo! User Inteface (YUI), librería desarrollada en JavaScript que explota las potencialidades de AJAX para el desarrollo de *Rich Internet Applications* (RIA). Escrito en JavaScript con la finalidad de

asistir al desarrollo de aplicaciones enriquecidas para Internet. Con el tiempo se convirtió en un framework independiente y a principios de 2007 se creó una compañía para comercializar y dar soporte del framework ExtJS. El mismo cuenta con dos tipos de licencias, GPL y comercial, basado en componentes soportados por recursos para la programación orientada a objetos en JavaScript, los que facilitan la implementación de extensiones y aplicaciones de gran complejidad. ExtJS es uno de los frameworks que, además de flexibilizar el manejo de componentes de la página como el DOM y peticiones AJAX tiene la gran ventaja de crear interfaces de usuario más funcionales. (22)

Herramienta de desarrollo NetBeans 6.9

Permite a los programadores escribir, compilar, depurar y ejecutar programas. Está escrito en Java, pero puede servir para un grupo grande de lenguajes, provee una estructura para los proyectos que se pueden crear junto a este IDE, a partir de un conjunto de componentes de software llamado módulo. Es un producto libre y gratuito, sin restricciones de uso. Algunas de sus características generales son: brinda soporte para PHP, JavaScript y otros lenguajes, ofreciendo completamiento de código. Este proyecto de código abierto tiene gran éxito, con una gran base de usuarios y una comunidad en constante crecimiento.

Una de las ventajas de utilizar NetBeans es justamente su integración con populares frameworks de PHP tales como: Symfony y ZendFramework, lo que brinda la oportunidad de dejar a un lado la consola de comandos de Symfony y centrarse en desarrollar. El editor de PHP es mucho más ágil y a la vez robusto, propiciando el reconocimiento de sintaxis y todo lo que provee la versión 5.3 de PHP. Al mismo tiempo presenta integración con PHP Unit Testing, MySQL, y con varios sistemas de control de versiones e incluye depuración de PHP. (23)

Conclusiones Parciales

Según los elementos abordados en este capítulo se concluye lo siguiente:

- Se realizó el estudio de los principales ORM existentes para el lenguaje PHP y se selecciono el ORM a emplear para la transformación.
- Se determinaron los elementos y características importantes de la estructura del esquema del modelo de datos asociado a Doctrine en formato YAML.
- Se conformó el ambiente de desarrollo, el cual cuenta con:
    - Metodología de Desarrollo: OpenUp.
    - Herramienta de Modelado: Visual Paradigm 6.4.

- Lenguaje de Modelado: UML 2.0.
- Herramienta de Desarrollo: NetBeans 6.9.
- Lenguaje de Programación: PHP 5.
- Tecnologías de Desarrollo: Framework Symfony 1.4.8 y ExtJS 3.3.1

CAPÍTULO II. ANÁLISIS Y DISEÑO DE LA SOLUCIÓN PROPUESTA

En este capítulo se exponen las características fundamentales de la Interfaz de Programación de Aplicaciones para la transformación de esquemas de modelos de datos a notación de objetos JavaScript, el cual incluye una descripción en detalles del proceso de transformación YAML a JSON y viceversa. Además, se realiza el modelo de dominio, la definición y especificación de requisitos (funcionales y no funcionales), el Modelo del sistema, las descripciones textuales correspondientes a los casos de uso (CU), el Modelo del diseño y la realización de los CU del diseño. También se describieron las principales clases de la Interfaz de Programación de Aplicaciones, los estilos arquitectónicos y patrones de diseño utilizados.

Interfaz de Programación de Aplicaciones

Una Interfaz de Programación de Aplicaciones es el conjunto de funciones y procedimientos (o métodos, en la programación orientada a objetos) que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción. Representa una interfaz de comunicación entre componentes de software. Se trata del conjunto de llamadas a ciertas bibliotecas que ofrecen acceso a determinados servicios desde los procesos y representa un método para conseguir abstracción en la programación, de manera general, aunque no necesariamente, entre los niveles o capas inferiores y superiores del software. (24)

Una de las principales ventajas de una API consiste en que proporciona un conjunto de funciones de uso general, por ejemplo, para dibujar ventanas o íconos en la pantalla. 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. Las API, asimismo, son abstractas: el software que proporciona determinada API generalmente es llamado la implementación de esa API. (25)

Solución propuesta

La Interfaz de Programación de Aplicaciones permitirá transformar en formato YAML la descripción del modelo de datos en formato JSON, dada por el Diseñador del MER de la aplicación Lycan o por otra herramienta o aplicación cualquiera que genere dicho fichero en el formato JSON requerido. Otra de las funciones implementadas será realizar la transformación desde un fichero en formato JSON o YAML a un fichero SQL facilitando de esta forma la generación de la base de datos en caso de no utilizar el ORM Doctrine.

Para el proceso de transformación de JSON a YAML y viceversa se realizó el estudio del formato YAML asociado al ORM Doctrine y del JSON, debido a que para la implementación de dicho proceso

es necesario contar con los elementos necesarios que permitan realizar la validación de la transformación en cuanto a la estructura y los tipos de datos que ambos lenguajes manejan.

A continuación se describe detalladamente el proceso de transformación en ambos sentidos y se muestra una representación gráfica del mismo a través de la siguiente imagen:

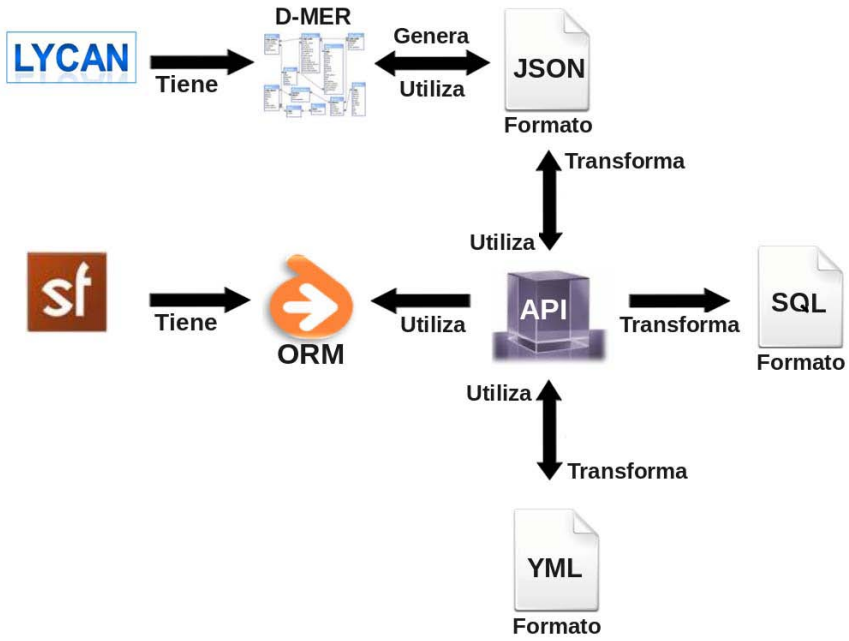


Figura # 15: Proceso de transformación JSON a YAML y viceversa

Proceso de transformación del formato JSON al esquema asociado a Doctrine YAML

El proceso de transformación del formato JSON al esquema asociado a Doctrine YAML comienza cuando el Diseñador del MER del IDE Lycan, o cualquier otra aplicación que cuente con las características necesarias, desee generar la Base de Datos a partir de un fichero en formato JSON que contenga la descripción del modelo de datos, este fichero es utilizado por la interfaz de programación de aplicaciones para realizar la transformación a formato YAML, debido a que es precisamente dicho formato el interpretado por el ORM Doctrine, que después de realizada la transformación es el encargado de generar la Base de Datos correspondiente.

Proceso de transformación del esquema asociado a Doctrine YAML a formato JSON

El proceso comienza cuando el Diseñador del MER del IDE Lycan o cualquier otra aplicación que cuente con las características necesarias necesite hacer un diagrama de entidad relación a partir de un fichero en formato YAML que contenga la descripción de un modelo de datos existente. Este fichero es utilizado por la Interfaz de Programación de Aplicaciones para realizar la transformación a formato

JSON, debido a que es precisamente dicho formato el interpretado por el Diseñador para construir el Modelo Entidad Relación.

Estructura del esquema del modelo de datos asociado a Doctrine en formato YAML

La descripción del Modelo de Datos asociado a Doctrine, contenido en el fichero schema.yml (ver figura # 16), contiene la descripción de las tablas y sus columnas.

```
doctrine:
  jobeet_category:
    id:           ~
    name:         { type: varchar(255), required: true, index: unique }

  jobeet_job:
    id:           ~
    category_id:  { type: integer, foreignTable: jobeet_category,
    foreignReference: id, required: true, onDelete: CASCADE }
    type:         { type: varchar(255) }
    company:      { type: varchar(255), required: true }
    logo:         { type: varchar(255) }
    url:          { type: varchar(255) }
    position:     { type: varchar(255), required: true }
    location:     { type: varchar(255), required: true }
    description:  { type: longvarchar, required: true }
    how_to_apply: { type: longvarchar, required: true }
    token:        { type: varchar(255), required: true, index: unique }
    is_public:    { type: boolean, required: true, default: 1 }
    is_activated: { type: boolean, required: true, default: 0 }
    email:        { type: varchar(255), required: true }
    expires_at:   { type: timestamp, required: true }
    created_at:   ~
    updated_at:   ~

  jobeet_affiliate:
    id:           ~
    url:          { type: varchar(255), required: true }
    email:        { type: varchar(255), required: true, index: unique }
    token:        { type: varchar(255), required: true }
    is_active:    { type: boolean, required: true, default: 0 }
    created_at:   ~

  jobeet_category_affiliate:
    category_id:  { type: integer, foreignTable: jobeet_category,
    foreignReference: id, required: true, primaryKey: true, onDelete: cascade }
    affiliate_id: { type: integer, foreignTable: jobeet_affiliate,
    foreignReference: id, required: true, primaryKey: true, onDelete: cascade }
```

Figura # 16: Estructura del fichero schema.yml

Cada columna se describe con la siguiente información:

- *type*: El tipo de columna (*boolean, integer, float, decimal, string, array, object, blob, clob, timestamp, time, date, enum, gzip*).
- *notnull:* Es true si deseas que la columna sea obligatoria.
- *unique:* Es true si deseas crear un índice único para la columna.

El atributo *onDelete* define el comportamiento ON DELETE para claves foráneas, y Doctrine da soporte para *CASCADE, SETNULL, y RESTRICT.* Por ejemplo, cuando un registro de JobeetTrabajo es borrado, todos los registros jobeet_categoria_afiliado relacionados serán automáticamente eliminados de la base de datos. A continuación se muestra una imagen que representa la estructura de un fichero en formato YAML.

Estructura del esquema del modelo de datos en formato JSON

La descripción del modelo de datos que enviará el Diseñador de MER deberá tener una estructura como se muestra en la figura #17, al igual que cualquier otra aplicación que requiera del servicio que brinda la API que da solución al problema anteriormente planteado. Este fichero schema.json contiene la descripción de todas las tablas, columnas y relaciones.

Cada columna se describe con la siguiente información:

- type: El tipo de columna (*boolean, integer, float, decimal, string, array, object, blob, clob, timestamp, time, date, enum, gzip*).
- notnull: Es true si deseas que la columna sea obligatoria.
- unique: Es true si deseas crear un índice único para la columna.

El atributo onDelete define el comportamiento *ON DELETE* para claves foráneas y Doctrine da soporte para *CASCADE, SETNULL, y RESTRICT*. Por ejemplo, cuando un registro de *job* es borrado, todos los registros jobeet_categoria_afiliado relacionados serán automáticamente eliminados de la base de datos.

```
1   "{"doctrine":{"
2
3   jobeet_category":{"
4   id":null,"
5   name":{"type":"varchar(255)","required":true,"index":"unique"}},"
6
7   jobeet_job":{"
8   id":null,"
9   category_id":{"type":"integer","foreignTable":"jobeet_category","
10  foreignReference":"id","required":true,"onDelete":"CASCADE"},"
11  type":{"type":"varchar(255)"},"
12  company":{"type":"varchar(255)","required":true},"
13  logo":{"type":"varchar(255)"},"
14  url":{"type":"varchar(255)"},"
15  position":{"type":"varchar(255)","required":true},"
16  location":{"type":"varchar(255)","required":true},"
17  description":{"type":"longvarchar","required":true},"
18  how_to_apply":{"type":"longvarchar","required":true},"
19  token":{"type":"varchar(255)","required":true,"index":"unique"},"
20  is_public":{"type":"boolean","required":true,"default":1},"
21  is_activated":{"type":"boolean","required":true,"default":0},"
22  email":{"type":"varchar(255)","required":true},"
23  expires_at":{"type":"timestamp","required":true},"
24  created_at":null,"
25  updated_at":null},"
26
27  jobeet_affiliate":{"
28  id":null,"
29  url":{"type":"varchar(255)","required":true},"
30  email":{"type":"varchar(255)","required":true,"index":"unique"},"
31  token":{"type":"varchar(255)","required":true},"
32  is_active":{"type":"boolean","required":true,"default":0},"
33  created_at":null},"
34
35  jobeet_category_affiliate":{"
36  category_id":{"type":"integer","foreignTable":"jobeet_category","
37  foreignReference":"id","required":true,"primaryKey":true,"onDelete":"cascade"},"
38  affiliate_id":{"type":"integer","foreignTable":"jobeet_affiliate","
39  foreignReference":"id","required":true,"primaryKey":true,"
40  onDelete":"cascade"}}}}"
```

Figura # 17: Estructura del fichero schema.json

Modelo de dominio

El modelo de dominio es una representación visual estática del entorno real objeto del proyecto. Es un diagrama con los objetos reales que existen, relacionados con el sistema que se va a desarrollar y las relaciones que hay entre ellos. Es un artefacto de la disciplina de análisis, construido con las reglas de UML durante la fase de concepción, en la tarea construcción del modelo de dominio, presentado como uno o más diagramas de clases que contiene, no conceptos propios de un sistema de software, sino de la propia realidad física. (26)

Modelo de dominio de la Interfaz de Programación de Aplicaciones

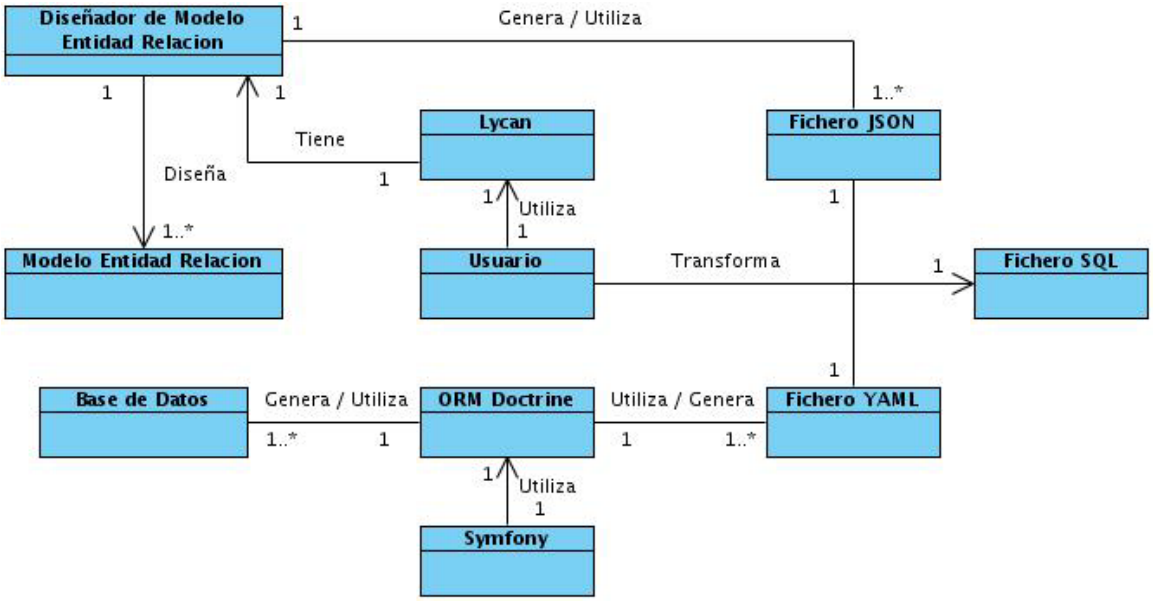


Figura # 18: Modelo de dominio

Descripción de las clases del modelo de dominio

Usuario: Representa el usuario que interactúa con el Diseñador de Modelo Entidad Relación que tiene el IDE Lycan.

Lycan: Representa el Entorno de Desarrollo Integrado llamado Lycan, que permite crear componentes de presentación con el marco de trabajo ExtJS.

Diseñador de MER: Representa el componente Diseñador de Modelo Entidad Relación del IDE Lycan que diseñará el diagrama en modelado de las bases de datos.

Modelo Entidad Relación: Representa el modelo realizado por el usuario de Lycan mediante el Diseñador de MER.

Fichero JSON: Fichero que brinda como salida el diseñador de modelos entidad relación en formato JSON que contiene la descripción del MER.

Fichero YAML: Fichero en formato YAML con la descripción del MER que necesita el ORM Doctrine para generar la base de datos.

ORM Doctrine: Representa la utilización del ORM Doctrine para la generación de la base de datos.

Base de Datos: Representa la base de datos generada por el ORM Doctrine correspondiente al Modelo Entidad Relación diseñado.

Fichero SQL: Representa el fichero transformado a formato SQL por el usuario, para la posterior generación de la base de datos sin la utilización del ORM Doctrine.

Symfony: Marco de trabajo Symfony utilizado para el desarrollo de Aplicaciones Web.

Descripción general del negocio

El usuario utiliza el diseñador de modelos entidad relación que tiene Lycan para diseñar un modelo entidad relación. Una vez concluido el diseño del diagrama, el usuario exporta los datos en un archivo en formato JSON, el cual debe transformar manualmente a formato YAML para generar la base de datos correspondiente haciendo uso del ORM Doctrine. En caso de no disponer del ORM Doctrine debe transformar el fichero exportado a formato SQL. Por otra parte, en caso de que el usuario desee —a través del ORM Doctrine que utiliza Symfony— generar un archivo con la descripción de la base de datos para hacerle algunos cambios a la misma en el diseñador de MER, debe transformar manualmente el archivo generado a formato JSON para que pueda ser interpretado por el diseñador.

2.1 Requisitos Funcionales

Los requisitos funcionales son capacidades o condiciones que el sistema debe cumplir, los cuales definen las funciones que el sistema será capaz de realizar. En el componente que se va a desarrollar se identificaron los siguientes requisitos funcionales:

Requisitos funcionales de la Interfaz de Programación de Aplicaciones

RF 1 Transformar de JSON a YAML.

RF 2 Transformar de YAML a JSON.

RF 3 Transformar de JSON o YAML a SQL.

RF 4 Cargar fichero.

RF 5 Mostrar contenido del fichero.

RF 6 Guardar fichero.

Especificación de los requisitos funcionales de la Interfaz de Programación de Aplicaciones

El artefacto de especificación de requisitos de software debe contener una lista detallada y completa de los requisitos que debe cumplir el componente que se va a desarrollar. El nivel de detalle de los requisitos tiene que ser claro, concreto, completo y consistente, para posteriormente diseñar un componente que satisfaga dichos requisitos.

RF 1: Transformar de JSON a YAML.

Descripción:	Permitirá realizar la transformación de un fichero en formato JSON a YAML.
Entrada:	**Fichero en formato JSON:** esta entrada es un fichero en formato JSON que representa la descripción del modelo de datos.
Salida:	Fichero en formato YAML.

RF 2: Transformar de YAML a JSON.

Descripción:	Permitirá realizar la transformación de un fichero en formato YAML a JSON.
Entrada:	**Fichero en formato YAML:** esta entrada es un fichero en formato YAML que representa la descripción de un modelo de datos.
Salida:	Fichero en formato JSON.

RF 3: Transformar de JSON o YAML a SQL.

Descripción:	Permitirá realizar la transformación de un fichero en formato JSON o YAML a SQL para que pueda ser usado por el Diseñador de Modelo Entidad Relación.
Entrada:	**Fichero en formato JSON:** esta entrada es un fichero en formato JSON que representa la descripción del modelo de datos. **Fichero en formato YAML:** esta entrada es un fichero en formato YAML que representa la descripción de un modelo de datos.
Salida:	Fichero en formato SQL.

RF 4: Cargar fichero.

Descripción:	Permitirá cargar un fichero en formato YAML o JSON.
Entrada:	**Fichero en formato YAML o JSON:** esta entrada representa un fichero en

	formato YAML o JSON.
Salida:	Fichero cargado.

RF 5: Mostrar contenido del fichero.

Descripción:	Permitirá mostrar el contenido del fichero convertido.
Entrada:	**Fichero convertido:** fichero en formato YAML o JSON o SQL.
Salida:	Se muestra el contenido del fichero.

RF 6: Guardar fichero.

Descripción:	Permitirá guardar un fichero.
Entrada:	**Fichero en formato YAML o JSON o SQL:** esta entrada es un fichero en formato YAML o JSON o SQL que representa los ficheros convertidos que se desean guardar.
Salida:	Fichero guardado.

2.2 Requisitos No Funcionales

Los requisitos no funcionales son propiedades o cualidades que el producto debe tener. Estas propiedades o cualidades se refieren a las características que hacen al producto atractivo, usable, rápido o confiable. Por lo general, los requerimientos no funcionales son fundamentales en el éxito del producto; normalmente están vinculados a los requerimientos funcionales, es decir, una vez que se conoce lo que el sistema debe hacer se puede determinar cómo ha de comportarse, qué cualidades o propiedades debe tener.

Requisitos no funcionales de la Interfaz de Programación de Aplicaciones

Fiabilidad

RNF 1 Se debe garantizar la confiabilidad y precisión de la información que se suministra al usuario en cuanto a su veracidad e integridad, durante toda su manipulación, para evitar cualquier tipo de error.

Disponibilidad

RNF 2 La Interfaz de Programación de Aplicaciones estará disponible y permitirá a los usuarios su utilización en el momento que se necesite.

Eficiencia

RNF 3 El tiempo de respuesta de la API no debe exceder 5 segundos.

Restricciones de diseño

RNF 4 Para el desarrollo de las interfaces se utilizará el marco de trabajo ExtJS.

Interfaz de usuario

RNF 5 La aplicación será diseñada con una interfaz amigable que resulte fácil de usar por el usuario.

Soporte

RNF 6 Se recopilará toda la información referente a los defectos y/o no conformidades existentes, para incorporar las mejoras sugeridas al sistema.

Requisito de Licencia

RNF 7 El sistema será desarrollado en base a las políticas del software libre.

Portabilidad

RNF 8 El sistema será multiplataforma: Linux-Windows, permitiendo su ejecución sobre diferentes sistemas operativos sin importar sus versiones y sin necesidad de modificar su código fuente.

Software

Servidor:	• Symfony 1.4.8 • ExtJS 3.3.1 • PHP 5 • Apache 2.0.
Cliente:	• Mozilla Firefox 3.0 (superior).

Hardware

Requerimientos mínimos para el servidor:
• Microprocesador superior a 2.0 GHz

• 512 MB RAM o superior. • 50 MB de Disco Duro como mínimo.
Requerimientos mínimos para la conexión del cliente:
• Microprocesador superior a 1.0 GHz • 256 MB RAM o superior.

2.3 Modelo del sistema

El artefacto fundamental del flujo de trabajo Requisitos, de la metodología OpenUp es el Modelo del sistema que representa las relaciones existentes entre actores y casos de uso. El mismo se construye a partir de la especificación de los requisitos y la modelación gráfica de estos, obteniendo las funciones deseadas para el sistema y su entorno, el cual contiene el Diagrama de casos de uso del sistema y las Descripciones textuales de estos casos de uso. (27)

Actor del sistema

Un actor representa papeles que las personas (o dispositivos) desempeñan como impulsores del sistema. Se comunica con el sistema o producto, que es externo al sistema en sí mismo. Un actor y un usuario no son la misma cosa. Un actor es un rol que un usuario desempeña con respecto al sistema. Es importante destacar el uso de la palabra rol, pues con esto se especifica que un actor no necesariamente representa a una persona en particular, sino, más bien, la labor que realiza frente al sistema. (28)

El Actor de la API será cualquier persona o aplicación que precise de las funcionalidades que la misma ofrece.

Casos de Uso del sistema

Los Casos de Uso del sistema describen el comportamiento del sistema mediante acciones y reacciones desde el punto de vista del usuario. Son el conjunto de operaciones o tareas específicas que se realizan tras una orden de algún agente externo, sea desde una petición de un actor o desde la invocación de otro caso de uso. Representan la descripción de la secuencia de interacciones que se producen entre un actor y el sistema, cuando el actor usa el sistema para llevar a cabo una tarea específica. (29)

Casos de uso de la Interfaz de Programación de Aplicaciones

CU 1 Transformar de JSON a YAML.

CU 2 Transformar de YAML a JSON.

CU 3 Transformar de JSON o YAML a SQL.

CU 4 Administrar fichero.

Diagrama de Casos de Uso

Un diagrama de casos de uso es una representación gráfica de una parte o el total de los actores y casos de uso del sistema incluyendo sus interacciones, estos muestran los distintos requisitos funcionales que se esperan de una aplicación o sistema y cómo se relaciona con su entorno (usuarios o aplicaciones). Los elementos que pueden aparecer en los Diagramas de casos de uso son: actores, casos de uso y relaciones entre ellos. (27)

Patrón de Caso de Uso utilizado

Para la obtención del diagrama de caso de uso (ver figura # 19) se empleó el Patrón Concordancia en su variante Reuso. El mismo fue necesario utilizarlo para modelar la secuencia de acciones que son comunes en los tres casos de uso.

Diagrama de casos de uso de la Interfaz de Programación de Aplicaciones

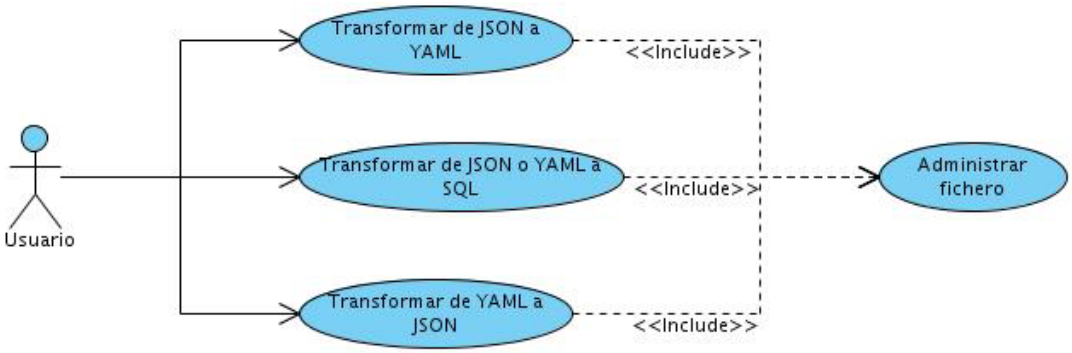


Figura # 19: Diagrama de Casos de Uso

2.3.1 Descripciones textuales de los casos de uso de la Interfaz de Programación de Aplicaciones

Las descripciones textuales se registran en un documento que describe cada uno de los casos de uso, explicando la forma en que interactúa el sistema con el usuario.

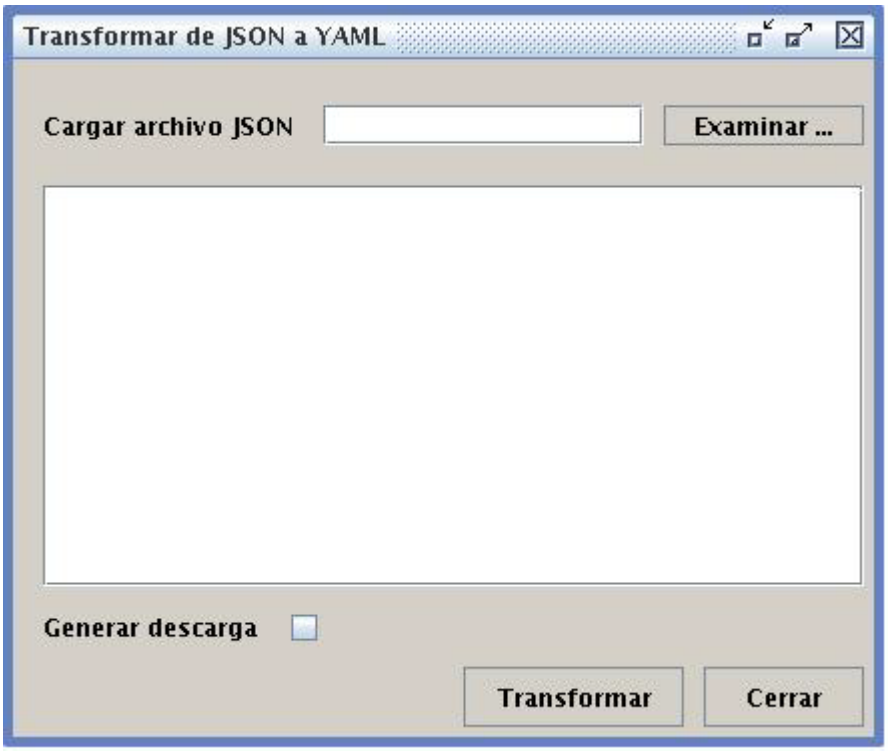
Caso de uso Transformar de JSON a YAML.

Caso de Uso:	Transformar de JSON a YAML.
Actores:	Usuario.
Resumen:	El presente caso de uso permite la transformación de un fichero en formato JSON a YAML.
Precondiciones:	Debe existir un fichero JSON.
Referencias	**RF1, RF 5, CU Administrar fichero (incluido)**
Prioridad	**Alta**

Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1. El caso de uso inicia cuando el actor accede al sistema y selecciona la opción “Transformar fichero JSON a YAML”.	2. El sistema muestra la interfaz correspondiente a Transformar fichero JSON a YAML. En caso de que el actor oprima el botón “Cerrar”, ver *Flujo Alterno* 2.1.
3. Ver descripción textual CU incluido Administrar fichero Sección Cargar fichero.	
4. El actor oprime el botón transformar.	5. El sistema transforma el fichero y muestra un mensaje notificando que la transformación se realizó

	correctamente y aparece el contenido del fichero YAML en el área de texto. En caso contrario ver *Flujo Alterno* 5.1.
6. Ver descripción textual CU incluido Administrar fichero. **Sección** Descargar fichero.	7. Termina el caso de uso.

Prototipo de Interfaz



Flujos Alternos

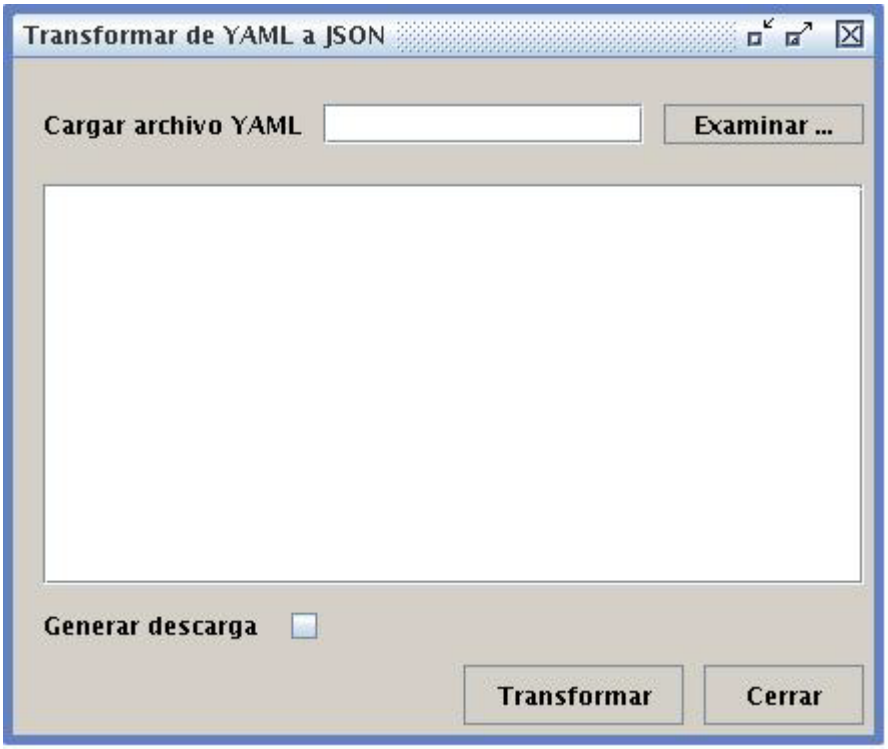
Acción del Actor	Respuesta del Sistema
	2.1 Se cancela el proceso de transformación y termina el caso de uso al cerrar la interfaz.

	5.1 El sistema muestra un mensaje al usuario notificando que no se pudo realizar la transformación.
Poscondiciones	Se transforma un fichero en formato JSON a YAML.

Caso de uso Transformar de YAML a JSON.

Caso de Uso:	Transformar de YAML a JSON.
Actores:	Usuario.
Resumen:	El presente caso de uso permite la transformación de un fichero en formato YAML a JSON.
Precondiciones:	Debe existir un fichero YAML.
Referencias	**RF2, RF 5, CU Administrar fichero (incluido)**
Prioridad	**Alta**

Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1. El caso de uso inicia cuando el actor accede al sistema y selecciona la opción “Transformar fichero YAML a JSON.”	2. El sistema muestra la interfaz correspondiente a Transformar fichero YAML a JSON. En caso de que el actor oprima el botón “Cerrar”, ver *Flujo Alterno* 2.1.
3. Ver descripción textual CU incluido Administrar fichero. **Sección** Cargar fichero.	

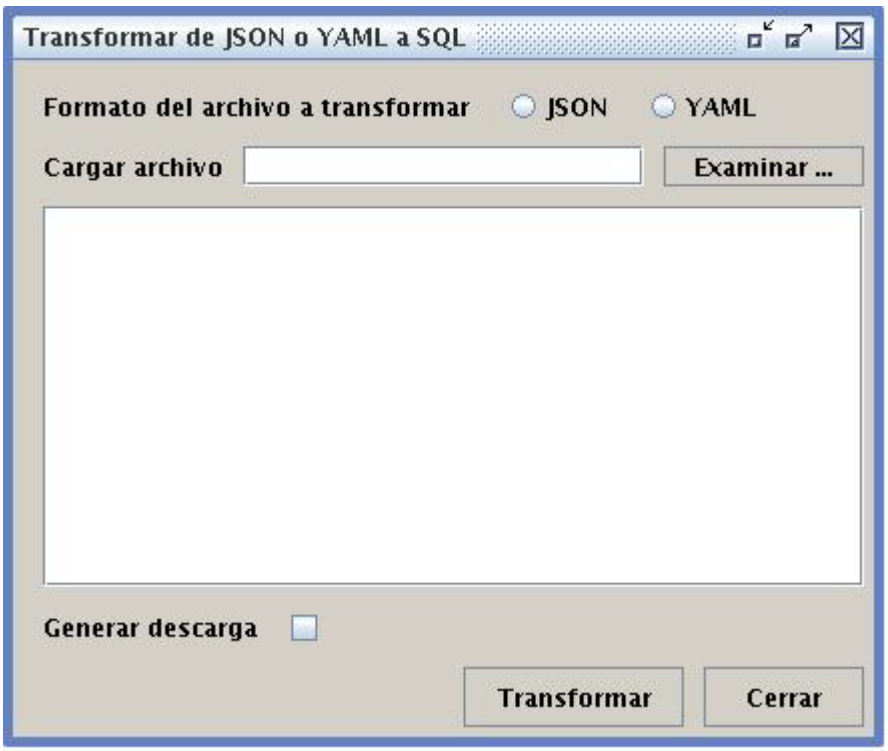
4. El actor oprime el botón transformar.	5. El sistema transforma el fichero y muestra un mensaje notificando que la transformación se realizó correctamente y aparece el contenido del fichero JSON en el área de texto. En caso contrario ver *Flujo Alterno* 5.1.
6. Ver descripción textual CU incluido Administrar fichero. **Sección** Descargar fichero.	7. Termina el caso de uso.
Prototipo de Interfaz	
	
Flujos Alternos	
Acción del Actor	**Respuesta del Sistema**

	2.1 Se cancela el proceso de transformación y termina el caso de uso al cerrar la interfaz.
	5.1 El sistema muestra un mensaje al usuario notificando que no se pudo realizar la transformación.
Poscondiciones	Se transforma un fichero en formato YAML a JSON.

Caso de uso Transformar de JSON o YAML a SQL.

Caso de Uso:	Transformar de JSON o YAML a SQL.
Actores:	Usuario.
Resumen:	El presente caso de uso permite la transformación de un fichero en formato JSON o YAML a SQL.
Precondiciones:	Debe de existir un fichero JSON o un fichero YAML.
Referencias	**RF3, RF 5, CU Administrar fichero (incluido)**
Prioridad	**Alta**
Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**

1. El caso de uso inicia cuando el actor accede al sistema y selecciona la opción “Transformar fichero JSON o YAML a SQL.”	2. El sistema muestra la interfaz correspondiente a Transformar fichero JSON o YAML a SQL. En caso de que el actor oprima el botón “Cerrar”, ver *Flujo Alterno* 2.1.
3. El actor especifica el tipo de formato del fichero que desea transformar.	
4. Ver descripción textual CU incluido Administrar fichero. **Sección** Cargar fichero.	
5. El actor oprime el botón transformar.	6. El sistema transforma el fichero y muestra un mensaje notificando que la transformación se realizó correctamente y aparece el contenido del fichero SQL en el área de texto. En caso contrario ver *Flujo Alterno* 6.1.
7. Ver descripción textual CU incluido Administrar fichero. **Sección** Descargar fichero.	8. Termina el caso de uso.
Prototipo de Interfaz	



Flujos Alternos	
Acción del Actor	**Respuesta del Sistema**
	2.1 Se cancela el proceso de transformación y termina el caso de uso al cerrar la interfaz.
	6.1 El sistema muestra un mensaje al usuario notificando que no se pudo realizar la transformación.
Poscondiciones	Se transforma un fichero en formato JSON o YAML a SQL.

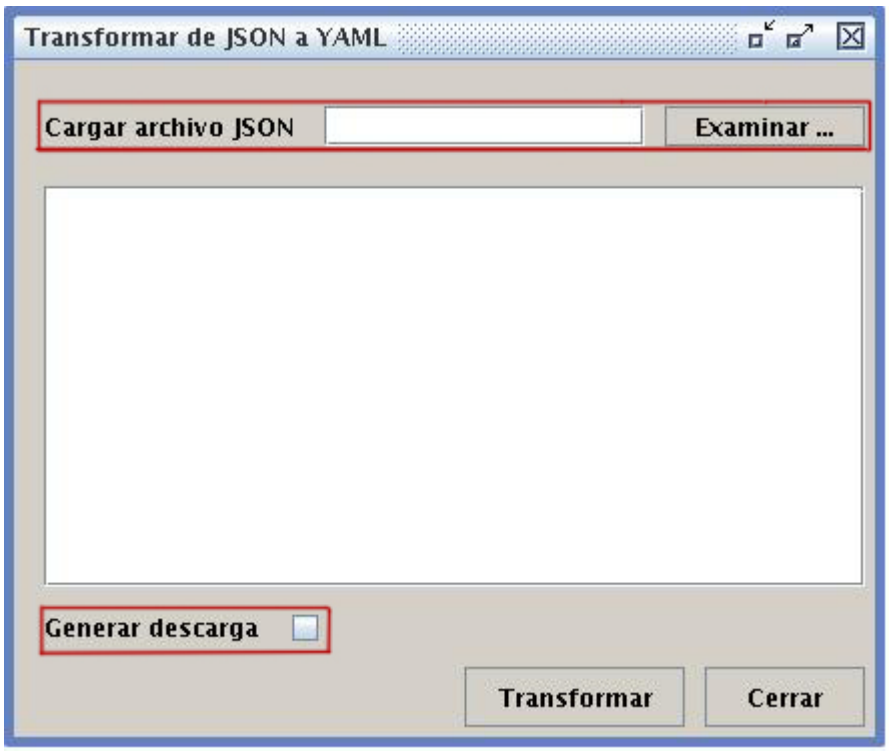
Caso de uso Administrar fichero.

Caso de Uso:	Administrar fichero.
Actores:	Usuario.
Resumen:	El CU se inicia cuando el Usuario va a realizar algunas de las siguientes operaciones: • **Cargar fichero**: permite al Usuario cargar un fichero en formato JSON o YAML, finalizando así el CU. • **Descargar fichero:** permite al Usuario descargar un fichero transformado en formato JSON, YAML o SQL finalizando así el CU.
Precondiciones:	Para poder realizar la descarga debe existir un fichero en formato JSON, YAML o SQL.
Referencias	**RF 4 y RF 6**
Prioridad	**Alta**

Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1. El Usuario selecciona cuál acción desea realizar: a. **Cargar fichero.** b. **Descargar fichero.**	2. El sistema, en dependencia de la acción solicitada por el Usuario, muestra la interfaz correspondiente: a. Cargar fichero: ir a la **Sección** Cargar fichero. b. Descargar fichero: ir a la **Sección** Descargar fichero.

Sección Cargar Fichero: Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**

1. El Usuario selecciona la opción cargar fichero.	2. El sistema muestra la interfaz correspondiente para cargar el fichero.
3. El actor selecciona la ruta del directorio donde se encuentra el fichero.	4. El sistema carga el fichero y termina el CU.
Sección Descargar Fichero: Flujo Normal de Eventos	
Acción del Actor	**Acción del Actor**
1. El Usuario selecciona checkbox Generar descarga.	2. El sistema muestra la interfaz para seleccionar la dirección donde se realizará la descarga del fichero.
	3. El sistema descarga el fichero y termina el CU.
Prototipo de Interfaz	



Poscondiciones	En dependencia de la acción del Genetista: • Se carga un fichero. • Se descarga un fichero.

2.4 Descripción de estilos arquitectónicos y patrones de diseño

Los estilos arquitectónicos definen las reglas generales de la organización en términos de patrones, las restricciones en la forma y la estructura de un grupo numeroso y variado de sistemas de software, además, determinan el vocabulario de componentes y conectores que pueden ser utilizados. Los patrones arquitectónicos especifican un conjunto predefinido de subsistemas con sus responsabilidades y una serie de recomendaciones para organizar los distintos componentes. (30)

2.4.1 Estilo arquitectónico utilizado

A nivel de proyecto se utiliza el estilo arquitectónico llamado “Estilo de llamada y retorno”, y dentro de él, el patrón de arquitectura Modelo Vista Controlador (MVC), utilizado frecuentemente en aplicaciones web para la organización de los componentes de una forma más flexible, modular y reutilizable. Uno de

los objetivos fundamentales de este patrón consiste en separar lo mejor posible las capas del Modelo, de la Vista y del Controlador.
A continuación se describe cómo se evidencia la utilización de dicho patrón en la solución:

El Modelo: Define la lógica de negocio, el sistema de gestión de base de datos; los objetos que interactúan con la base de datos y efectúan los procesos pesados. Es independiente de cualquier representación de salida y/o comportamiento de entrada. La lógica de datos asegura la integridad de estos y permite derivar nuevos datos. Es importante aclarar que el modelo no se refleja en la solución desarrollada.
La Vista: Es utilizada por los usuarios para interactuar con la aplicación, ya que es la presentación final de los datos procesados al cliente, comúnmente en formato HTML, y el código que provee de datos dinámicos a dichas páginas. En Symfony la capa de la vista está formada principalmente por plantillas en PHP. Este presenta el modelo en un formato adecuado para interactuar, de manera usual con la interfaz de usuario.
El Controlador: Recibe las entradas, traducidas a solicitudes de servicio para el modelo. Es un bloque de código que realiza llamadas al modelo para obtener los datos y se los pasa a la vista para que los muestre al usuario. Este responde a eventos, usualmente acciones del usuario, e invoca cambios en el modelo. En fin, representa la separación clara entre el Modelo (lógica de negocio) y la Vista (interfaz gráfica).

2.4.2 Patrones de diseño utilizados

Los patrones de diseño solucionan problemas que existen en varios niveles de abstracción. Con el uso de estos se evita la reiteración en la búsqueda de soluciones a problemas ya conocidos y solucionados anteriormente. Permiten formalizar un vocabulario común entre diseñadores y estandarizar el modo en que se realiza el diseño. (30)

A continuación se muestran los patrones GRASP utilizados:
Una de las principales ventajas a señalar en el uso de un framework es que está basado en patrones de diseño. Symfony está concebido de tal manera que obliga al uso de diferentes patrones de diseño. Los empleados en la solución pertenecen a dos principales grupos. El primero de ellos es el conjunto de patrones GRASP, la utilización de estos ha ayudado a refinar el diseño y a asignar las responsabilidades de las distintas clases de diseño, haciéndolas más sencillas, reutilizables y encapsuladas.

Alta Cohesión: En la solución las clases se caracterizan por no estar sobrecargadas y tener responsabilidades estrechamente relacionadas y enfocadas. Las clases que conforman la Interfaz de Programación de Aplicaciones contienen varias funcionalidades, las que poseen un propósito único, no desempeñado por el resto de las clases, esto hace posible que las clases no se vean afectadas constantemente debido a los cambios y que resulten fáciles de comprender, reutilizar y conservar. Además, una de las características principales del framework Symfony es la organización del trabajo en cuanto a la estructura del proyecto, lo cual permite crear y trabajar con clases, con una alta cohesión.

Bajo Acoplamiento: En la Interfaz de Programación de Aplicaciones las clases se encuentran estructuradas de forma tal que cuentan con la cantidad mínima de asociaciones.

A continuación se muestra el patrón GOF utilizado:

Decorador: El comportamiento de Symfony dispone de un fichero llamado layout.php que contiene las etiquetas <html> y <body>, este fichero, que también se denomina plantilla global, almacena el código HTML, el cual es común a todas las páginas de la aplicación evitando tener que repetir el código en cada página. El patrón Decorador se ve implementado en el contenido de la plantilla que se integra en el layout.

A continuación se muestran los patrones del marco de trabajo ExtJS utilizados:

Instanciación tardía de componente: Los componentes visuales de ExtJS utilizados en la solución se construyen cuando se utiliza la Interfaz de Programación de Aplicaciones debido a que no es necesaria la creación de estos hasta que no se utilicen. De esta manera disminuye la cantidad de recursos en memoria. Lo anteriormente se evidencia en la clase TransformarYAML2JSON donde en vez de crear los componentes fileuploadfield y textarea se les pasa la configuración de los mismos, garantizando su creación cuando se inicializa la API. A continuación se muestra una imagen que pone de manifiesto dicho ejemplo:

```
15          items:  [
16              {
17                    xtype: 'fileuploadfield',
18                    id: 'yml',
19                    name: 'yml',
20                    width: 540,
21                    textAlign: 'left',
22                    allowBlank:false,
23                    emptyText: 'Seleccione yml...',
24                    fieldLabel: 'YML',
25                    buttonCfg: {
26                       text: '',
27                       iconCls: 'upload'
28                       }
29              },
30              {
31                    xtype: 'textarea',
32                    fieldLabel: 'JSON',
33                    id: 'json',
34                    name: 'json',
35                    width: 540,
36                    height: 420
37              }
38          ],
```

Figura # 20: Ejemplo de código de la configuración de los componentes del formulario de la clase TransformarYAML2JSON

Construcción de clases preconfiguradas: En la Interfaz de Programación de Aplicaciones se hace uso de las clases preconfiguradas que son las extensiones de las clases de ExtJS con opciones de configuración implícitas. Ejemplo de esto se encuentra en un fragmento de la clase TransformarYAML2JSON donde al componente FormPanel, que hereda de una clase de ExtJS, se le pasa la configuración predeterminada con la cual va a ser posteriormente creado. A continuación se muestra la imagen correspondiente a dicho ejemplo:

```
5       var formYml2Json = new Ext.FormPanel({
6           id: 'formYml2Json',
7           name: 'formYml2Json',
8           renderTo: 'renderFormYml2Json',
9           fileUpload: true,
10          width: 600,
11          height: 500,
12          frame: true,
13          labelWidth: 35,
14          bodyStyle: 'padding: 10px 10px 10 10px;',
```

Figura # 21: Fragmento de código de la clase TransformarYAML2JSON

2.5 Modelo del Diseño

El Modelo de Diseño se utiliza para documentar el diseño de un sistema. Es un modelo de objeto que describe la realización de los casos de uso y sirve como una abstracción del Modelo de Implementación y del código fuente. Se utiliza como entrada esencial para las actividades en los flujos de trabajo Implementación y Prueba. Es un artefacto integral que abarca todas las clases del diseño y sus relaciones, e incluye los diagramas de clases y de interacción del diseño. (27)

Diagrama de clases

Durante el diseño, el Diagrama de Clases se elabora para tener en cuenta los detalles concretos de la implementación del sistema. En él, la estructura de clases del sistema se especifica, con relaciones entre clases y estructuras de herencia. El Diagrama de clases define los objetos, con los cuales se implementan los casos de uso representando una abstracción de las clases de la implementación del sistema. (28)

Diagrama de clases del diseño de la Interfaz de Programación de Aplicaciones

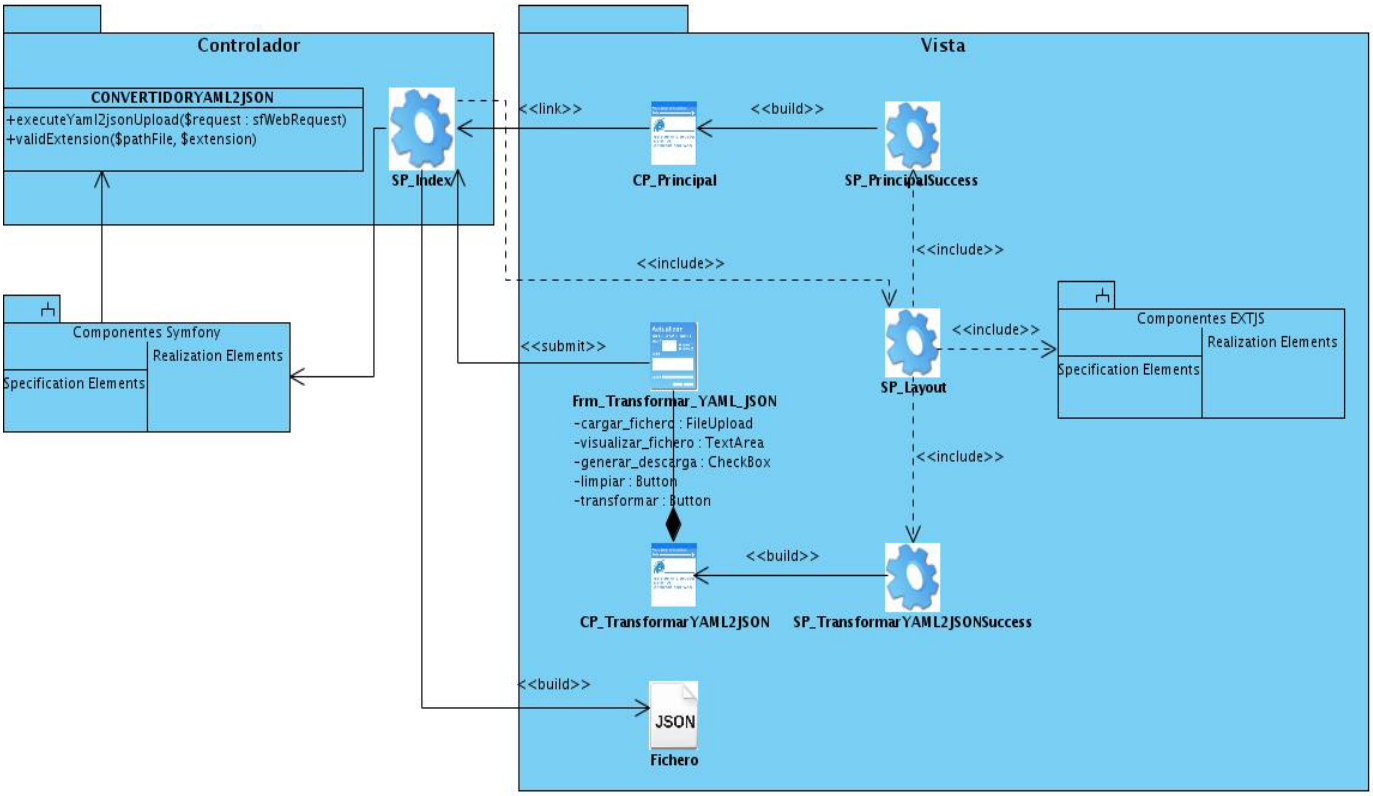


Figura # 22: Diagrama de clases del diseño del caso de uso Transformar de YAML a JSON

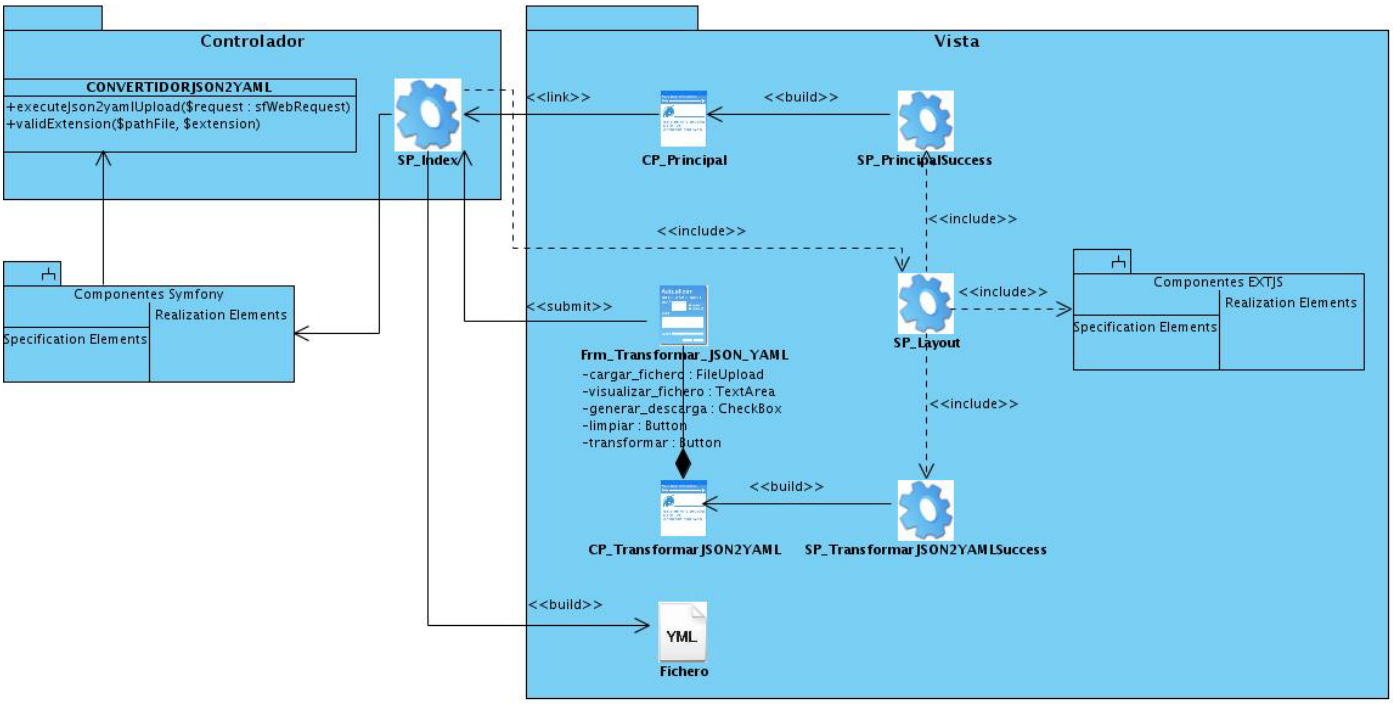


Figura # 23: Diagrama de clases del diseño del caso de uso Transformar de JSON a YAML

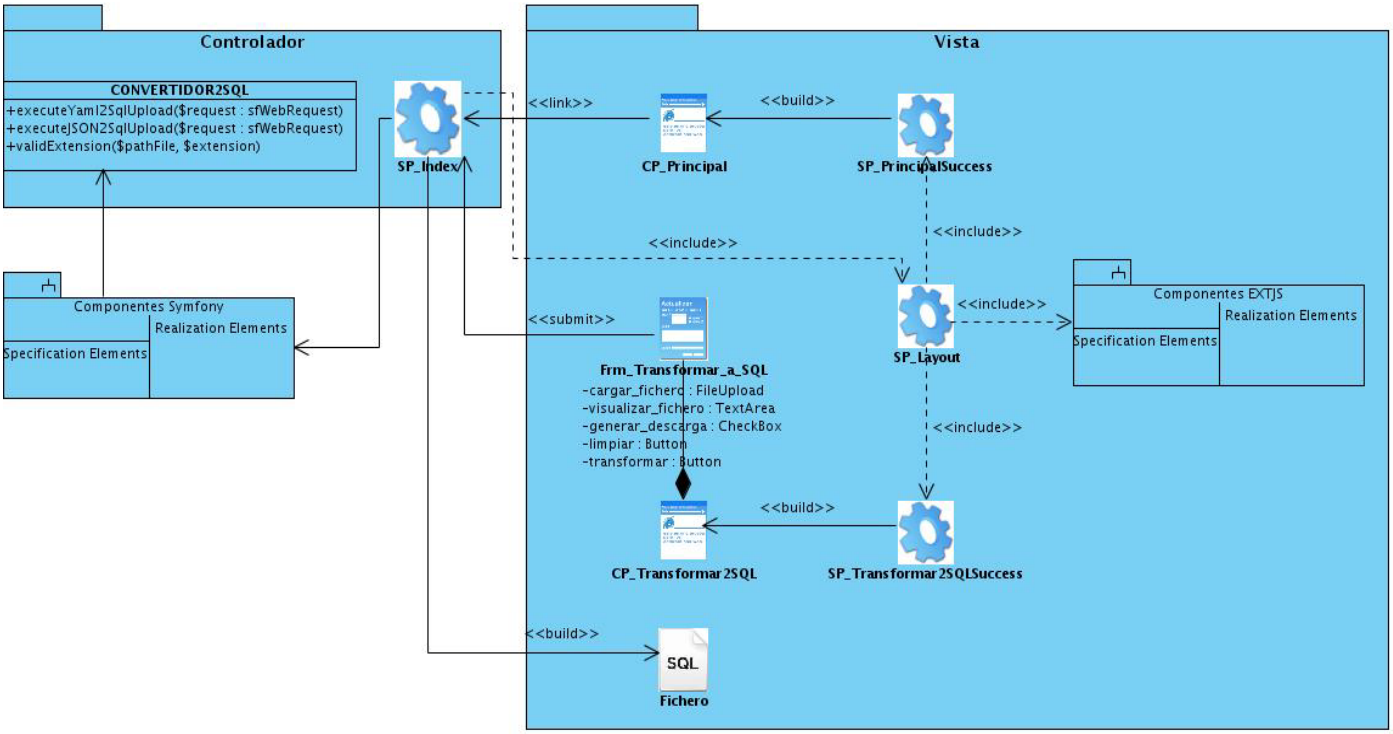


Figura # 24: Diagrama de clases del diseño del caso de uso Transformar de JSON o YAML a SQL

Descripción de las clases del diagrama

Nombre: CONVERTIDORYAML2JSON	
Tipo de clase: Controladora	
Descripción:	
Método	**Descripción**
executeYaml2josnUpload($request: sfWebRequest)	Transforma de formato YAML a formato JSON.
validExtension(pathFile, $extension)	Valida la extensión del fichero cargado.

Nombre: CONVERTIDORJSON2YAML	
Tipo de clase: Controladora	
Descripción:	
Método	**Descripción**
executeJson2yamlUpload($request: sfWebRequest)	Transforma de formato JSON a formato YAML.
validExtension(pathFile, $extension)	Valida la extensión del fichero cargado.

Nombre: CONVERTIDOR2SQL	
Tipo de clase: Controladora	
Descripción:	
Método	**Descripción**
executeJson2SqlUpload($request: sfWebRequest)	Transforma de formato JSON a formato SQL.
executeYaml2SqlUpload($request: sfWebRequest)	Transforma de formato YAML a formato SQL.
validExtension(pathFile, $extension)	Valida la extensión del fichero cargado.

Diagramas de interacción del diseño. Secuencia

Un diagrama de interacción explica gráficamente las interacciones existentes entre las instancias y las clases del modelo de estas. El punto de partida de las interacciones es el cumplimiento de las poscondiciones de los contratos de operación.

El UML define dos tipos de estos diagramas; ambos sirven para expresar interacciones semejantes o idénticas de mensaje:

Los diagramas de colaboración describen las interacciones entre los objetos en un formato de grafo o red, como se aprecia en la figura que se muestra a continuación:

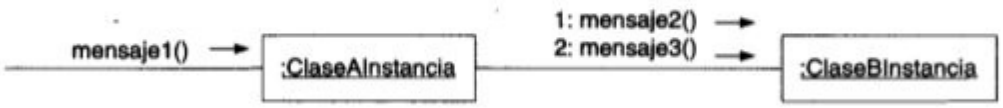


Figura # 25: Ejemplo de diagrama de colaboración

Los diagramas de secuencia describen las interacciones en una especie de formato de cerca o muro, como se observa en la figura que se muestra a continuación:

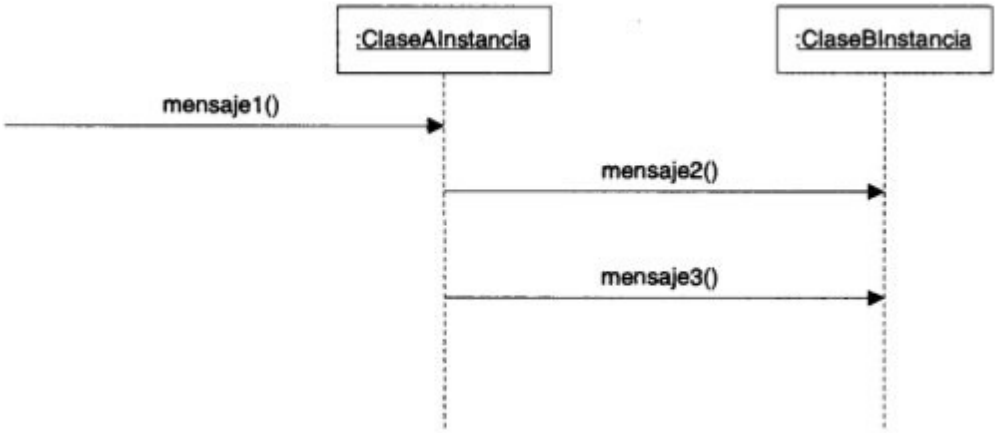


Figura # 26: Ejemplo de diagrama de secuencia

Los diagramas de interacción muestran las interacciones entre objetos mediante transferencia de mensajes entre objetos o subsistemas, por lo que son empleados para modelar aspectos dinámicos del sistema. Los diagramas de colaboración y de secuencia son dos tipos de diagramas de interacción, semánticamente equivalentes, pero, sin embargo, los diagramas de colaboración destacan el orden estructural de los objetos que interactúan y los de secuencia destacan el orden temporal de los mensajes. (28)

Para la realización de los casos de uso del diseño es más factible el empleo de los diagramas de secuencia ya que representan con más claridad el flujo de las acciones que debe realizar el sistema. A continuación se muestran los diagramas de secuencia pertenecientes a la Interfaz de Programación de Aplicaciones:

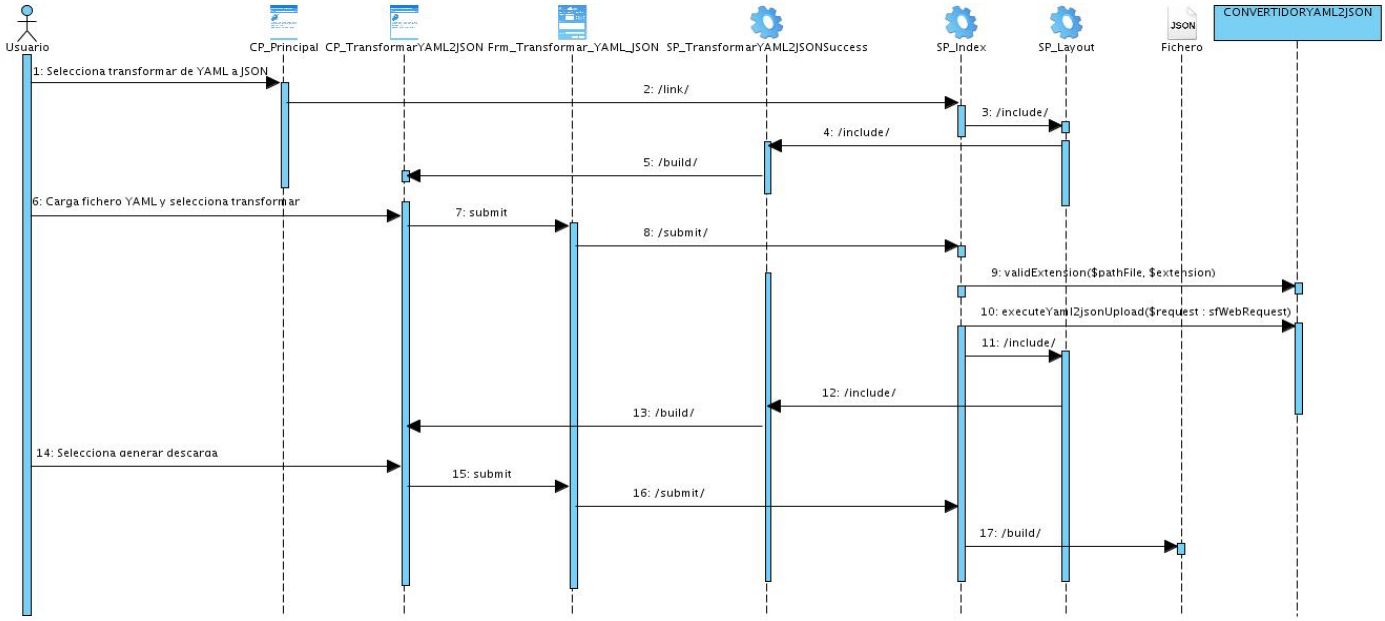


Figura # 27: Diagrama de secuencia del caso de uso Transformar de YAML a JSON

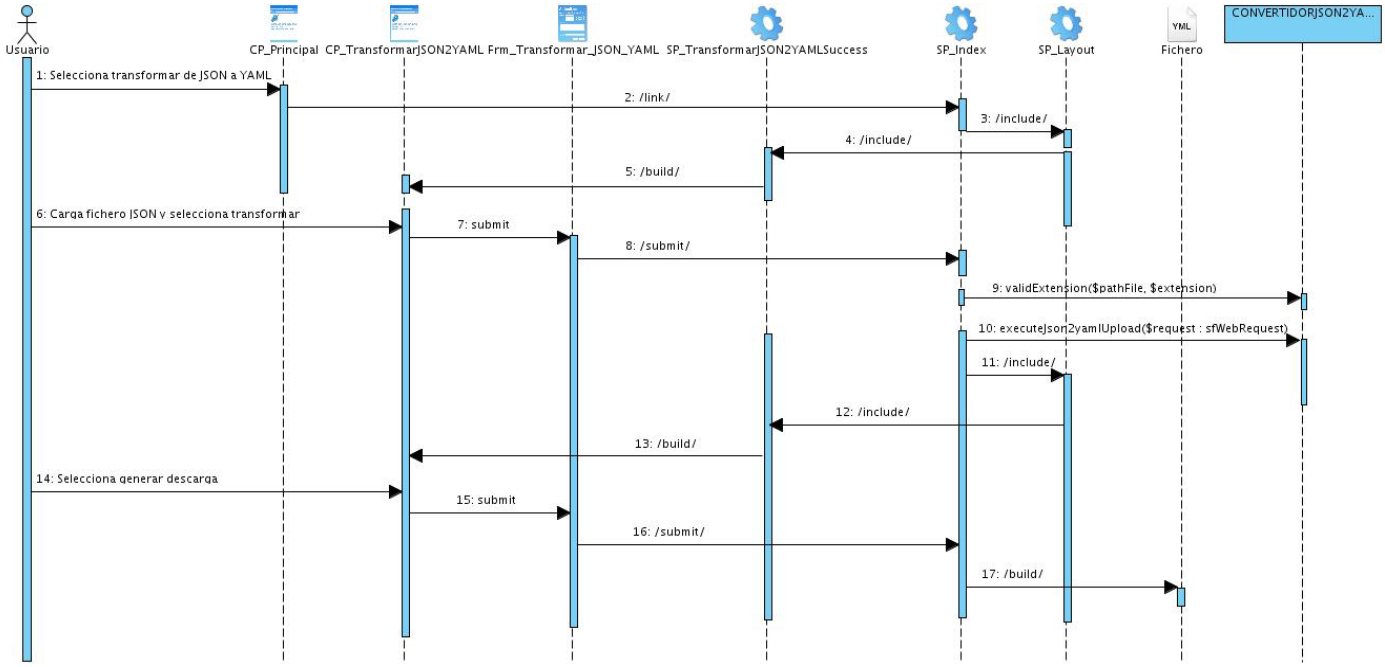


Figura # 28: Diagrama de secuencia del caso de uso Transformar de JSON a YAML

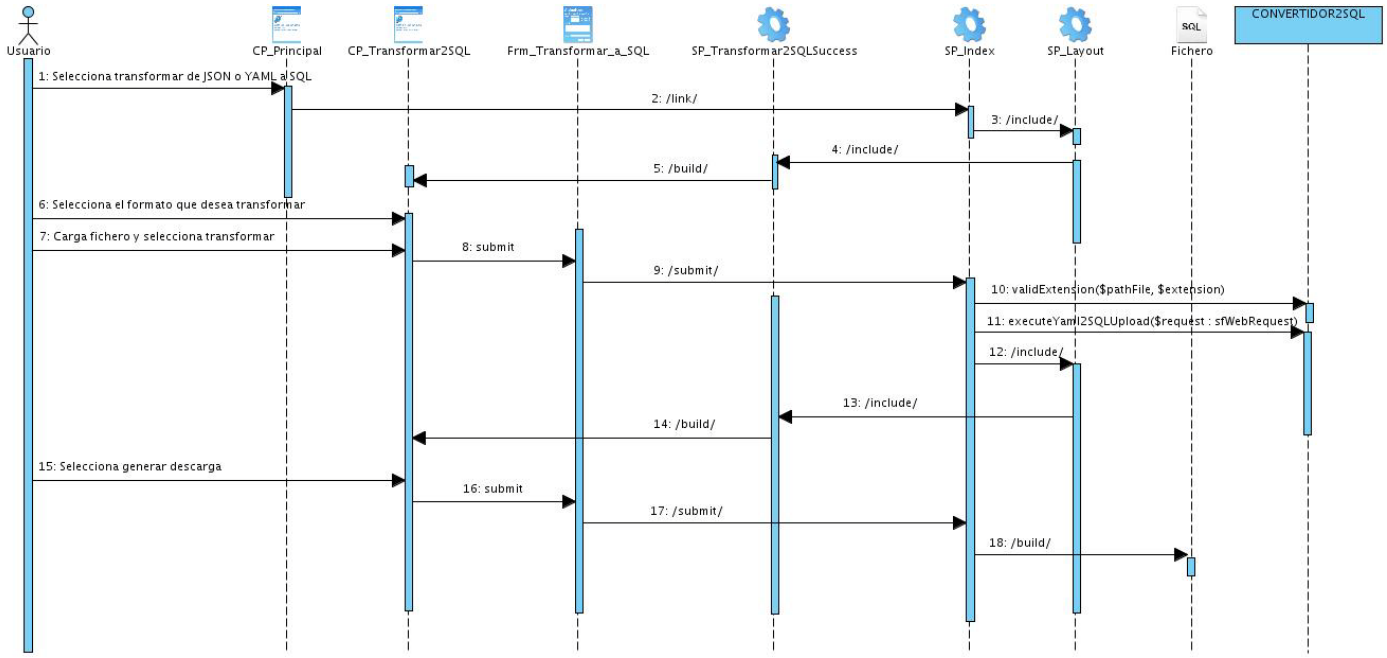


Figura # 29: Diagrama de secuencia del caso de uso Transformar de JSON o YAML a SQL

El diagrama de secuencia del caso de uso Transformar de JSON o YAML a SQL solo se representó para el caso transformar de YAML a SQL, porque para el caso transformar de JSON a SQL es la misma secuencia, solo varía el método que se llama de la controladora.

Diagrama de Despliegue

El Modelo de Despliegue muestra la disposición física de los distintos nodos que componen un sistema y el reparto de los componentes sobre dichos nodos. La vista de despliegue representa la disposición de las instancias de componentes de ejecución en instancias de nodos conectados por enlaces de comunicación. (27)

Se puede observar lo siguiente sobre el modelo de despliegue:

- Cada nodo representa un recurso de cómputo, normalmente un procesador o un dispositivo hardware similar.
- Los nodos poseen relaciones que representan medios de comunicación entre ellos, tales como HTTP, USB, TCP/IP, etc.
- Puede describir diferentes configuraciones de red, incluidas las configuraciones para pruebas y para simulación.
- La funcionalidad (los procesos) de un nodo se define por los componentes que se distribuyen sobre ese nodo.

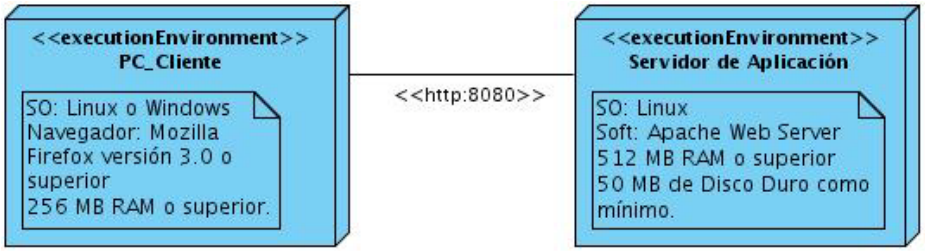


Figura # 30: Diagrama de Despliegue

Conclusiones parciales

Según los elementos abordados en este capítulo para documentar el análisis y diseño de la Interfaz de Programación de Aplicaciones se concluye lo siguiente:

- Se describió de manera detallada el proceso de transformación en ambos sentidos.
- El modelo de dominio quedó conformado por ocho clases con sus respectivas descripciones.
- Se especificaron seis requisitos funcionales y ocho no funcionales.
- El Sistema quedó conformado por tres casos de uso, representados y descritos textualmente en el Modelo del Sistema.
- Se realizaron los diagramas de clases y de interacción, los cuales quedaron registrados en el Modelo de diseño.
- Se describieron los estilos arquitectónicos y patrones de diseño utilizados en la solución.

CAPÍTULO III. IMPLEMENTACIÓN Y PRUEBA

En este capítulo se describe la implementación de la Interfaz de Programación de Aplicaciones en términos de componentes, se realiza el Modelo de implementación que incluye los diagramas de componentes de la interfaz, se muestran fragmentos de código de la implementación y las vistas de las interfaces funcionales. Además, se evalúa la solución mediante la documentación de las pruebas correspondientes.

3.1 Modelo de implementación

El Modelo de implementación es de gran utilidad para la implementación del sistema. Organiza el trabajo para los desarrolladores y describe cómo se implementan en términos de componentes los elementos del modelo de diseño. (28)

Diagrama de componentes

Los diagramas de componentes describen los elementos físicos del sistema y sus relaciones, se utiliza para mostrar la estructura de alto nivel del modelo de implementación en términos de subsistemas de implementación y las relaciones entre los componentes. (28)

Diagrama de componentes de la Interfaz de Programación de Aplicaciones

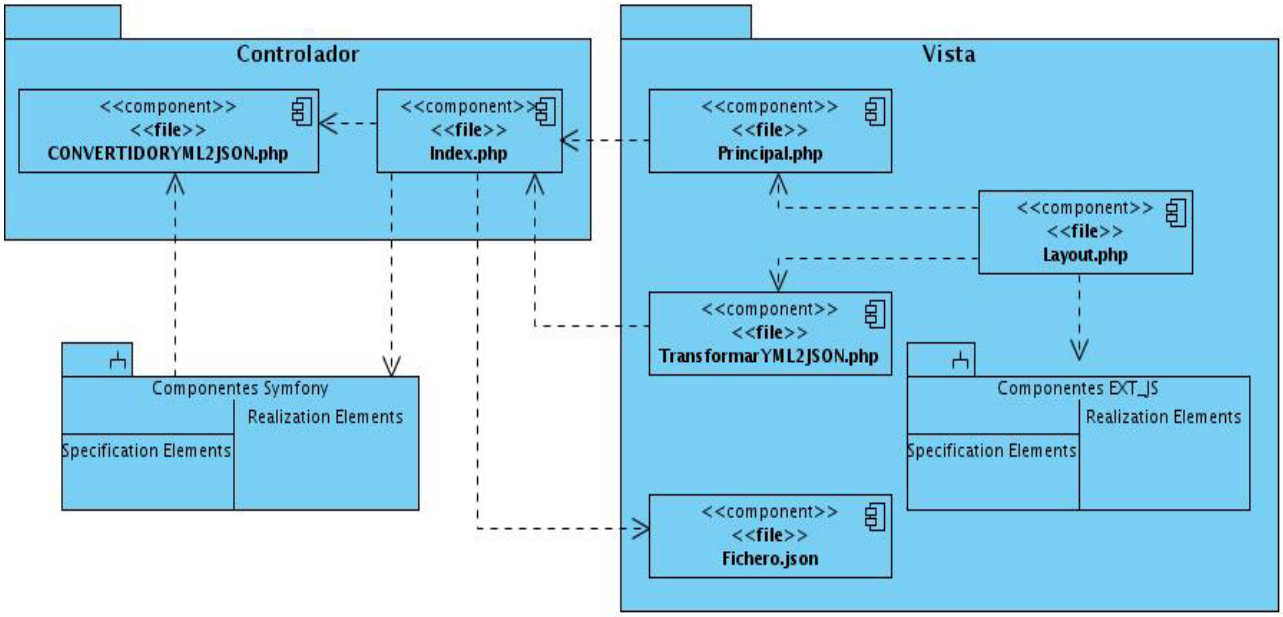


Figura # 31: Diagrama de Componentes del caso de uso Transformar de YAML a JSON

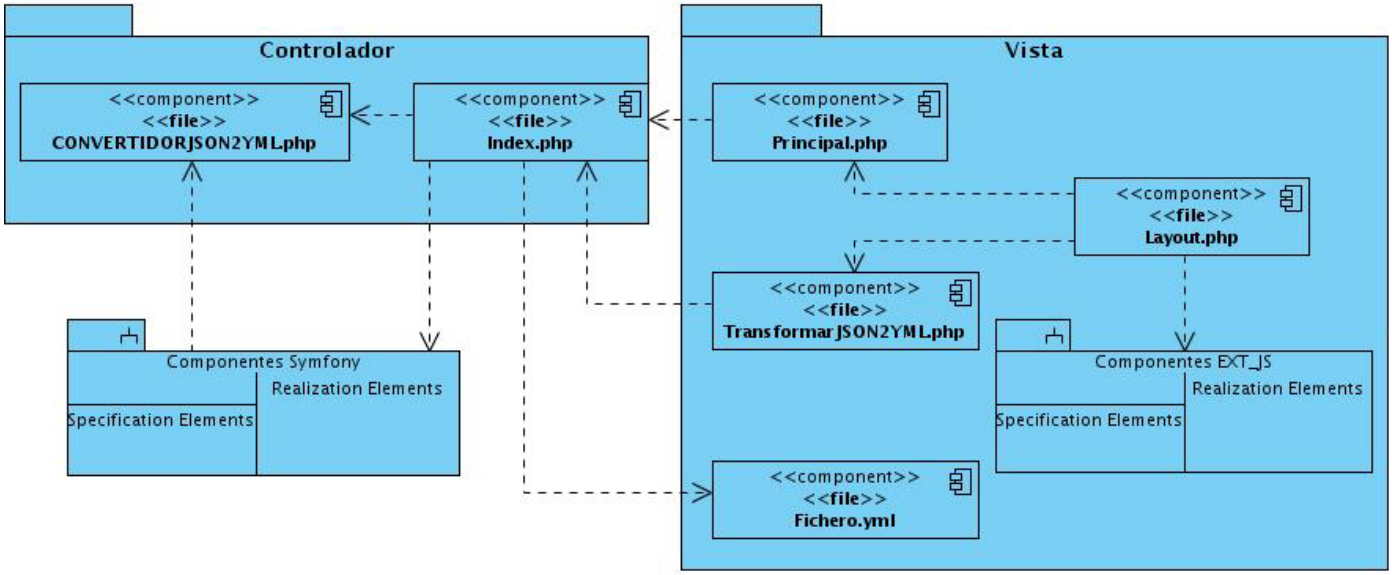


Figura # 32: Diagrama de Componentes del caso de uso Transformar de JSON a YAML

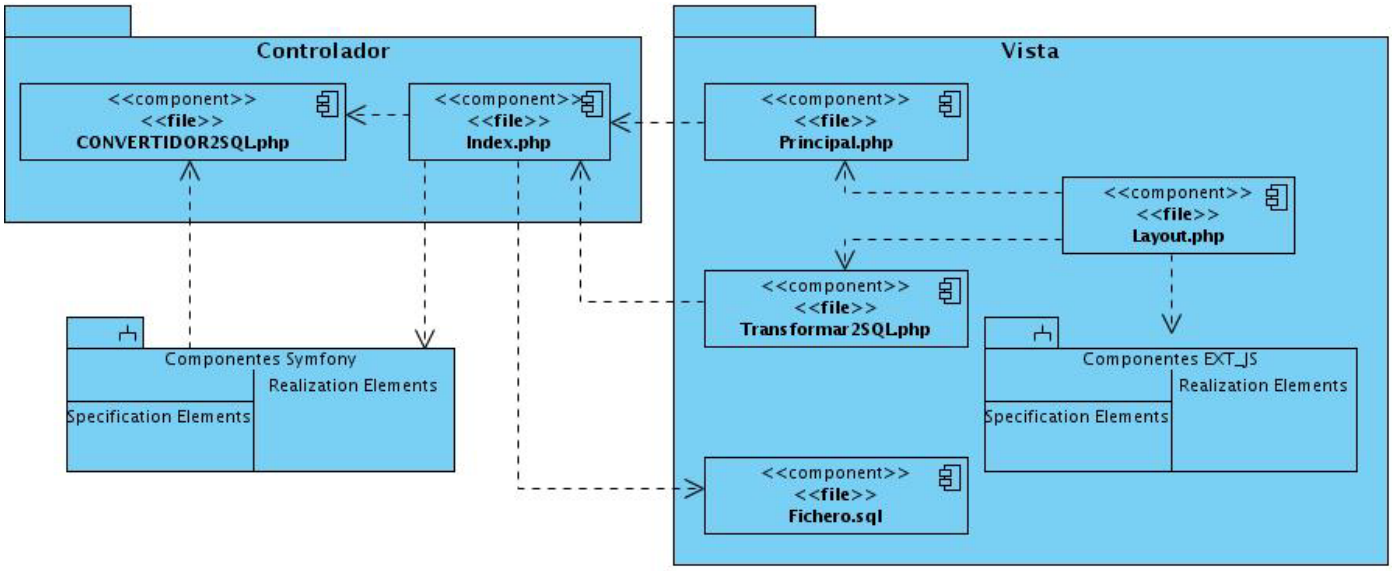


Figura # 33: Diagrama de Componentes del caso de uso Transformar de JSON o YAML a SQL

3.2 Estándar de codificación utilizado

Para obtener una versión funcional de la aplicación se deben implementar los componentes definidos. Como resultado se obtienen ficheros que contienen el código fuente de la aplicación, que no es más que un conjunto de líneas de texto que conforman las instrucciones que debe seguir la computadora para ejecutar dicho programa. Por tanto, en el código fuente de un programa está descrito por completo su funcionamiento. Estas instrucciones son escritas en un lenguaje de programación que consiste en un conjunto de símbolos, reglas sintácticas y semánticas que definen su estructura y el

significado de sus elementos y expresiones. Un estándar de codificación comprende todos los aspectos de la generación de código. Usar técnicas de codificación sólidas y realizar buenas prácticas de programación con vistas a generar un código de alta calidad es de gran importancia para la calidad del software y para obtener un buen rendimiento.

Estilo de codificación utilizado

- Todas las etiquetas php deben ser completas (<?php ?>)... no reducidas (<? ?>).
- Los bloques de código siempre deben estar encerrados por llaves (incluso, si solo constan de una línea).
- Todas las funciones y clases deben estar comentadas. Los comentarios deben ser añadidos de forma que resulten prácticos, para explicar el flujo del código y el propósito de las funciones o variables.
- Indentación: tamaño = 4 (espacios) para:
    - Declaraciones dentro de las clases.
    - Enunciados dentro de métodos y funciones.
    - Enunciados dentro de bloques de comandos.
    - Enunciados dentro de cuerpos switch/case.
- Espacios en blanco:
    - Antes de abrir paréntesis.
    - Luego de abierto el paréntesis.
    - Antes de abrir corchetes.
    - Luego de la coma en declaraciones.
- Líneas en blanco:
- Antes de la declaración de una constante.
- Luego de la declaración de un campo al comienzo del cuerpo de una declaración/método.
- Nueva línea:
- En enunciados vacíos.
- Líneas de código:
- Máximo tamaño de línea (ancho) 300 caracteres.

Fragmento del código fuente de la clase CONVERTIDORYAML2JSON

```
11    public function executeYml2jsonUpload(sfWebRequest $request)
12    {
13         $files = $request->getFiles();
14         $yml = $files['yml'];
15         $arrMsg['success'] = false;
16         if ( $this->validExtension ( $yml [ 'name' ], 'yml' ) )
17         {
18              try
19              {
20                   $rutaYml = '../upLoad/' . uniqid () . '.yml';
21                   move_uploaded_file ( $yml ["tmp_name" ], $rutaYml );
22                   $yml = sfYaml::load ( $rutaYml );
23                   $this->TransformarYaml2Json ( $yml );
24                   $arrMsg [ 'msg' ] = $json;
25                   $arrMsg [ 'success' ] = true;
26              }
27              catch ( Exception $error)
28              {
29                   $arrMsg [ 'msg' ] = 'El formato YML no es válido.';
30              }
31         }
32         else
33         {
34              $arrMsg [ 'msg' ] = 'La extensión del archivo no es válida(.yml).';
35         }
36
37         sfConfig::set ( 'sf_web_debug', false );
38         return $this->renderText ( json_encode ( $arrMsg ));
39    }
```

Figura # 34: Fragmento del código fuente

Vistas de la Interfaz de Programación de Aplicaciones

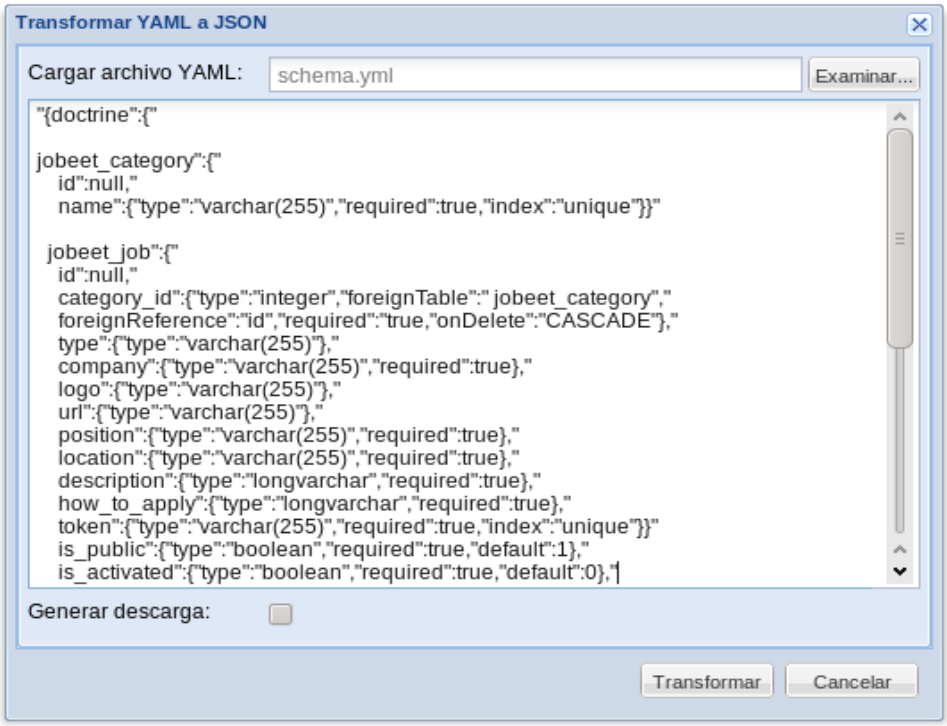


Figura # 35: Interfaz para la transformación de YAML a JSON

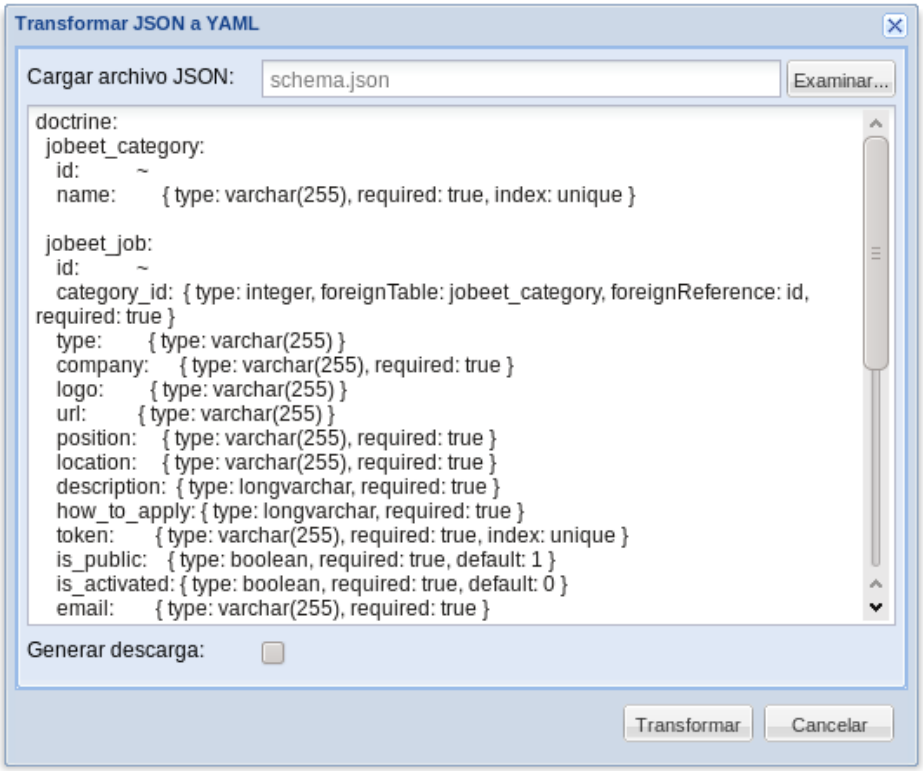


Figura # 36: Interfaz para la transformación de JSON a YAML

3.3 Documentación de la ejecución de las pruebas

Las pruebas de software constituyen un factor importante para la garantía de la calidad del producto, estas se realizan con el objetivo de llevar a cabo la ejecución de un programa con la intención de descubrir y documentar errores, verificar que el software funcione como fue diseñado, validar y probar el cumplimiento de los requisitos y la correcta implementación de estos.

Nivel de Prueba

Existen varios niveles de prueba, donde cada uno contiene una técnica de prueba específica según los atributos de calidad que se deseen verificar. Las técnicas a su vez pueden derivarse en distintos tipos de pruebas, las cuales utilizan métodos para llevar a cabo la ejecución de estas al software.

Los programadores siempre prueban el código durante todo el proceso de desarrollo. Estas son las llamadas pruebas a Nivel de Desarrollador, que es el primer nivel de pruebas, las cuales son diseñadas e implementadas por el equipo de desarrollo.

Técnica de Prueba

Existen varias técnicas disponibles para implementar una prueba, un paso importante es determinar la técnica de prueba a utilizar. La que se aplicará es la Prueba de Funcionalidad, la que se realiza fijando su atención en la validación de las funciones, métodos, servicios y casos de uso. Se analiza cada funcionalidad implementada para verificar que se cumplan todos los requisitos establecidos y de esta forma satisfacer las necesidades existentes.

Tipo de prueba

Dentro de las Pruebas de Funcionalidad se utilizarán las Pruebas Funcionales, que es un tipo de prueba que tiene como objetivo asegurar el trabajo apropiado de los requisitos funcionales, incluyendo la navegación, entrada de datos, procesamiento y obtención de resultados. El método que utiliza este tipo de prueba es el de Caja Negra.

Método de Prueba

El método de Caja Negra consiste en ejecutar cada caso de uso, flujo de caso de uso, o función, usando datos válidos y no válidos, para verificar lo siguiente:

- Que se aplique apropiadamente cada regla de negocio.
- Que los resultados esperados ocurran cuando se usen datos válidos.
- Que sean desplegados los mensajes apropiados de error y precaución cuando se usan datos inválidos.

Las pruebas de Caja Negra pretenden demostrar que las funciones del software son operativas, que la entrada se acepta de forma adecuada y que se produce un resultado correcto.

Casos de Prueba

La aplicación del método de Caja Negra se realizará mediante los Casos de Prueba que constituyen un conjunto de entradas, condiciones de ejecución y resultados esperados, desarrollados para cumplir un objetivo en particular o una función esperada. Para diseñar los Casos de Prueba se utilizó la técnica de Partición de Equivalencia, que divide el campo de entrada de un programa en variables de equivalencia con juegos de datos de entrada y salida. Las variables de equivalencia representan un conjunto de estados válidos y no válidos para las condiciones de entrada de un programa. Se definen dos tipos de variables de equivalencia, las válidas, que representan entradas válidas al programa, y las no válidas, que representan valores de entrada erróneos, aunque pueden existir valores no relevantes a los que no sea necesario proporcionar un valor real de dato. A continuación se muestran las tablas correspondientes al diseño de casos de prueba aplicado a los tres casos de uso de la solución.

Diseño del caso de prueba para el caso de uso Transformar de YAML a JSON

Escenario	Descripción	V1	Respuesta del sistema	Flujo central
EC1. Transformar de YAML a JSON con fichero válido.	En este escenario se transforma un fichero del formato YAML a formato JSON.	V formato válido YAML	El sistema muestra el mensaje "El fichero se transformó correctamente."	1-El usuario selecciona transformar un fichero YAML a formato JSON. 2-Desde el contexto de trabajo, dar un clic en la opción cargar fichero. 3-Seleccionar el fichero a transformar. 4-Dar clic en el botón **Transformar** y obtener el fichero transformado.
EC 2. Transformar de YAML a JSON con fichero incorrecto.		I formato no válido XML, PHP, HTML…	El sistema muestra el mensaje "El formato del fichero no es válido."	
EC 3. Transformar de YAML a JSON con fichero vacío.		vacío formato YAML	El sistema muestra el mensaje "El fichero está vacío."	

Tabla # 2: Caso de Prueba Transformar de YAML a JSON

Diseño del caso de prueba para el caso de uso Transformar de JSON a YAML

Escenario	Descripción	V1	Respuesta del sistema	Flujo central
EC1. Transformar de JSON a YAML con fichero válido.	En este escenario se transforma un fichero del formato JSON a formato YAML	V formato válido JSON	El sistema muestra el mensaje "El fichero se transformó correctamente."	1-El usuario selecciona transformar un fichero JSON a formato YAML 2-Desde el contexto de trabajo, dar un clic en la opción cargar fichero. 3-Seleccionar el

EC 2. Transformar de JSON a YAML con fichero incorrecto.		I formato no válido XML, PHP, HTML…	El sistema muestra el mensaje "El formato del fichero no es válido."	fichero a transformar. 4-Dar clic en el botón **Transformar** y obtener el fichero transformado.
EC 3. Transformar de JSON a YAML con fichero vacío.		vacío formato JSON	El sistema muestra el mensaje "El fichero está vacío."	

Tabla # 3: Caso de Prueba Transformar de JSON a YAML

Diseño del caso de prueba para el caso de uso Transformar de JSON o YAML a SQL

Escenario	Descripción	V1	V2	V3	Respuesta del sistema	Flujo central
EC1. Transformar de JSON o YAML a SQL con fichero válido.	En este escenario se transforma un fichero del formato JSON o YAML a formato SQL.	V formato correcto JSON	V Es marcado	NA	El sistema muestra el mensaje "El fichero se transformó correctamente."	1-El usuario selecciona transformar un fichero JSON o YAML a formato SQL. 2-Desde el contexto de trabajo, selecciona el formato que va a transformar (JSON o YAML). 3-Dar un clic en la opción cargar fichero.
		V formato correcto YAML	NA	V Es marcado		
EC2. Transformar de JSON o YAML a SQL con fichero incorrecto.		I formato no válido JSON	NA	I Es marcado	El sistema muestra el mensaje "El formato del fichero no es válido o no coincide con su selección en el campo Formato del fichero a transformar."	
		I formato no válido YAML	I Es marcado	NA		

EC3. Transformar de JSON o YAML a SQL con fichero vacío.		vacío Formato JSON	V Es marcado	NA	El sistema muestra el mensaje "El fichero está vacío."	4-Seleccionar el fichero a transformar. 5-Dar clic en el botón **Transformar** y obtener el fichero transformado.
		vacío formato YAML	NA	V Es marcado		

Tabla # 4: Caso de Prueba Transformar de JSON o YAML a SQL

Resultados de las pruebas

Después de realizar las pruebas, las dificultades encontradas fueron registradas en la planilla de No Conformidades y solucionadas posteriormente, quedando lista la Interfaz de Programación de Aplicaciones para pasar a otro nivel de prueba. A continuación se muestra la tabla de las No Conformidades encontradas:

Registro de defectos y dificultades detectadas

Elemento	No.	No Conformidad	Aspecto Correspondiente	Etapa de Detección	Estado NC	Respuesta del Equipo de Desarrollo
Interfaz de Programación de Aplicaciones	1	En el escenario 1 del caso de uso Transformar de YAML a JSON existe una falta de ortografía cuando se lanza el mensaje que dice: "El fichero se transformó correctamente."	API/ Transformar de formato YAML a formato JSON / /Ejecutar la transformación del fichero cargado.	Prueba a Nivel de Desarrollador utilizando la técnica de Pruebas de Funcionalidad con el tipo de Pruebas Funcionales a través del Método de Prueba Caja Negra.	22/05/2011 PD 28/05/2011 RA	Se solucionó la dificultad detectada.

Interfaz de Programación de Aplicaciones	2	En el escenario 2 del caso de uso Transformar JSON a YML existe una falta de ortografía cuando se lanza el mensaje que dice: "El fichero se transformó correctamente."	API/ Transformar de formato JSON a formato YAML / /Ejecutar la transformación del fichero cargado.	Prueba a Nivel de Desarrollador utilizando la técnica de Pruebas de Funcionalidad con el tipo de Pruebas Funcionales a través del Método de Prueba Caja Negra.	22/05/2011 PD 28/05/2011 RA	Se solucionó la dificultad detectada.
Interfaz de Programación de Aplicaciones	3	En el escenario 3 del caso de uso Transformar JSON o YAML a SQL existe una falta de ortografía cuando se lanza el mensaje que dice: "El fichero se transformó correctamente."	API/ Transformar de formato JSON o YAML a formato SQL / /Ejecutar la transformación del fichero cargado.	Prueba a Nivel de Desarrollador utilizando la técnica de Pruebas de Funcionalidad con el tipo de Pruebas Funcionales a través del Método de Prueba Caja Negra.	22/05/2011 PD 28/05/2011 RA	Se solucionó la dificultad detectada.

Conclusiones Parciales

Según las actividades realizadas en este capítulo se concluye lo siguiente:

- Se realizó el Modelo de Implementación donde quedaron registrados los tres diagramas de componentes correspondientes a los casos de uso de la Interfaz de Programación de Aplicaciones.
- Se describió el estándar de codificación utilizado.
- La implementación de la Interfaz de Programación de Aplicaciones se logró cumpliendo las funcionalidades necesarias que se definieron para facilitar el trabajo con el Diseñador de Modelo Entidad Relación del IDE Lycan.
- El sistema quedó evaluado verificando que cumpliera con las funcionalidades definidas, mediante las pruebas a Nivel de Desarrollador realizadas donde se detectaron, registraron y solucionaron los tres errores identificados.

CONCLUSIONES

Una vez concluida la presente investigación se arribó a las siguientes conclusiones:

- A partir del análisis realizado al ORM Doctrine se determinaron los elementos y características importantes de la estructura del esquema del modelo de datos asociado a Doctrine en formato YAML a tener en cuenta para realizar la validación de los ficheros a transformar.
- La documentación del Análisis y Diseño de la API se obtuvo a través de la realización de las actividades correspondientes para alcanzar el avance del proceso.
- La implementación de la Interfaz de Programación de Aplicaciones se logró cumpliendo las funcionalidades definidas para obtener las transformaciones necesarias.
- El sistema quedó evaluado mediante las pruebas realizadas a Nivel de Desarrollador verificando que cumpliera con las funcionalidades definidas, donde se registraron y solucionaron los errores identificados.

RECOMENDACIONES

- Integrar la Interfaz de Programación de Aplicaciones al Diseñador de Modelo Entidad Relación del IDE Lycan actualmente en desarrollo.
- Integrar la Interfaz de Programación de Aplicaciones a cualquier otro sistema que cuente con las características necesarias para la aplicación de sus funcionalidades.

REFERENCIAS BIBLIOGRÁFICAS

1. Cumbre Mundial sobre la Sociedad de la Información. [En línea] 12 de Mayo de 2004. [Citado el: 10 de Junio de 2011.] http://www.itu.int/wsis/docs/geneva/official/dop-es.html.

2. Depto de Sistemas y Computación. [En línea] [Citado el: 10 de 10 de 2010.] http://sistemas.itlp.edu.mx/tutoriales/basedat1/tema1_4.htm.

3. **Carrero, Angel.** Programación en castellano. [En línea] [Citado el: 10 de 10 de 2010.] http://www.programacion.com/articulo/conceptos_basicos_de_orm_object_relational_mapping_349.

4. Introducción a JSON. [En línea] [Citado el: 21 de 01 de 2011.] http://www.json.org/json-es.html.

5. **Dulio.** Cristalab. [En línea] 24 de 08 de 2009. [Citado el: 07 de 01 de 2011.] http://www.cristalab.com/tutoriales/tutorial-de-php-y-yaml-c271l/.

6. **Biancardi, Domenico.** Syriusorm. [En línea] 06 de 02 de 2011. [Citado el: 12 de 02 de 2010.] http://syriusorm.blogspot.com/.

7. Php Object Generator. [En línea] [Citado el: 12 de 11 de 2010.] http://www.phpobjectgenerator.com/.

8. ADOdb Active Record. [En línea] [Citado el: 12 de 11 de 2010.] http://phplens.com/lens/adodb/docs-active-record.htm.

9. PhpActiveRecord. [En línea] [Citado el: 12 de 11 de 2010.] http://www.phpactiverecord.org/.

10. **Rotaru, Rostislav.** Sphorm php query generator. [En línea] 17 de 10 de 2010. [Citado el: 12 de 11 de 2010.] http://sphorm.net/documentation.

11. Propel Website. [En línea] [Citado el: 12 de 11 de 2010.] http://www.propelorm.org/.

12. PHP Object Persistence Libraries and More. [En línea] [Citado el: 12 de 11 de 2010.] http://www.doctrine-project.org/.

13. **Emiliano.** Hasheado. [En línea] 16 de Mayo de 2009. [Citado el: 12 de Diciembre de 2010.] http://codeutopia.net/blog/2009/05/16/doctrine-vs-propel-2009-update/.

14. **Fabien Potencier, François Zaninotto.** *Symfony la guía definitiva.* 2008.

15. **Carmina Lizeth Torres Flores, Germán Harvey Alférez Salinas.** *Establecimiento de una Metodología de Desarrollo de Software para la Universidad de Navojoa Usando OpenUP.* Navojoa : s.n., 2008.

16. **Alonso, Evelyn Menéndez.** Herramientas CASE para el proceso de desarrollo de Software. [En línea] Diciembre de 2009. [Citado el: 12 de Enero de 2011.] http://www.monografias.com/trabajos73/herramientas-case-proceso-desarrollo-software/herramientas-case-proceso-desarrollo-software2.shtml#bibliograa.

17. **James Rumbaugh, y Grady Booch Ivar Jacobson.** *The Unified Modeling Language Reference Manual. Addison-wesley.* 2010.

18. **Henst, Christian Van Der.** Maestros del Web. [En línea] [Citado el: 20 de 02 de 2011.] http://www.maestrosdelweb.com/editorial/phpintro/.

19. **Merelo, Juan J.** Metodologías de desarrollo del software. [En línea] 2010 de Octubre de 17. [Citado el: 14 de Noviembre de 2010.] http://latecladeescape.com/ingenieria-del-software/metodologias-de-desarrollo-del-software.html.

20. **Hernández, Marcos Legido.** *Manual JavaScript. Características.* 2009.

21. **Eguiluz, Javier.** Maestros del Web. [En línea] [Citado el: 20 de 02 de 2011.] http://www.maestrosdelweb.com/editorial/el-framework-symfony-una-introduccion-practica-i-parte/.

22. **Shea Frederick, Colin Ramsay, y Steve Cutter Blades.** *Learning Ext JS.* s.l. : Akshara Aware, 2008.

23. **Montes, Ezequiel.** Netbeans. [En línea] [Citado el: 20 de 02 de 2011.] http://netbeans.org/community/releases/69/.

24. **Bongiovanni, Pablo.** Portal Capacitación. [En línea] [Citado el: 12 de Febrero de 2011.] http://portalcapacitacion.educ.ar/cursos-moderados/herramientas-y-aplicaciones-colaborativas-para-el-aula/.

25. **Failurez.** Pyme Crunch. [En línea] 19 de Junio de 2006. [Citado el: 12 de Febrero de 2011.] http://pymecrunch.com/que-es-una-api.

26. **Weitzenfeld, Alfredo.** *Ingeniería de software orientada a objetos con UML, Java e Internet.* s.l. : Cengage Learning Editores, 2005.

27. **Pressman, Roger S.** *Ingeniería del software. Un enfoque práctico.* 2005.

28. **l.Jacobson, G.Booch, J.Rumbauggh.** *El proceso unificado de desarrollo de software.* s.l. : Pearson Addison Wesley, 2000.

29. **García, Joaquín.** Desarrollo de Software Orientado a Objetos. [En línea] 27 de Septiembre de 2003. [Citado el: 03 de Marzo de 2011.] http://www.ingenierosoftware.com/analisisydiseno/casosdeuso.php.

30. **Gonzáles, Mario.** *Arquitectura de Software.*

BIBLIOGRAFÍA

ADOdb Active Record. [En línea] [Citado el: 12 de 11 de 2010.] http://phplens.com/lens/adodb/docs-active-record.htm.

Alonso, Evelyn Menéndez. Herramientas CASE para el proceso de desarrollo de Software. [En línea] Diciembre de 2009. [Citado el: 12 de Enero de 2011.] http://www.monografias.com/trabajos73/herramientas-case-proceso-desarrollo-software/herramientas-case-proceso-desarrollo-software2.shtml#bibliograa.

Arquitectura de Software Dirigida por Modelos.pdf.

Biancardi, Domenico. Syriusorm. [En línea] 06 de 02 de 2011. [Citado el: 12 de 02 de 2010.] http://syriusorm.blogspot.com/.

Bongiovanni, Pablo. Portal Capacitación. [En línea] [Citado el: 12 de Febrero de 2011.] http://portalcapacitacion.educ.ar/cursos-moderados/herramientas-y-aplicaciones-colaborativas-para-el-aula/.

Carrero, Ángel. Programación en castellano. [En línea] [Citado el: 10 de 10 de 2010.] http://www.programacion.com/articulo/conceptos_basicos_de_orm_object_relational_mapping_349.

CBASQA. [En línea] 2 de Septiembre de 2008. [Citado el: 03 de 12 de 2010.] http://cbasqa.wordpress.com/2008/09/02/proceso-de-desarrollo-openup/.

Daniele, Marcela.*Teoría 11: El Arte de Modelar UML.* 2007.

Dpto. de Sistemas y Computación. [En línea] [Citado el: 10 de 10 de 2010.] http://sistemas.itlp.edu.mx/tutoriales/basedat1/tema1_4.htm.

dsadada. http://www.cetic.guerrero.gob.mx/pics/art/articles/113/file.TiposPruebasSoftware.pdf. *http://www.cetic.guerrero.gob.mx/pics/art/articles/113/file.TiposPruebasSoftware.pdf.* [En línea]

Dulio. Cristalab. [En línea] 24 de 08 de 2009. [Citado el: 07 de 01 de 2011.] http://www.cristalab.com/tutoriales/tutorial-de-php-y-yaml-c271l/.

Eguiluz, Javier. Maestros del Web. [En línea] [Citado el: 20 de 02 de 2011.] http://www.maestrosdelweb.com/editorial/el-framework-symfony-una-introduccion-practica-i-parte/.

Emiliano. Hasheado. [En línea] 16 de Mayo de 2009. [Citado el: 12 de Diciembre de 2010.] http://codeutopia.net/blog/2009/05/16/doctrine-vs-propel-2009-update/.

Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides.*Design Pattens-Elements of Reusable Object Oriented Software.*

Failurez. Pyme Crunch. [En línea] 19 de Junio de 2006. [Citado el: 12 de Febrero de 2011.] http://pymecrunch.com/que-es-una-api.

García, Joaquín. Desarrollo de Software Orientado a Objetos. [En línea] 27 de Septiembre de 2003. [Citado el: 03 de Marzo de 2011.] http://www.ingenierosoftware.com/analisisydiseno/casosdeuso.php.

Gonzáles, Mario. *Arquitectura de Software.*

Henst, Christian Van Der. Maestros del Web. [En línea] [Citado el: 20 de 02 de 2011.] http://www.maestrosdelweb.com/editorial/phpintro/.

Hernández, Marcos Legido. *Manual JavaScript. Características.* 2009.

http://www.exa.unicen.edu.ar/catedras/modysim/teoria/casos_de_uso_a.pdf. [En línea]

Introducción a JSON. [En línea] [Citado el: 21 de 01 de 2011.] http://www.json.org/json-es.html.

James Rumbaugh, y Grady Booch Ivar Jacobson.*The Unified Modeling Language Reference Manual. Addison-wesley.* 2010.

l.Jacobson, G.Booch, J.Rumbauggh.*El proceso unificado de desarrollo de software.* s.l. : Pearson Addison Wesley, 2000.

Larman, Craing.*UML y patrones.*

Marciniak, J.J.*Process Models in Software Engineering.* s.l. : 2nd Edition, John Wiley and Sons, New York, December 2001.

MDA y el papel de los modelos en el proceso de desarrolo de software. **Quintero, Juan Bernardo y Anaya, Raquel.** Número 8, Colombia, Medelli : s.n., Diciembre 2007. ISSN 1794-1237.

Merelo, Juan J. Metodologías de desarrollo del software. [En línea] 2010 de Octubre de 17. [Citado el: 14 de Noviembre de 2010.] http://latecladeescape.com/ingenieria-del-software/metodologias-de-desarrollo-del-software.html.

ModelosDeProcesoDeSoftware. http://www.mitecnologico.com/Main/ModelosDeProcesoDeSoftware. [En línea]

Montes, Ezequiel. Netbeans. [En línea] [Citado el: 20 de 02 de 2011.] http://netbeans.org/community/releases/69/.

netbeans.org. http://netbeans.org/index_es.html. [En línea]

Pacheco, Henry Jesus Mendoza. Sistemas de gestión de bases de datos. [En línea] [Citado el: 11 de 12 de 2010.] http://www.monografias.com/trabajos56/sistemas-bases-de-datos/sistemas-bases-de-datos2.shtml.

Perfiles UML y Desarrollo Dirigido por Modelos: Desafíos y Soluciones para Utilizar UML como Lenguaje de Modelado. **Giachetti, Giovanni, Marín, Beatríz y Pastor, Oscar.** Valencia, España : SISTEDES, 2008. ISSN 1988–3455.

Php Object Generator. [En línea] [Citado el: 12 de 11 de 2010.] http://www.phpobjectgenerator.com/.

PHP Object Persistence Libraries and More. [En línea] [Citado el: 12 de 11 de 2010.] http://www.doctrine-project.org/.

PhpActiveRecord. [En línea] [Citado el: 12 de 11 de 2010.] http://www.phpactiverecord.org/.

Popkin.*Modelado de Sistemas com UML.*

Pressman, Roger S.*Ingeniería del software. Un enfoque práctico.* 2005.

Propel Website. [En línea] [Citado el: 12 de 11 de 2010.] http://www.propelorm.org/.

Pruebas_Funcionales.pdf. Pruebas Funcionales. [En línea] [Citado el: 23 de mayo de 2011.] http://carolina.terna.net/ingsw3/datos/Pruebas_Funcionales.pdf..

Rotaru, Rostislav. Sphorm php query generator. [En línea] 17 de 10 de 2010. [Citado el: 12 de 11 de 2010.] http://sphorm.net/documentation.

Shea Frederick, Colin Ramsay, y Steve Cutter Blades.*Learning Ext JS.* s.l. : Akshara Aware, 2008.

visual-paradigm. [En línea] http://www.visual-paradigm.com/product/vpsuite/.

Weitzenfeld, Alfredo.*Ingeniería de software orientada a objetos con UML, Java e Internet.* s.l. : Cengage Learning Editores, 2005.