Universidad de las Ciencias Informáticas

Facultad 3

Título: Módulo de Estructura y Composición para el marco de trabajo Symfony 2.

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

Autores:

Saidel Concepción Ramírez

Randy Herrera Blanco

Tutores:

Ing. Yunet Suárez Abrante

Ing. René Bauta Camejo

La Habana

Junio 2014

"Año del 56 Aniversario de la Revolución".

"La soberanía del hombre está oculta en la dimensión de sus conocimientos".

Ernesto Ché Guevara.

DECLARACIÓN DE AUTORÍA

Declaramos que somos los autores de este trabajo y autorizamos a la Facultad 3 y al Centro de Informatización para la gestión de Entidades de la Universidad de las Ciencias Informáticas que hagan el uso que estimen pertinente con este trabajo.

Para que así conste firmamos la presente a los ____ días del mes de _____ del año 2014.

Saidel Concepción Ramírez

Randy Herrera Blanco

_____
AUTORES

Ing. Yunet Suárez Abrante

_____
TUTOR

Ing. René Bauta Camejo

_____
TUTOR

Doy gracias a mis amigos, a mis queridos compañeros de estudio y profesores que son partícipes y responsables de que este trabajo se haya llevado a cabo de la mejor forma posible.

Agradezco a mi tutora y a mi compañero de tesis por el empeño y esfuerzo dedicado.

A mi novia Belkis por tener tanta paciencia conmigo, por estar a mi lado, ayudarme siempre y dedicarme todo su amor y ternura.

A mi familia por apoyarme siempre, a mi padrastro por comportarse tan bien conmigo y cuidar de mi mamita. A mi hermano que lo amo mucho, por ser ejemplo de sacrificio y abnegación para mí. A mi pequeña y hermosa sobrina por brindarme tantos momentos de felicidad.

En especial doy mil gracias a mi mamita por ser padre y madre al mismo tiempo, por traerme al mundo y darme su amor incondicional. Gracias mamita por estar a mi lado desde que nací, sin ti seguro mi vida no sería tan maravillosa como lo es hoy, te debo todo lo que tengo y todo lo que soy.

Saidel

El hecho de que este trabajo exista se debe primeramente al apoyo incondicional de mis padres, de mis compañeros, amigos y profesores, puesto que siempre estuvieron ahí cuando más los necesitaba, haciendo posible la existencia de este trabajo.

Mil gracias por lograr hacer de mí una persona de futuro, por ayudarme a cumplir mi más grande sueño, ser Ingeniero.

Gracias a la revolución por darme la oportunidad de estudiar en esta maravillosa escuela y convertirme en profesional.

Al tribunal y a la oponente por el respeto que inspiran, por su profesionalidad y ayuda que nos brindaron.

A los amigos de siempre Saidel, René, y Andrés, porque no los hay mejores que ustedes.

A mi querida Universidad de las Ciencias Informáticas por estos maravillosos cinco años y por darme la oportunidad de ser mejor persona.

Randy

A mi novia quien lloró y sonrió en cada momento junto a mí y fue capaz de contenerme cuando todo iba mal. Podría tener cientos de oportunidades de enamorarme en mi vida, pero no tendré ninguna otra oportunidad de enamorarme como lo he hecho con ella.

A mi hermano que si no lo hubiese tenido, mi vida no tendría sentido, porque juntos hemos pasado muchas cosas y siempre nos hemos mantenido unidos, sino sabes cuánto te quiero hoy te lo digo, te amo hermano mío.

A su persona, en honor a ella y en acto perpetuo, dedico este trabajo a mi madre. Nunca podré agradecerle tantas cosas que ha hecho por mí y que me ha enseñado para ser siempre mejor hombre. Eres la persona más importante en mi vida y quiero que eternamente lo recuerdes. Cuídese mi tesoro, donde te encuentres, cuide su alma que a ella le debo y mientras lleve la mía lo seguiré haciendo.

Saidel

Le dedico todo este trabajo y esfuerzo a mis padres que son todo para mí y por eso todo lo que hago es para ellos.

A mi familia en general que siempre ha demostrado ser la mejor familia del mundo.

A mis amigos ya que son una parte muy importante de mí y estar presentes en esta escuela.
A mi tutora por haber puesto todo su conocimiento y empeño al servicio del trabajo.

Randy

Debido a la creciente necesidad del país de poseer un mayor control de los recursos empresariales y obtener un comportamiento común en cada una de las empresas, instituciones y entidades, se hace necesario buscar alternativas que permitan integrar y mejorar la idoneidad de las mismas. En este sentido el presente trabajo se ejecutó con el propósito de implementar un módulo para el marco de trabajo Symfony 2, que soporte la gestión de las estructuras organizacionales en entornos multidominios, y que brinde a los futuros usuarios la posibilidad de definir la estructura organizacional de las empresas. Su elaboración fue regida por el modelo de desarrollo de software definido por el Centro de Informatización de la Gestión de Entidades. Se desarrolló un módulo de Estructura y Composición que permite gestionar las estructuras organizacionales de una empresa. Este sistema resulta novedoso si se toma en consideración que Symfony 2 no cuenta con un módulo similar, por lo que sobresale su alta capacidad y adaptabilidad ya que puede ser reutilizado por otros sistemas de gestión web desarrollados en este marco de trabajo. El módulo posee una funcionalidad para definir una estructura organizacional, el mismo es configurable y genérico logrando con esto que se visualice cualquier definición estructural que se necesite. Fueron aplicadas pruebas de caja negra a la solución propuesta, las cuales validaron la realización satisfactoria de todas las funcionalidades deseadas.

PALABRAS CLAVES

Entorno multidominio, estructura organizacional, composición.

INTRODUCCIÓN

La competencia existente, para la producción de mejores bienes y servicios, ha provocado que las organizaciones se preocupen cada vez más, por los procedimientos administrativos, los procesos productivos y en general las estructuras organizacionales. La composición de una empresa u organización se representa por su estructura organizacional o una forma de organización de acuerdo a sus necesidades, por medio de la cual se pueden ordenar las actividades, los procesos y en si el funcionamiento de la misma.

La capacidad de soportar una estructura organizacional que contenga muchas entidades que puedan adaptarse a las especificidades de cada una de ellas, se convierte en un requisito fundamental en los sistemas de gestión existentes. Esto permite conocer en todo momento a quién pertenece la información que se maneja, establecer permisos sobre la información en dependencia de la jerarquía de los usuarios, posibilita la compartimentación de la información y la seguridad de los datos. Este requisito es aún más necesario en entornos multidominios, que abarcan varias organizaciones que colaboran para satisfacer sus necesidades de negocio.

Symfony 2 es un marco de trabajo muy utilizado en el desarrollo de aplicaciones web de gestión, cuyo uso ha ganado auge en la Universidad de las Ciencias Informáticas. Este marco de trabajo no cuenta con una solución para implementar el multidominio organizacional, dificultando la compartimentación de la información y la réplica de datos. Los proyectos que desarrollan sobre Symfony 2 han buscado alternativas para satisfacer estas necesidades, algunas de ellas han derivado en soluciones a la medida, solo aplicables en sus propios desarrollos.

En otros casos estos proyectos han buscado la integración con Sauxe, marco de trabajo desarrollado por el Centro de Informatización de la Gestión de Entidades (CEIGE), que cuenta con un módulo que gestiona la estructura organizacional de las entidades y define la jerarquía de los elementos que la componen. Esta integración se dificulta debido a que Sauxe utiliza Zend Framework y la comunicación con Symfony 2 se debe realizar mediante servicios, provocando así, que se eleve el costo en cuanto a rendimiento y tiempo; a esto se agrega el hecho de que ambos marcos de trabajo deben permanecer instalados para lograr dicha integración.

Teniendo en cuenta los aspectos anteriormente expuestos, se establece como **problema a resolver**:

¿Cómo lograr que el marco de trabajo Symfony 2 soporte la gestión de estructuras organizacionales en entornos multidominios?

Para dar solución al problema planteado se propone como **objetivo general:**
Implementar el módulo Estructura y Composición para el marco de trabajo Symfony 2 que soporte la gestión de las estructuras organizacionales en entornos multidominios.

Objeto de estudio: La gestión de estructuras organizacionales.

Campo de acción: Sistemas informáticos de gestión de estructuras organizacionales.

Idea a Defender: La implementación del módulo Estructura y Composición para el marco de trabajo Symfony 2 que soporte la gestión de las estructuras organizacionales, permitirá que los sistemas informáticos desarrollados sobre este marco de trabajo puedan ser desplegados en entornos multidominios.

Objetivos Específicos:

1. Fundamentar la investigación mediante la elaboración del marco teórico para sustentar los conceptos y la propuesta de desarrollo del módulo.
2. Realizar el diseño e implementación del módulo Estructura y Composición para mejorar el rendimiento del mismo.
3. Realizar pruebas de caja negra para validar el módulo implementado.

Tareas a cumplir:

1. Elaboración del marco teórico-conceptual con los temas relativos a la gestión de estructuras organizacionales, mediante un estudio del estado del arte de esta materia.
2. Estudio de las metodologías, herramientas y tecnologías a utilizar durante el desarrollo de este trabajo.
3. Levantamiento de los requisitos del sistema para cumplir con las condiciones que debe satisfacer el módulo.
4. Modelación del diseño del módulo a implementar.
5. Implementación del módulo.
6. Validación de los resultados obtenidos a través de pruebas de caja negra.

Métodos Científicos

1. **Métodos teóricos**
    - **Análisis y síntesis:** Ambos métodos empleados en el estudio de la gestión de estructuras organizacionales, permiten separar la información en múltiples

componentes para facilitar su estudio. Posibilitan la unión entre las partes previamente analizadas, facilitan descubrir sus características generales y las relaciones esenciales. Permitiendo la extracción de los elementos más importantes, para seguidamente integrarlos de manera coherente en la solución general del problema anteriormente planteado.

- **Histórico-Lógico:** Permite conocer y comprender el estado del arte de los sistemas informáticos existentes en el mundo, que incorporan un módulo similar a la solución propuesta por la investigación, las distintas etapas por las que ha atravesado, tendencias actuales, regularidades y proyecciones futuras, conociendo así su evolución y desarrollo.

2. **Métodos empíricos**
   - **La entrevista:** Es útil para la captura de requisitos; en la que se realiza una conversación planificada entre el investigador y el o los funcionales para la obtención de los requisitos. Esto constituye uno de los mayores afluentes del conocimiento para la descripción de los requisitos funcionales del módulo Estructura y Composición.
   - **Observación:** La observación del funcionamiento del Sistema Integral de Gestión (XEDRO), desarrollado sobre el marco de trabajo SAUXE, permitirá realizar una valoración de la situación actual, observando diferentes peculiaridades y características en la estructura organizacional de las empresas, entidades e instituciones cubanas, en el intercambio de información entre estas y en la toma de decisiones.

Este trabajo de diploma se encuentra estructurado de la siguiente forma:

Capítulo 1: Fundamentación teórica de la investigación: Se realiza un análisis de los principales conceptos asociados a la investigación, se estudian los sistemas informáticos que gestionan las estructuras organizacionales, sus antecedentes y tendencias, se elige la metodología de desarrollo de software a seguir, se detallan las tecnologías, las herramientas, lenguaje de modelado y de programación a utilizar en la implementación de la solución.

Capítulo 2: Características del sistema: En este capítulo se define la propuesta de solución del sistema, para ello se obtiene el modelo conceptual, se describen los requisitos funcionales y no funcionales del sistema, se aplican las técnicas de captura de requisitos y se validan los mismos a través de técnicas de validación. También se define la arquitectura del sistema, se detallan los patrones de diseño y arquitectónicos

utilizados, se elabora el diagrama de clases de diseño, modelo de datos y diagrama de componentes.

Capítulo 3: Validación de la solución: Se lleva a cabo la validación de la solución, a través del uso de métricas de software orientadas a clase para evaluar el diseño y se realizan pruebas de caja negra para verificar el correcto funcionamiento del sistema y comprobar los resultados obtenidos.

1. CAPÍTULO 1. FUNDAMENTACIÓN TEÓRICA DE LA INVESTIGACIÓN

En el presente capítulo se exponen aquellos conceptos asociados al dominio del problema que permiten la comprensión de la investigación, así como la descripción del objeto de estudio como precedente teórico importante. Se hace una breve reseña de los sistemas informáticos existentes que incorporan un módulo para la gestión de las estructuras organizacionales, sus principales características y tendencias actuales. Se desarrolla además el estudio de los lenguajes de programación, de modelado, las herramientas, tecnologías y la metodología de desarrollo de software a utilizar para la implementación de la solución, con el objetivo de seleccionar aquellas que puedan brindar mejores resultados.

1.1. Conceptos básicos asociados al dominio del problema

Los elementos más significativos a tener en cuenta para llevar a cabo y tener éxito en una investigación son la calidad y claridad de los conceptos relacionados con la temática del problema científico planteado. Para este fin se detallan los conceptos fundamentales que aborda la investigación.

1.1.1. Organización

Las organizaciones son sistemas sociales compuestos por individuos que, mediante la utilización de recursos financieros, materiales y humanos, desarrollan un sistema de actividades interrelacionadas y coordinadas para el logro de un objetivo común, dentro de un contexto con el que interactúan de manera permanente (Baryolo, 2012).

“Una organización es un sistema diseñado para alcanzar ciertas metas y objetivos. Estos sistemas pueden, a su vez, estar conformados por otros subsistemas relacionados que cumplen funciones específicas” (Colectivo de autores, 2014).

1.1.2. Entornos multidominios

“Escenarios donde sea necesario lograr la interoperabilidad entre dominios heterogéneos con diferentes políticas de seguridad, manteniendo la capacidad de evolucionar ante las necesidades de cambios operacionales y de los servicios que se ofrecen. Estos escenarios se han convertido en una realidad que se evidencia en la mayoría de las aplicaciones empresariales basadas en Internet” (Joshi, y otros, 2001).

“Entornos que abarcan varias organizaciones que colaboran para satisfacer sus necesidades de negocio” (Shafiq, y otros, 2006).

“Un entorno multidominio es una estructura jerárquica compleja, que agrupa a varias organizaciones atendiendo a un criterio común, para facilitar el acceso e

interoperabilidad entre sus sistemas informáticos distribuidos de una forma segura" (Iranmanesh, 2008).

Después de analizar los conceptos abordados, se llega a la conclusión de que un entorno multidominio es una estructura jerárquica compleja, que permite la interoperabilidad entre sus sistemas informáticos de manera segura. Engloba varias organizaciones siguiendo un criterio común, las cuales colaboran, ofrecen servicios y evolucionan, para satisfacer sus necesidades de negocio.

1.1.3. Estructura organizacional

Las estructuras organizacionales forman la base para la compartimentación de la información en entornos multidominios. La composición de una organización se representa por su estructura organizacional y se entiende por esta a los diferentes patrones de diseño que se utilizan para organizar una empresa, con el fin de cumplir las metas propuestas y lograr el objetivo deseado.

"La estructura organizacional, es el marco en el que se desenvuelve la organización, de acuerdo con el cual las tareas son divididas, agrupadas, coordinadas y controladas, para el logro de objetivos. Desde un punto de vista más amplio, comprende tanto la estructura formal (que incluye todo lo que está previsto en la organización), como la estructura no formal (que surge de la interacción entre los miembros de la organización y el medio externo a ella) dando lugar a la estructura real de la organización" (Colectivo de autores, 2007).

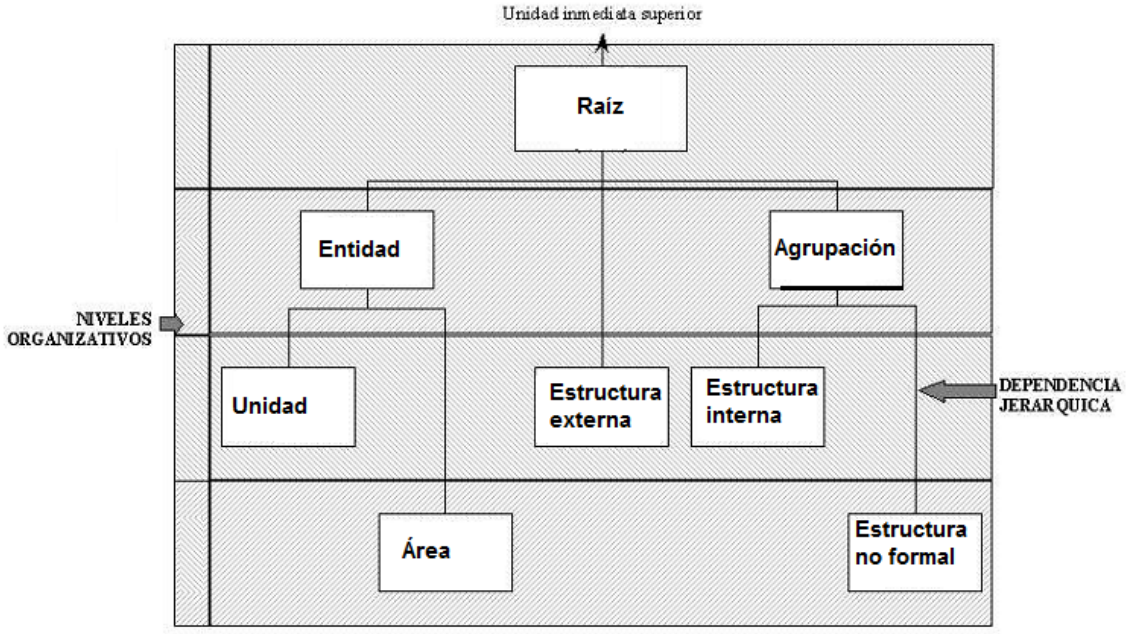
- **Estructura formal**: "Incluye funciones, actividades, relaciones de autoridad y de dependencia, responsabilidades, objetivos, manuales y procedimientos, descripciones de puestos de trabajo, asignación de recursos, y todo aquello que está previamente definido de alguna manera" (Hintze, 2011).
- **Estructura no formal**: "Se conforma a partir de las relaciones entre las personas que comparten uno o varios procesos de trabajos dentro de la organización, valores, intereses, sentimientos, afectos, liderazgo y todas las relaciones humanas que no pueden ser determinadas previamente" (Hintze, 2011).

La representación más simple de la configuración de una estructura organizacional es el organigrama, una forma clásica de graficar que muestra las unidades organizativas con sus dependencias jerárquicas.

- **Organigrama**: "Es una representación gráfica de la estructura formal de una organización, constituye los niveles jerárquicos de autoridad, sus relaciones externas, internas, funcionales, de coordinación y algunos de ellos comprenden

las actividades más importantes de cada cargo" (Cortes, 2006). La figura 1 muestra un ejemplo de organigrama.

Figura 1: Organigrama de la estructura organizacional de una empresa.



Finalidad de la estructura organizacional

Debe reflejar en forma esquemática la descripción de las unidades que la integran, su respectiva relación, niveles jerárquicos y canales formales de comunicación. Representa los aspectos fundamentales de una organización, y permite entender un esquema general, así como el grado de diferenciación e integración funcional de los elementos que la componen. La **estructura organizacional** tiene la finalidad de:

- Indicar la relación de jerarquía que guardan entre sí los principales órganos que integran una dependencia o entidad.
- Facilitar al personal el conocimiento de su ubicación y relaciones dentro de la organización.
- Permitir que la información que se almacene en los centros de datos se agrupe por estructuras, brindando la posibilidad de mostrar al usuario solo la información a la que tiene acceso.
- Representar las diferentes unidades que constituyen la organización con sus respectivos niveles jerárquicos.
- Reflejar los diversos tipos de trabajo, especializados o no, que se realizan en la empresa debidamente asignados por área de responsabilidad o función.

- Además muestra una representación de la división de trabajo, indicando los cargos existentes en la compañía y como estos se agrupan en unidades administrativas.

1.2. Estudio de sistemas informáticos que gestionan las estructuras organizacionales

Entre los sistemas informáticos que dan soporte a la gestión de las estructuras organizacionales de una empresa están los sistemas de planificación de Recursos Humanos (RRHH) y los sistemas de gestión empresarial, también conocidos como ERP o software para la Planificación de Recursos Empresariales, que constituyen un conjunto de aplicaciones que persiguen integrar todas las funciones de una empresa y se encuentran entre las más solicitadas del mercado.

Escoger convenientemente la aplicación informática que gestione y planifique las estructuras organizacionales es una decisión compleja, sobre todo por la amplia gama de soluciones informáticas que pueden realizar estas labores, a continuación se describen algunos sistemas informáticos que cuentan con un módulo encargado de estas funciones:

1.2.1. Sauxe

"Es un marco de trabajo desarrollado por el CEIGE para la construcción de aplicaciones web de gestión, que centra su desarrollo en los requerimientos funcionales, las interfaces de usuario y la lógica del negocio" (Torres, 2012). El marco de trabajo brinda funcionalidades de seguridad, auditoría, interoperabilidad, concurrencia, administración de transacciones, entre otras. Posee un módulo llamado Estructura y Composición que permite crear, actualizar y eliminar las estructuras organizacionales de una empresa. Además le brinda la posibilidad de definir la organización jerárquica que tienen los elementos que la componen. Y puede también establecer las estructuras en cada una de las unidades a través de las áreas y definir la plantilla de cargos que está asociada a dicha unidad, así como lo puestos de trabajo.

1.2.2. SAP

Software desarrollado por SAP AG, empresa multinacional alemana, consiste en un conjunto de programas que permiten a las empresas ejecutar y optimizar distintos aspectos como los sistemas de ventas, finanzas, operaciones bancarias, compras, fabricación, inventarios y relaciones con los clientes. Incorpora un módulo de Gestión Organizacional por medio del subsistema de Recursos Humanos, el cual permite definir

la estructura organizacional y funcional de una empresa, a través de un plan organizacional. Una de las principales características de dicho módulo es la administración de la seguridad, permite reflejar la forma en que se organiza la empresa mediante la estructura organizacional, en un inicio el módulo crea las estructuras organizacionales por separado y se relacionan entre sí posteriormente. El plan organizacional es el conjunto de información que describe la estructura organizacional y el ambiente funcional de la empresa. Entre los objetivos principales de este módulo está el diseñar y desarrollar el plan organizacional, encargado de integrar la administración de usuarios a la estructura organizacional de la empresa. Esto trae consigo ventajas como mayor flexibilidad al representar complejas estructuras de grupos de empresas y posibilita las modificaciones en dichas estructuras (Cortes, 2006).

1.2.3. Sistema Humano

Humano es sistema informático de la empresa colombiana SOPORTE LÓGICO Ltda. Es una empresa desarrolladora, integradora e implantadora de Tecnología Informática (TI), además presta servicios de soporte, asesoría y consultoría gerencial para su implantación y ayuda a compañías de todo nivel optimizando así su desempeño en los negocios. Cuenta con un módulo Estructura Organizacional, que administra toda la información relacionada con la estructura u organización de la entidad. El módulo permite manejar los cambios que haya tenido la entidad, así como también los cargos propios de la empresa. Además permite tener un manejo de los perfiles de los cargos con una especificación interna, es decir, el control de la información relacionada con las características de cada uno de los cargos (SOPORTE LÓGICO Ltda., 2010).

Después de realizar el estudio de los sistemas informáticos anteriores, se obtuvo como resultado que cada uno de ellos gestiona las estructuras organizacionales de forma diferente, las principales limitantes de Sistema Humano y SAP son fundamentalmente que no están pensados para su despliegue en empresas cubanas, debido a que son herramientas privativas y no siguen los principios de independencia tecnológica por el cual el país aboga, además de no estar enfocadas a entornos multidominios. Se decide que el marco de trabajo Sauxe es el más apropiado para tomar como punto de referencia en la presente investigación. El mismo posee un módulo de Estructura y Composición que ofrece las funcionalidades necesarias para definir los niveles estructurales de una empresa y la relación entre estos, permite gestionar las estructuras organizacionales y los cargos asociados a dichas estructuras, también gestiona la configuración de los nomencladores generales del sistema. Además los procesos definidos por dicho módulo utilizan la lógica de las estructuras organizacionales cubanas. Sin embargo la reutilización de este sistema no es viable, ya que requiere la instalación de las

tecnologías utilizadas para el desarrollo de Sauxe, lo cual trae consigo grandes dificultades en el rendimiento de la solución que se desea desarrollar en cuanto a recursos de memoria, espacio de almacenamiento y tiempos de respuestas de ejecución.

1.3. Metodología de desarrollo de software

La calidad de un producto de software es determinada en gran medida por la calidad del proceso utilizado para desarrollarlo y mantenerlo. La metodología es la base a emplear en el proceso de construcción de software, permite definir las actividades a desarrollar en cada fase por las que transita el mismo. El desarrollo de la solución propuesta se regirá por el modelo de desarrollo de software elaborado para el CEIGE, este modelo está compuesto por 2 fases, una primera fase de estudio preliminar y una segunda fase de desarrollo. Esta última a su vez está formada por una serie de disciplinas que definen una serie de actividades, procesos y artefactos que se generan en el transcurso de una investigación (Centro de Informatización de Gestión de Entidades, 2013).

Este modelo de desarrollo de software permitirá generar los siguientes artefactos que son de gran importancia para el presente trabajo investigativo:

- Glosario de términos.
- Modelo Conceptual.
- Descripción de requisitos.
- Evaluación de requisitos.
- Matriz de trazabilidad.
- Modelo de datos.
- Diagrama de clases de diseño.
- Diseño de casos de prueba.
- Diagrama de componentes.
- Diagrama de despliegue.

1.4. Tecnologías y herramientas de desarrollo

- **Marco de trabajo**

Symfony 2: Es un marco de trabajo diseñado para optimizar el desarrollo de las aplicaciones web basado en el patrón Modelo Vista Controlador. Proporciona varias herramientas y clases encaminadas a reducir el tiempo de desarrollo de una aplicación web compleja. Además, automatiza las tareas más comunes, permitiendo al desarrollador dedicarse por completo a los aspectos específicos de cada aplicación.

Symfony está desarrollado completamente en PHP 5.3. Ha sido probado en numerosos proyectos reales y se utiliza en sitios web de comercio electrónico de primer nivel. Symfony es compatible con la mayoría de los gestores de bases de datos como MySQL, PostgreSQL, Oracle y Microsoft SQL Server. Se puede ejecutar tanto en plataformas *nix (Unix, Linux, etc.) como en plataformas Windows (Rodríguez, 2014).

Doctrine 2.3: Doctrine es un marco de trabajo para PHP que permite trabajar con un esquema de base de datos como si fuese un conjunto de objetos, y no de tablas y registros, está dividido en dos capas principales, la DBAL (*Database Abstraction Layer*) y el mapeado objeto-relacional más conocido por sus siglas en inglés ORM (*Object-Relational Mapping*). Está inspirado en Hibernate, que es uno de los ORM más populares y grandes que existen y brinda una capa de abstracción de la base de datos muy completa. La característica más importante que presenta es que brinda la posibilidad de escribir consultas de base de datos en un lenguaje propio llamado *Doctrine Query Language* (DQL) (Colectivo de autores, 2012).

Extjs 4.2: Es un marco de trabajo escrito en JavaScript para el desarrollo de aplicaciones web interactivas, usa tecnologías AJAX2, DHTML3 y DOM4. Permite realizar completas interfaces de usuario, fáciles de usar, muy parecidas a las conocidas aplicaciones de escritorio, posibilitando a los desarrolladores concentrarse en la funcionalidad de las aplicaciones en vez de en las advertencias técnicas (Colectivo de autores, 2012).

Twig 1.13: Es un motor de plantillas para PHP cuyo objetivo es ofrecer una alternativa flexible, potente y segura. Ha sido creado por los desarrolladores del marco de trabajo Symfony. Su sintaxis se origina a partir de Jinja y plantillas de Django. Es un producto de código abierto y se distribuye bajo licencia BSD. Symfony 2 viene con soporte incluido para Twig como su motor de plantillas por defecto (Pacheco, 2011).

- **Herramienta y lenguaje de modelado**

Visual Paradigm for UML 8.0: Es una herramienta de diseño UML libre y profesional, diseñada para contribuir al desarrollo de software. Soporta los principales estándares como UML, BPMN y XML. Ofrece un completo conjunto de herramientas a los equipos de desarrollo de software, necesarios para la captura de requisitos, la planificación de software, la planificación de pruebas, el modelado de clases y el modelado de datos (Morales, y otros, 2013).

UML 2.1: Es un lenguaje de modelado visual que se usa para especificar, visualizar, construir y documentar artefactos de un sistema de software. Captura decisiones y conocimiento sobre los sistemas que se deben construir. Se usa para entender, diseñar,

hojear, configurar, mantener, y controlar la información sobre tales sistemas. Está pensado para usarse con todos los métodos de desarrollo, etapas del ciclo de vida, dominios de aplicación y medios (Schmuller, 2000).

- **Lenguajes de programación**

Un lenguaje de programación es un lenguaje diseñado para describir el conjunto de acciones consecutivas que un equipo debe ejecutar. Por lo tanto, un lenguaje de programación es un modo práctico para que los seres humanos puedan dar instrucciones a un equipo.

PHP 5.3: Es un lenguaje interpretado de propósito general ampliamente usado, diseñado especialmente para desarrollo web y que puede ser incrustado dentro de código HTML. Generalmente se ejecuta en un servidor web, tomando el código en PHP como su entrada y creando páginas web como salida. Puede ser desplegado en la mayoría de los servidores web y en casi todos los sistemas operativos y plataformas sin costo alguno (Colectivo de autores, 2011).

YAML 1.2: Es un formato para serializar datos que es fácil de procesar por las máquinas, fácil de leer para las personas y fácil de interactuar con los lenguajes de script. Es un lenguaje que permite describir los datos como en XML, pero con una sintaxis mucho más sencilla. Es un formato especialmente útil para describir datos que pueden ser transformados en arreglos simples y asociativos (Eguiluz, 2011).

JavaScript 1.8: Es un lenguaje de programación que permite a los desarrolladores crear acciones en sus páginas web. Gran parte de su programación está centrada en describir objetos, escribir funciones que respondan a movimientos del mouse, aperturas, utilización de teclas, cargas de páginas, entre otros. Permite la programación de pequeños scripts y de programas más grandes orientados a objetos, con funciones y estructuras de datos complejas. Además, pone a disposición del programador todos los elementos que forman la página web para poder acceder a ellos y modificarlos dinámicamente. Gracias a su compatibilidad con la mayoría de los navegadores modernos, es el lenguaje de programación del lado del cliente más utilizado, es soportado por Internet Explorer, Netscape, Opera, Mozilla Firefox, entre otros (Sánchez, 2003).

- **Entorno de desarrollo**

NetBeans 7.4: Es un entorno de desarrollo integrado libre, hecho principalmente para el lenguaje de programación Java, es una herramienta para programadores pensada para escribir, compilar, depurar y ejecutar programas. Está escrito en Java, pero puede servir

para cualquier otro lenguaje de programación. Existe además un número importante de módulos para extenderlo. El IDE NetBeans es un producto libre y gratuito sin restricciones de uso (NetBeans IDE. La alternativa a Eclipse, 2005).

- **Servidor web**

Apache 2.4: Es un servidor web HTTP gratuito de código fuente abierto, potente y que ofrece un servicio estable y sencillo de mantener y configurar. Se ejecuta en varios sistemas operativos, haciéndolo esta característica universal. Permite personalizar la respuesta ante los posibles errores que se puedan dar en el servidor. Apache presenta entre otras características altamente configurables, bases de datos de autenticación y negociado de contenido, pero fue criticado por la falta de una interfaz gráfica que ayude en su configuración (Apache Software Foundation).

- **Navegador web**

Mozilla Firefox 25: Es un navegador web libre, multiplataforma y de código abierto desarrollado para Microsoft Windows, Mac OS X y GNU/Linux, coordinado por la corporación Mozilla y la Fundación Mozilla. Presenta una forma rápida y eficiente de navegar por la web, que permite abrir varias páginas en una misma ventana mediante el empleo de pestañas separadas. Contiene un plugin Firebug que se utiliza para ver los errores de implementación. Su código fuente es software libre, publicado bajo una triple licencia GPL/LGPL/MPL (Comunidad Mozilla Firefox de Cuba).

- **Sistema de gestión de base de datos**

PostgreSQL 9.2: Sistema de gestión de bases de datos. Tiene soporte total para transacciones, disparadores, vistas, procedimientos almacenados, almacenamiento de objetos de gran tamaño. Se destaca en ejecutar consultas complejas, consultas sobre vistas, consultas y joins de gran tamaño. Permite la definición de tipos de datos personalizados e incluye un modelo de seguridad completo. Cuenta con una gran comunidad de desarrollo en Internet, su código fuente está disponible sin costo alguno y es multiplataforma (Fundación de código libre dominicano, 2007).

PgAdmin III 1.16: PgAdmin III es una aplicación gráfica para gestionar el gestor de bases de datos PostgreSQL, siendo la más completa y popular con licencia Open Source. Está escrita en C++ usando la librería gráfica multiplataforma, lo que permite que se pueda usar en Linux, FreeBSD, Solaris, Mac OS X y Windows. Es capaz de gestionar versiones a partir de la PostgreSQL 7.3 ejecutándose en cualquier plataforma, así como versiones comerciales de PostgreSQL. Está diseñado para responder a las necesidades de todos los usuarios, desde escribir consultas SQL simples hasta desarrollar bases de datos

complejas. La interfaz gráfica soporta todas las características de PostgreSQL y facilita enormemente la administración (Colectivo de autores, 2010).

- **Sistema de control de versiones**

SVN (Subversion) 1.8: Es un software de sistema de control de versiones, libre bajo una licencia de tipo Apache/BSD y se le conoce también como svn por ser ese el nombre de la herramienta de línea de comandos. Una característica importante de Subversion es que los archivos versionados no tienen cada uno un número de revisión independiente. En cambio, todo el repositorio tiene un único número de versión que identifica un estado común de todos los archivos del repositorio en cierto punto del tiempo. Subversion puede acceder al repositorio a través de redes, lo que le permite ser usado por personas que se encuentran en distintas computadoras (Küng, y otros, 2013)

RapidSVN 0.12: Es un cliente gráfico multiplataforma de Subversión que entre sus principales rasgos están proporcionar una interfaz fácil de usar para las características de Subversion. Es eficiente y simple para los principiantes pero lo suficientemente flexible como para aumentar la productividad para los usuarios con experiencia. Está completamente escrito en C++ y ha sido traducido a varios idiomas. Posee soporte completo para caracteres Unicode (Küng, y otros, 2013).

- **Sistema operativo**

Ubuntu 14.04 (LTS): Ubuntu es un sistema operativo enfocado en la facilidad de uso e instalación y la libertad de los usuarios. Está basado en Linux y se distribuye como software libre, el cual incluye su propio entorno de escritorio denominado Unity. Su nombre proviene de la ética homónima, en la que se habla de la existencia de uno mismo como cooperación de los demás. Las versiones LTS (Soporte Técnico Extendido), que se liberan cada dos años, reciben soporte durante cinco años en los sistemas de escritorio y de servidor (Comunidad de software libre de la UCI).

1.5. Requisitos de software

Los requisitos son características que se debe exhibir por el software desarrollado para solventar un problema en el mundo real o alcanzar un objetivo. Un rasgo esencial de todos los requisitos del software es que sean comprobables. Son también capacidades que deben estar implícitas en un sistema para satisfacer un contrato, estándar, especificación u otro documento formal.

Los requisitos de software se clasifican en funcionales y no funcionales. Los requisitos funcionales describen qué es lo que el sistema debe ejecutar, se conocen también como

capacidades. Los requisitos no funcionales imponen restricciones de cómo los requisitos funcionales deben ser implementados, son restricciones de los servicios o funciones ofrecidas por el sistema en cuanto a fiabilidad, tiempo de respuesta, capacidad de almacenamiento, entre otros (Comité de la Práctica Profesional del IEEE Computer Society, 2004).

1.6. Técnicas de captura de requisitos

La captura de requisitos es la actividad mediante la cual se extraen las necesidades del sistema, es el comienzo de cada ciclo de desarrollo. Las técnicas de captura son las que posibilitan que la extracción de los requisitos sea efectiva y sirven de base para verificar si se alcanzaron los objetivos establecidos en la investigación. Seguidamente se describen las técnicas que se utilizan en el presente trabajo:

Entrevistas: Es un método clásico que posibilita tomar conocimiento del problema y comprender los objetivos de la solución propuesta. Mediante esta técnica el equipo de desarrollo se acerca al problema de una forma natural consultando al cliente.

Sistemas existentes: Consiste en analizar y estudiar los sistemas existentes que estén relacionados con la solución del sistema que va a ser construido, analizándose las interfaces de usuarios y el comportamiento que el mismo produce.

Observación: Con esta técnica se busca profundizar sobre lo que realmente se está haciendo manual o automáticamente en el sistema actual. La información que se obtiene a través de la observación incluye las actividades que realizan y todo aquello que es susceptible de mejora.

1.7. Técnicas de validación de requisitos

La validación de requisitos permite demostrar que la descripción de requisitos define realmente la solución que el usuario necesita. Su objetivo es verificar todos los requisitos obtenidos para asegurarse que presentan una descripción correcta del sistema a implementar. A continuación se detallan las técnicas que se emplearán para validar los requisitos funcionales:

Revisión técnica formal: Esta técnica consiste en la lectura y corrección de la documentación de requisitos. De esta forma se puede validar la correcta interpretación de la información transmitida.

Matriz de Trazabilidad: Esta técnica posibilita marcar los objetivos del sistema y verificarlos contra los requisitos del mismo. Se hace necesario chequear qué objetivos

cubre cada requisito, de esta manera se pueden detectar inconsistencias u objetivos no cubiertos.

Prototipos de interfaz de usuario: Existen propuestas que se basan en obtener la descripción de requisitos en prototipos, estas permiten al cliente hacerse una idea de la estructura de la interfaz del sistema. Esta técnica tiene el problema de que el usuario debe entender que lo que está viendo es un prototipo y no el sistema final.

1.8. Patrones de diseño

Según Pressman un patrón de diseño es una estructura de diseño que resuelve un problema de diseño particular dentro de un contexto específico y en medio de "fuerzas" que pueden tener un impacto en la manera en que se aplica y utiliza el patrón.

Los patrones de diseño son propuestas generales planteadas en término de una colaboración de objetos mediante las que se resuelven una gran cantidad de problemas que aparecen una y otra vez en el desarrollo de aplicaciones informáticas, se consideran una serie de buenas prácticas de aplicación recomendable en el diseño de software. Existen varias clasificaciones para los patrones de diseño, entre las principales se encuentran:

1.8.1. Patrones GRASP

Los patrones GRASP del inglés (*General Responsibility Assignment Software Patterns*) se encargan de asignar responsabilidades a los objetos en sentido general. Las principales responsabilidades son conocer los atributos y las relaciones con otros objetos y realizar las tareas que debe cumplir cada objeto. Existen nueve patrones GRASP pero a continuación se mencionan y describen los que se utilizarán durante el desarrollo de la investigación (Larman, 1999):

- **Experto**: El patrón experto en información es el principio básico de asignación de responsabilidades. Indica que la responsabilidad de la creación de un objeto o la implementación de un método, debe recaer sobre la clase que conoce toda la información necesaria para crearlo. De este modo se obtiene un diseño con mayor cohesión y así la información se mantiene encapsulada (disminución del acoplamiento).
- **Creador**: El patrón creador ayuda a identificar quién debe ser el responsable de la creación o instanciación de nuevos objetos o clases. La nueva instancia deberá ser creada por la clase que tiene la información necesaria para realizar la creación del objeto, o usar directamente las instancias creadas del objeto.

- **Controlador**: El patrón controlador es un patrón que sirve como intermediario entre una determinada interfaz y el algoritmo que la implementa, de tal forma que es la que recibe los datos del usuario y la que los envía a las distintas clases según el método llamado. Este patrón sugiere que la lógica de negocios debe estar separada de la capa de presentación, esto para aumentar la reutilización de código y a la vez tener un mayor control. Se recomienda dividir los eventos del sistema en el mayor número de controladores para poder aumentar la cohesión y disminuir el acoplamiento.
- **Alta Cohesión**: Este patrón determina, que la información almacenada en una clase debe ser coherente y estar relacionada con esta, en mayor medida y enfocada en sus responsabilidades. Al realizar un diseño donde las clases del componente mantengan una alta cohesión como por ejemplo las clases controladoras, es posible ganar en claridad y facilidad a la hora de entender el diseño, además de simplificar el mantenimiento y soportar mayor capacidad de reutilización.
- **Bajo acoplamiento**: Consiste en tener las clases lo menos relacionadas entre sí, para que en caso de producirse una modificación en alguna de ellas, se tenga la mínima repercusión en las demás, potenciando la reutilización y disminuyendo la dependencia entre las clases (Larman, 1999).

1.8.2. Patrones GoF

Los patrones *Gang Of Four* (GoF), son patrones de diseño publicados en el libro *Design Patterns: Elements of Reusable Object-Oriented Software* por Gamma, Helm, Johnson, Vlissides, conocidos por Banda de los cuatro. Están divididos fundamentalmente en tres grandes grupos:

- *Creacionales*: Concierne al proceso de creación de objetos.
- *Estructurales*: Tratan la composición de clases y objetos.
- *Comportamiento*: Caracterizan las formas en las que interactúan y reparten responsabilidades las distintas clases u objetos.

A continuación se describen algunos que podemos encontrar en Symfony 2:

- **Estrategia**: El patrón de estrategia encapsula algoritmos de la misma naturaleza en clases dedicadas a ser intercambiables.
- **Inicialización retardada**: El un patrón que retrasa la creación de un objeto, ejecución del cálculo de algún valor, o alguna operación costosa hasta que sea realmente necesaria por primera vez.

- **Composite**: El patrón Composite sirve para construir objetos complejos a partir de otros más simples y similares entre sí, gracias a la composición recursiva y a una estructura en forma de árbol. Esto simplifica el tratamiento de los objetos creados, ya que al poseer todos ellos una interfaz común, se tratan todos de la misma manera.
- **Iterator**: Define una interfaz que declara los métodos necesarios para acceder secuencialmente a un grupo de objetos de una colección.
- **Mediator**: Este patrón define un objeto que encapsula como un conjunto de objetos interactúan.
- **Fachada**: Es un patrón estructural que define una interfaz de alto nivel que hace que el componente sea más fácil de usar. Propone la definición de un único punto de conexión con un componente, este objeto fachada presenta una interfaz unificada y es responsable de colaborar con otros componentes.
- **Command**: Encapsula una operación en un objeto, permitiendo ejecutar dicha operación sin necesidad de conocer el contenido de la misma.
- **Decorator**: Añade funcionalidad a una clase dinámicamente.
- **Registry**: Este patrón es muy útil para los desarrolladores en la Programación Orientada a Objetos. Este patrón es un medio sencillo y eficiente de compartir datos y objetos en la aplicación sin la necesidad de preocuparse por conservar numerosos parámetros o hacer uso de variables globales (Gamma, et al., 1995).

1.9. Arquitectura de software

La arquitectura representa un modelo relativamente pequeño, intelectualmente tratable, de la forma en que un sistema se estructura y sus componentes se entienden entre sí; este modelo es transferible a través de sistemas; en particular, se puede aplicar a otros sistemas que exhiben requerimientos. El diseño arquitectónico soporta reutilización de grandes componentes o incluso de framework en los que se pueden integrar componentes (Reynoso, 2004).

Paul Clements, plantea que la arquitectura de software es, a grandes rasgos, una vista general del sistema que incluye los componentes principales, la conducta de estos componentes con el resto del sistema y las formas en que deben interactuar y coordinarse para alcanzar la misión del sistema (Clements, 1996).

Según Pressman, la arquitectura de software es la representación que capacita al ingeniero del software para: analizar la efectividad del diseño para la consecución de los

requisitos fijados, considerar las alternativas arquitectónicas en una etapa en la cual hacer cambios en el diseño es relativamente fácil, y reducir los riesgos asociados a la construcción del software (Pressman, 2010). Sin embargo, para este trabajo se utilizará la definición oficial establecida por la IEEE: "La Arquitectura de Software es la organización fundamental de un sistema encarnada en sus componentes, las relaciones entre ellos y el ambiente y los principios que orientan su diseño y evolución".

1.10. Conclusiones parciales

En este capítulo se definieron una serie de conceptos necesarios para el entendimiento correcto del problema tratado en el presente trabajo y se realizó un estudio de las herramientas que gestionan las estructuras organizacionales de una empresa, enfatizando sus características y funcionalidades. Se decidió tomar al marco de trabajo Sauxe como basamento y guía para las principales funcionalidades con las que debe cumplir la propuesta de solución. Por lo antes mencionado se determinó que es necesario el desarrollo de un módulo de Estructura y Composición que se integre con el marco de trabajo Symfony 2 y permita que este soporte la gestión de estructuras organizacionales en entornos multidominios. Para ello se seleccionó una metodología para regir el proceso de desarrollo de software y se propusieron las herramientas y tecnologías que se utilizarán para llevar a cabo el desarrollo de la solución.

2. CAPÍTULO 2. CARACTERÍSTICAS DEL SISTEMA

El presente capítulo acumula los resultados obtenidos durante el proceso de desarrollo de la solución, así como algunos de los artefactos generados por el mismo. Se describen las técnicas de captura requisitos empleadas y los requisitos funcionales de la solución para lograr un entendimiento de lo que se quiere lograr. Posteriormente se realiza una descripción del diseño elaborado para alcanzar el objetivo propuesto, además de la estrategia a seguir para la implementación. Es necesario que se traten puntos claves del desarrollo como la arquitectura de la aplicación, patrones de diseño utilizados, algunos artefactos que propone la metodología seleccionada y el diseño de clases del sistema.

2.1. Descripción de la solución propuesta

El sistema informático a desarrollar en Symfony 2 utilizando el marco de trabajo Extjs para la capa de presentación, debe permitir el soporte a la gestión de estructuras organizacionales en entornos multidominios, para ello el nuevo módulo llamado Estructura y Composición será encargado de llevar cabo esta labor. Entre las responsabilidades principales de este módulo se encuentran, brindar la posibilidad al cliente de definir y gestionar la estructura organizacional en la cual se va a ubicar su empresa y la estructura dentro de dicha empresa, así como permitir que se puedan especificar los cargos por los cuales van a estar compuestas las diferentes áreas dentro de las unidades. También se permitirá especificar el nivel jerárquico que existe entre las entidades que se estén gestionando, es decir, a quien se subordina las mismas.

Este trabajo investigativo persigue que el módulo a desarrollar pueda ser reutilizable y se integre con otros módulos, como el de Seguridad y el de Gestión de Trazas, mediante la implementación de interfaces que brinden o soliciten servicios específicos. Además, el módulo desarrollado necesita ser configurable para que los usuarios puedan crear otros tipos de conceptos estructurales.

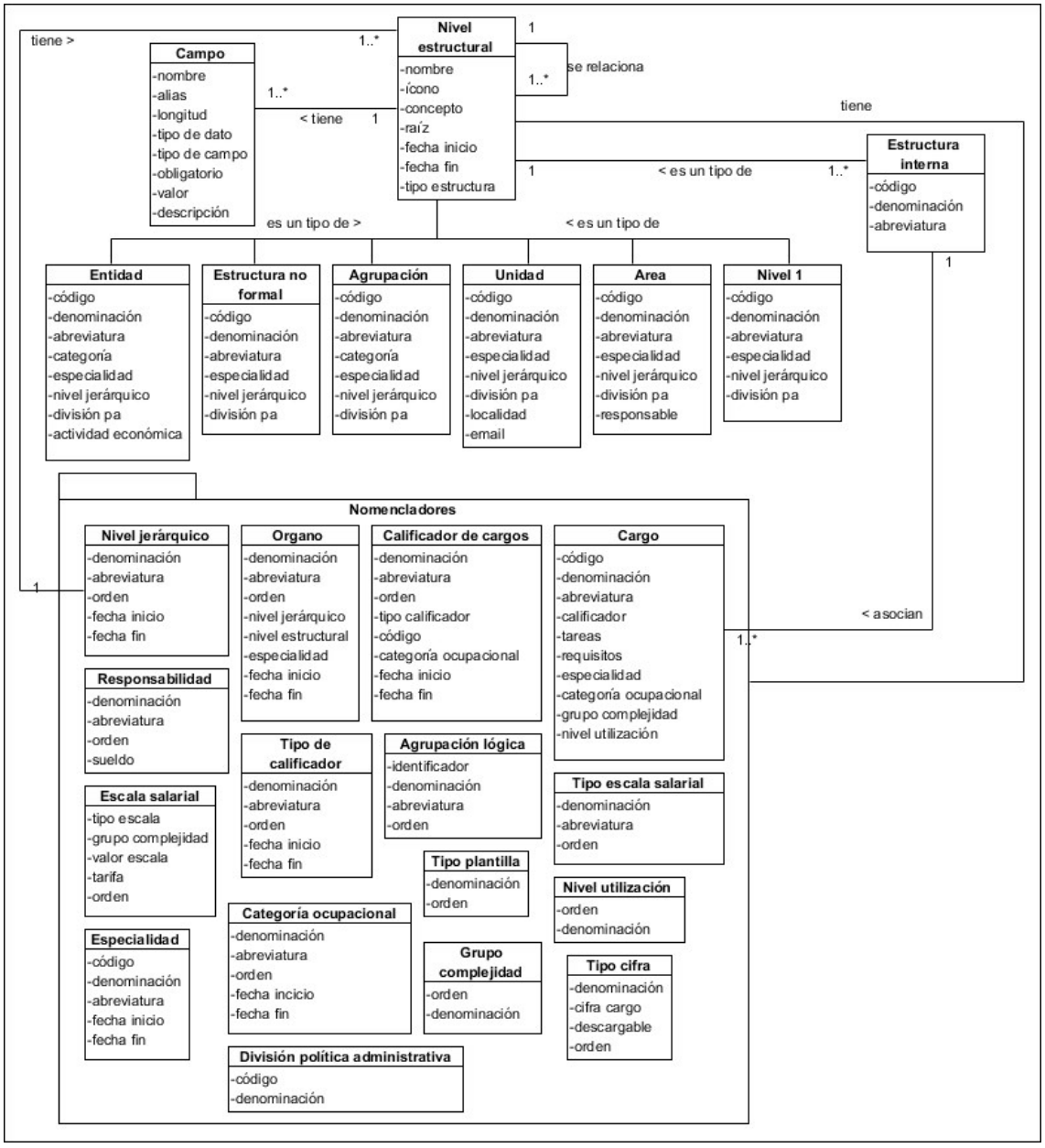
2.2. Modelado del negocio

Antes de identificar los requisitos funcionales es necesario conocer el dominio del problema de la investigación y el contexto organizacional y operacional, es decir la situación actual. El modelado del negocio constituye una actividad fundamental para lograr una buena comprensión de las actividades que se llevan a cabo dentro de la organización, de esta forma se conocen las necesidades del cliente.

2.2.1. Modelo Conceptual

La figura 2 muestra el Modelo Conceptual que relaciona los principales conceptos identificados en la investigación.

Figura 2: Modelo Conceptual del módulo Estructura y Composición.



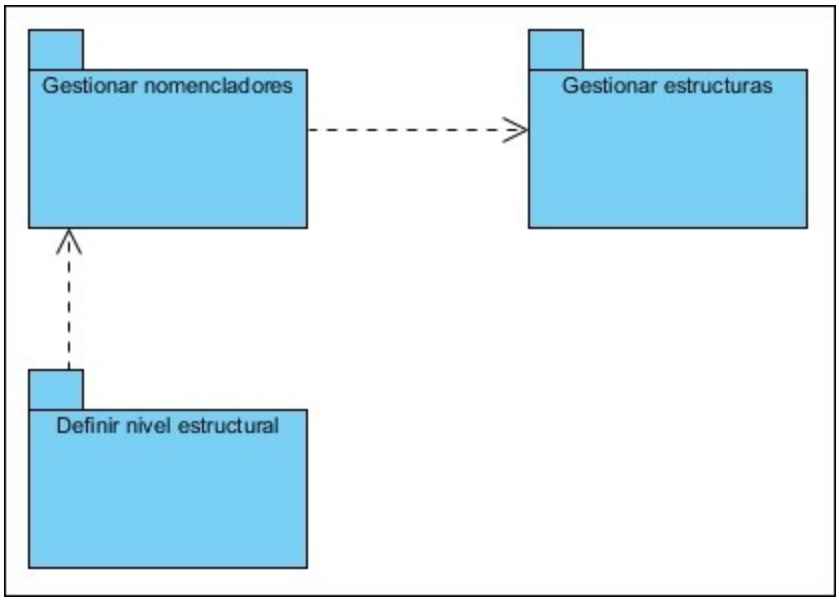
El Modelo conceptual es una representación de conceptos del mundo real. Este artefacto se crea con el objetivo de aumentar la comprensión del problema y contribuir a esclarecer la terminología usada. En realidad es un modelo que comunica a los involucrados, cuáles son los términos importantes y cómo se relacionan entre sí. Rigiéndose por lo establecido en la disciplina Modelado de negocio del modelo de desarrollo de software, se identificaron 26 conceptos fundamentales.

El modelo conceptual de la figura 2 ejemplifica que un nivel estructural tiene un nivel jerárquico, puede contener uno o más niveles estructurales, y poseen varios campos asociados. La estructura interna es un tipo de estructura que se le asocia un conjunto de cargos determinados. Existen otros tipos de estructura que estarán presentes en la solución propuesta de manera predefinida, esto se realiza con el objetivo de brindar la posibilidad de reutilizarlas, ya que las mismas se adaptan a las especificidades de las estructuras organizacionales cubanas, siendo, ejemplos de estas estructuras las representadas por los conceptos: Entidad, Agrupación, Nivel 1, Unidad, Estructura no formal y Área. Además se muestra un conjunto de nomencladores con los que se trabajará durante el desarrollo del sistema, estos se encuentran contenidos en un paquete para un mejor entendimiento y comprensión del modelo conceptual, los mismos pueden formar parte de una estructura organizacional en dependencia de sus características.

2.2.2. Agrupación de requisitos por paquetes

Para un mejor entendimiento del módulo Estructura y Composición se muestran los requisitos agrupados por funcionalidad en los paquetes definidos (figura 3), con el propósito de lograr una mejor organización y comprensión de dichos requisitos en el diseño de la solución. Los cuales especifican cómo se gestionan los niveles estructurales, las estructuras e información que persistirá en los nomencladores.

Figura 3: Agrupación funcional de requisitos por paquetes.



2.2.3. Descripción de requisitos funcionales

Con la identificación y la descripción de los requisitos de software se pretende establecer y mantener un acuerdo con los clientes y otros involucrados en lo que el sistema debe

hacer. Permite definir la interfaz de usuario del sistema enfocándose en las necesidades y aspiraciones de los usuarios.

Mediante las técnicas de captura de requisitos se entrevistaron a desarrolladores y analistas del centro CEIGE que estaban involucrados directamente con el desarrollo del módulo Estructura y Composición que tiene Sauxe, para constatar en qué consistía el funcionamiento del mismo y luego modelar el diseño de la solución. Se estudiaron herramientas relacionadas con el objeto de estudio, con el objetivo de identificar potenciales funcionalidades y aspectos nuevos que pudieran concebirse en el nuevo sistema.

Finalmente se obtuvo un total de 108 requisitos funcionales recogidos en 27 agrupaciones. La tabla 1 muestra como están organizados por agrupación y funcionalidad los requisitos del software a construir:

Tabla 1: Muestra los requisitos funcionales por agrupación.

Definir nivel estructural	
Gestionar nivel estructural	Adicionar nivel estructural.
	Modificar nivel estructural.
	Eliminar nivel estructural.
	Listar niveles estructurales.
Gestionar relación de niveles estructurales	Adicionar relación de niveles estructurales.
	Eliminar relación de niveles estructurales.
	Listar relación de niveles estructurales.
Gestionar campos	Adicionar Campo.
	Modificar Campo.
	Eliminar Campo.
	Listar Campos.
Gestionar estructuras	
Gestionar Nivel 1	Adicionar Nivel 1.
	Modificar Nivel 1.
	Eliminar Nivel 1.
Gestionar Agrupación	Adicionar Agrupación.
	Modificar Agrupación.
	Eliminar Agrupación.

Gestionar Entidad	Adicionar Entidad.
	Modificar Entidad.
	Eliminar Entidad.
Gestionar Unidad	Adicionar Unidad.
	Modificar Unidad.
	Eliminar Unidad.
Gestionar estructura externa	Adicionar estructura externa.
	Modificar estructura externa.
	Eliminar estructura externa.
Gestionar Área	Adicionar Área.
	Modificar Área.
	Eliminar Área.
Gestionar estructura no formal	Adicionar estructura no formal.
	Modificar estructura no formal.
	Eliminar estructura no formal.
Gestionar estructura interna	Adicionar estructura interna.
	Modificar estructura interna.
	Eliminar estructura interna.
Gestionar cargos en estructura interna	Adicionar cargo en estructura interna.
	Modificar cargo en estructura interna.
	Eliminar cargo en estructura interna.
Gestionar nomencladores	
Gestionar Agrupación lógica	Adicionar Agrupación lógica.
	Modificar Agrupación lógica.
	Eliminar Agrupación lógica.
	Buscar Agrupación lógica.
	Listar Agrupaciones lógicas.
Gestionar Cargo	Adicionar Cargo.
	Modificar Cargo.
	Eliminar Cargo.
	Buscar Cargo.
	Listar Cargos.
Gestionar Escala salarial	Adicionar Escala salarial.
	Modificar Escala salarial.

	Eliminar Escala salarial.
	Buscar Escala salarial.
	Listar Escalas salariales.
Gestionar Calificador de cargos	Adicionar Calificador de cargo.
	Modificar Calificador de cargo.
	Eliminar Calificador de cargo.
	Buscar Calificador de cargo.
	Listar Calificadores de cargo.
Gestionar Categoría ocupacional	Adicionar Categoría ocupacional.
	Modificar Categoría ocupacional.
	Eliminar Categoría ocupacional.
	Buscar Categoría ocupacional.
	Listar Categorías ocupacionales.
Gestionar Órgano	Adicionar Órgano.
	Modificar Órgano.
	Eliminar Órgano.
	Buscar Órgano.
	Listar Órgano.
Gestionar Nivel jerárquico	Adicionar Nivel jerárquico.
	Modificar Nivel jerárquico.
	Eliminar Nivel jerárquico.
	Buscar Nivel jerárquico.
	Listar Niveles jerárquicos.
Gestionar Tipo de calificador	Adicionar Tipo de calificador.
	Modificar Tipo de calificador.
	Eliminar Tipo de calificador.
	Buscar Tipo de calificador.
	Listar Tipos de calificador.
Gestionar Grupo de complejidad	Adicionar Grupo de complejidad.
	Modificar Grupo de complejidad.
	Eliminar Grupo de complejidad.
	Buscar Grupo de complejidad.
	Listar Grupo de complejidad.
Gestionar Nivel de utilización	Adicionar Nivel de utilización.

	Modificar Nivel de utilización.
	Eliminar Nivel de utilización.
	Buscar Nivel de utilización.
	Listar Niveles de utilización.
Gestionar Responsabilidad	Adicionar Responsabilidad.
	Modificar Responsabilidad.
	Eliminar Responsabilidad.
	Buscar Responsabilidad.
	Listar Responsabilidades.
Gestionar Tipo de cifra	Adicionar Tipo de cifra.
	Modificar Tipo de cifra.
	Eliminar Tipo de cifra.
	Buscar Tipo de cifra.
	Listar Tipos de cifra.
Gestionar Tipo de escala salarial	Adicionar Tipo de escala salarial.
	Modificar Tipo de escala salarial.
	Eliminar Tipo de escala salarial.
	Buscar Tipo de escala salarial.
	Listar Tipos de escala salarial.
Gestionar Tipo de plantilla	Adicionar Tipo de plantilla.
	Modificar Tipo de plantilla.
	Eliminar Tipo de plantilla.
	Buscar Tipo de plantilla.
	Listar Tipos de plantilla.

Como parte de las actividades que propone el subproceso Administración de requisitos se generó el artefacto Evaluación de requisitos que se encuentra en el expediente de proyecto, a través del cual se obtuvo como resultado la complejidad y prioridad de los requisitos funcionales.

2.2.4. Especificación de requisitos funcionales

En las especificaciones de requisitos son plasmadas las características y condiciones precisas que debe cumplir cada requisito funcional. En la tabla 2 y 3 se presenta la especificación de algunos requisitos funcionales, los restantes se pueden consultar en los anexos:

Tabla 2: Especificación del requisito Adicionar nivel estructural.

Precondiciones	El usuario se ha identificado y autenticado ante el sistema.
Flujo de eventos	
Flujo básico	
1	Se introducen los datos del nivel estructural: Nombre. Icono. Concepto. Raíz. Tipo de estructura (Interna, Externa). Fecha inicio. Fecha fin.
2	El sistema valida (ver validación 1) los datos introducidos.
3	Si los datos son correctos el sistema los registra.
4	El sistema confirma el registro de los datos.
5	Concluye el requisito.
Post-condiciones	
1	Se registró en el sistema un nuevo nivel estructural.
2	El sistema muestra un listado actualizado de los niveles estructurales existentes. (Ver requisito-Listar niveles estructurales).
Flujos alternativos	
Flujo alternativo 3.a Información errónea	
1	El sistema señala los datos erróneos y permite corregirlos.
2	El usuario corrige los datos.
3	Volver al paso 2 del flujo básico.
Post-condiciones	
1	N/A
Flujo alternativo 3.b Información incompleta	
1	El sistema señala los datos vacíos y permite corregirlos.
2	El usuario corrige los datos.
3	Volver al paso 2 del flujo básico.
Post-condiciones	
1	N/A
Flujo alternativo *.a El usuario cancela la acción	
1	Concluye el requisito.

Post-condiciones		
1	No se registran los datos.	
Validaciones		
1	Se validan los datos según lo establecido en el Modelo Conceptual CIG-ERP-N-EC-i1301.	
Conceptos	**Nivel estructural**	Visibles en la interfaz: Nombre. Icono. Concepto. Raíz. Tipo de estructura (Interna, Externa). Fecha inicio. Fecha fin. Utilizados internamente: N/A.
Requisitos especiales	N/A.	
Asuntos pendientes	N/A.	

Figura 4: Muestra el prototipo de interfaz del RF Adicionar nivel estructural.

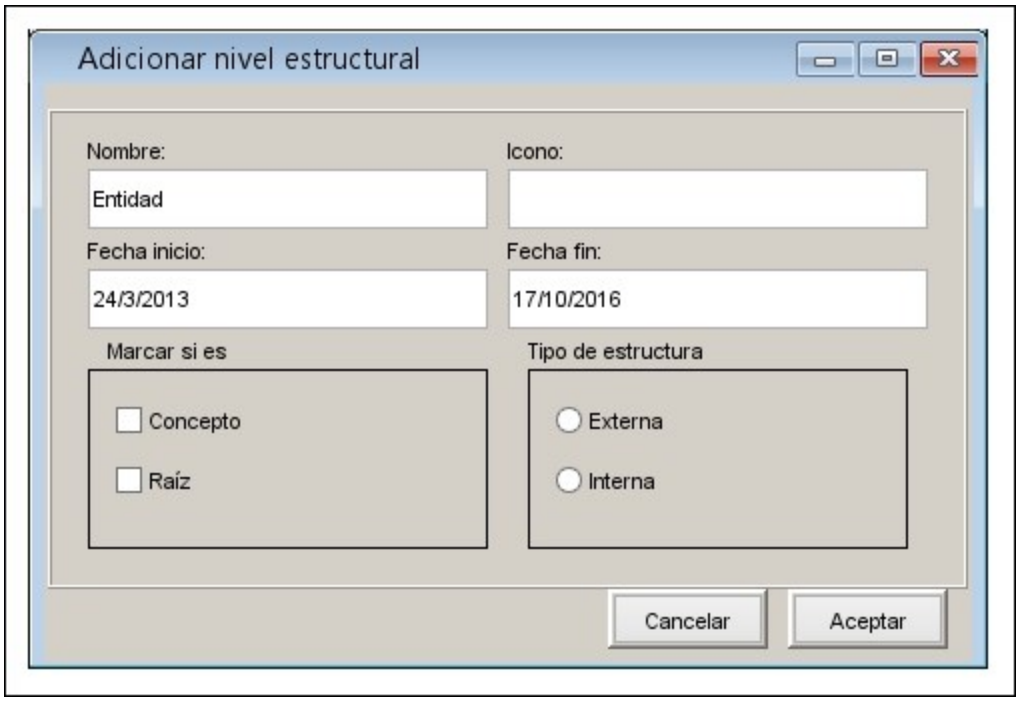


Tabla 3: Especificación del requisito Adicionar campo.

Precondiciones	Se ha creado al menos un nivel estructural. El usuario se ha identificado y autenticado ante el sistema.
Flujo de eventos	
Flujo básico	
1	Seleccionar el nivel estructural al cual se le desea adicionar el campo.
2	Se introducen los datos del campo: Nombre Alias Longitud Tipo de dato Tipo de campo Obligatorio (si o no) Valor por defecto Descripción
3	El sistema valida (ver validación 1) los datos introducidos.
4	Si los datos son correctos el sistema los registra.
5	El sistema confirma el registro de los datos.
6	Concluye el requisito.
Post-condiciones	
1	Se registró en el sistema un nuevo campo.
2	Se asoció el campo creado a un nivel estructural.
3	El sistema muestra un listado actualizado de los campos existentes. (Ver requisito-Listar campos).
Flujos alternativos	
Flujo alternativo 4.a Información errónea	
1	El sistema señala los datos erróneos y permite corregirlos.
2	El usuario corrige los datos.
3	Volver al paso 3 del flujo básico.
Post-condiciones	
1	N/A
Flujo alternativo 4.b Información incompleta	
1	El sistema señala los datos vacíos y permite corregirlos.
2	El usuario corrige los datos.
3	Volver al paso 3 del flujo básico.

Post-condiciones		
1	N/A	
Flujo alternativo *.a El usuario cancela la acción		
1	Concluye el requisito.	
Post-condiciones		
1	No se registran los datos.	
Validaciones		
1	Se validan los datos según lo establecido en el Modelo Conceptual CIG-CDX-N-EC-i1301	
	Extensiones	N/A.
Conceptos	**Campo**	Visibles en la interfaz: Nombre Alias Longitud Tipo de dato Tipo de campo Obligatorio (si o no) Valor por defecto Descripción Utilizados internamente: N/A
Requisitos especiales	N/A.	
Asuntos pendientes	N/A.	

Figura 5: Muestra el prototipo de interfaz del RF Adicionar campo.

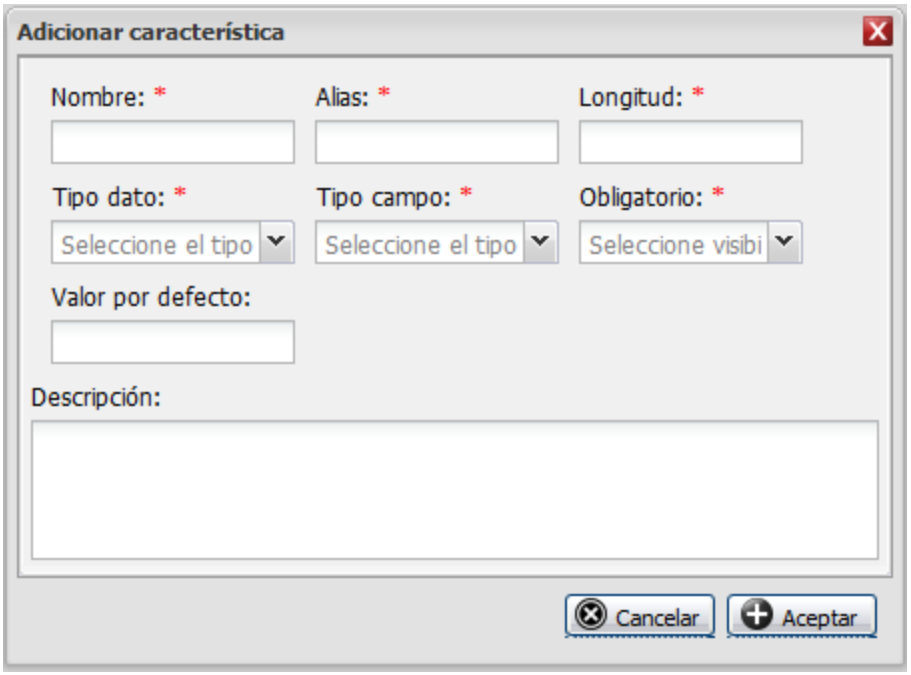


Tabla 4: Especificación del requisito Adicionar cargo en estructura interna.

Precondiciones	El usuario se ha identificado y autenticado en el sistema. Se debe haber creado el nivel estructural que sea de tipo Estructura interna. Se debe haber adicionado los campos necesarios de este nivel estructural. Se debe haber adicionado un órgano al nomenclador de Órgano. Se debe haber adicionado una especialidad al nomenclador de Especialidad. Se debe haber adicionado un nivel jerárquico al nomenclador Nivel Jerárquico. Se debe haber adicionado una división política administrativa al nomenclador de División P.A. Debe existir al menos un nivel estructural de tipo Nivel 1 al cual subordinar la Estructura interna. Se ha mostrado un listado de las estructuras existentes. El nomenclador Cargo debe tener datos.
Flujo de eventos	
Flujo básico	
6	Se selecciona la Estructura interna a la cual se le desea adicionar un cargo.
7	Se listan los cargos existentes.
8	Se selecciona la opción adicionar Cargo.
9	Se introducen los datos del Cargo: Denominación del cargo. Categoría ocupacional. Grupo de Complejidad. Tipo de escala salarial. Responsabilidad del cargo. Tipo de plantilla. Escala salarial. Especialidad. Tipo de Cifra. Fecha inicio. Fecha fin. Cantidad.

	Orden. Cantidad en tiempo de guerra (CTG). Modificable (Si, No).
10	El sistema valida (ver validación 1) los datos introducidos.
11	Si los datos son correctos el sistema los registra.
12	El sistema confirma el registro de los datos.
13	Concluye el requisito.
Post-condiciones	
3	Se le asoció a una Estructura interna un cargo en el sistema.
Flujos alternativos	
Flujo alternativo 5.a Información errónea	
4	El sistema señala los datos erróneos y permite corregirlos.
5	El usuario corrige los datos.
6	Volver al paso 4 del flujo básico.
Post-condiciones	
2	N/A
Flujo alternativo 5.b Información incompleta	
4	El sistema señala los datos vacíos y permite corregirlos.
5	El usuario corrige los datos.
6	Volver al paso 4 del flujo básico.
Post-condiciones	
2	N/A
Flujo alternativo *.a El usuario cancela la acción	
2	Concluye el requisito.
Post-condiciones	
2	No se registran los datos.
Validaciones	
2	Se validan los datos según lo establecido en el Modelo conceptual CIG-ERP-N-EC-i1301.
Conceptos	**Categoría ocupacional** — Visibles en la interfaz: Denominación Utilizados internamente: N/A.

	Grupo de complejidad	Visibles en la interfaz: Denominación Utilizados internamente: N/A.
	Calificador de cargos	Visibles en la interfaz: Denominación Utilizados internamente: N/A.
	Nivel de utilización del cargo	Visibles en la interfaz: Denominación Utilizados internamente: N/A.
Requisitos especiales	N/A.	
Asuntos pendientes	N/A.	

Figura 6: Muestra el prototipo de interfaz del RF Adicionar cargo en estructura interna.

2.2.5. Descripción de requisitos no funcionales

Los resultados del presente trabajo investigativo formarán parte de un nuevo marco de trabajo desarrollado en Symfony 2, por lo tanto los requisitos no funcionales de la solución propuesta se acogerán a los establecidos por el centro CEIGE. Inmediatamente se describen algunos de los más significativos.

Usabilidad: El sistema podrá ser usado por cualquier persona que posea conocimientos básicos en el manejo de la computadora.

Eficiencia: Los tiempos de respuesta y velocidad de procesamiento de la información serán rápidos, no mayores de 5 segundos para las actualizaciones y 20 para las recuperaciones.

Seguridad: Protección contra acciones no autorizadas o que puedan afectar la integridad de los datos. Verificación sobre las acciones irreversibles como las eliminaciones.

Portabilidad: El sistema debe ser multiplataforma.

2.3. Modelado del diseño

El Modelo de diseño es utilizado como entrada esencial en las actividades relacionadas con la implementación del sistema. El mismo contiene diagramas, clases, paquetes, componentes, interfaces, relaciones, colaboraciones y atributos. Dentro de este epígrafe se verán los elementos que se tuvieron en cuenta durante el diseño de la solución, y se obtendrán como artefactos los diagramas de clases del análisis y diseño. De esta manera se define la línea de trabajo para sentar las bases de la implementación del sistema.

2.3.1. Patrones de diseño

Los patrones de diseño son la base para la búsqueda de soluciones a problemas comunes en el desarrollo de software, por lo que es de suma importancia aplicarlos en la construcción del diseño de un sistema. Con su uso, se pretende establecer un lenguaje común entre los programadores, contribuir a la reutilización, ahorrar tiempo en la implementación y obtener un producto con calidad. Son además la solución a determinado problema de diseño que se presente en el desarrollo de un sistema. Entre sus características se debe incluir la habilidad de ser reutilizable y aplicable a diferentes problemas de diseño en distintas circunstancias. A continuación se describe como se manifiesta la utilización de los patrones de diseño para este trabajo investigativo.

2.3.1.1. Patrones GRASP

Una de las principales ventajas de los marcos de trabajo de desarrollo es que están basados en patrones de diseño, característica que hace posible la gran usabilidad que tienen, casi siempre independientes del tipo de aplicación web que se desea desarrollar. El uso de Symfony 2, evidencia el uso de varios patrones de diseño que este trae incluidos por defecto en su arquitectura, además de estar concebidos de tal manera que obliga al programador a aplicarlos.

- **Experto**: Es uno de los patrones que más se utiliza cuando se trabaja con Symfony 2, con la inclusión de Doctrine para mapear la base de datos. Symfony 2 utiliza este ORM para implementar su capa de abstracción en el modelo, encapsular toda la lógica de los datos y generar las clases con todas las funcionalidades comunes de las entidades, las clases de abstracción de datos poseen un grupo de funcionalidades que están relacionadas directamente con la entidad y contienen la información necesaria de la tabla que representan.
- **Creador**: En las clases controladoras del bundle Estructura y Composición se encuentran las funcionalidades con el sufijo “action”, aquí se encuentran las acciones definidas para el sistema y se ejecutan en cada una de ellas. En dichas acciones se crean los objetos de las clases que representan las entidades, lo que evidencia que funcionalidades son creadoras de dichas entidades.
- **Controlador**: Todas las peticiones web son manipuladas por un controlador frontal, ejemplo de esto son las clases “app.php” y “app_dev.php”, que es el punto de entrada único de toda la aplicación en un entorno determinado. Este patrón se evidencia en las clases controladoras del sistema. En Symfony 2 hay una capa específicamente para los controladores, que son el núcleo del mismo, aquí cada clase en esta capa tiene su responsabilidad y es única, hay controladores que se encargan de la seguridad del sistema trabajando con ficheros “yml” de configuración. Esto se aplica a la clase “app.php” que es el controlador frontal.
- **Alta Cohesión**: Una de las características principales del marco de trabajo Symfony 2 es la organización del trabajo en cuanto a la estructura del proyecto. Realizar un diseño donde las clases mantengan una alta cohesión, permite que el software sea flexible a cambios sustanciales con efecto mínimo. Se emplea en las clases controladoras ya que fueron asignadas responsabilidades a las clases de forma tal que la cohesión siguiera siendo alta, o sea, cada clase se encargará de realizar solamente las funciones que estén en correspondencia con la responsabilidad que posea.

- **Bajo acoplamiento**: Este patrón se evidencia dentro del marco de trabajo Symfony 2 en las clases que implementan la lógica del negocio y de acceso a datos que se encuentran en el modelo, las cuales no tienen asociaciones con las de la vista o el controlador, lo que proporciona que la dependencia en este caso sea baja. Como existe poca dependencia entre esas clases se permite una mayor reutilización.

2.3.1.2. Patrones GoF

Con el uso del marco de trabajo Symfony 2 se aplican los siguientes patrones GoF:

- **Estrategia**: El componente Profiler de Symfony 2 almacena los datos recogidos en un motor de almacenamiento. Luego los algoritmos de persistencia están intercambiables, por lo que se puede utilizar cualquier tipo de almacenamiento de datos (sistema de ficheros, base de datos, almacenes de datos NoSQL). Las clases que implementan la interface ProfilerStorageInterface.php son las encargadas de guardar los datos en dependencia del tipo de almacenamiento.
- **Inicialización retardada**: La inyección de dependencias en Symfony 2 permite al contenedor de servicios estandarizar y centralizar la manera que en que los objetos son construidos en la aplicación. La clase Container.php retrasa la instanciación de servicios que reutiliza y comparte al mismo tiempo.
- **Composite**: Cada elemento que compone a un Formulario es una instancia de la clase Formulario. Cada instancia de Formulario mantiene una referencia a la instancia de su Formulario padre y a una colección de Formularios hijos. Cuando se crea el árbol de estructuras cada nodo posee una referencia a su nodo padre y a sus nodos hijos.
- **Iterator:** El componente Finder de Symfony 2 permite leer el contenido de algún directorio recursivamente o buscar elementos en una colección de objetos. La clase PhpAdapter.php por ejemplo posee el método searchInDirectory ($dir) para buscar elementos en directorios de manera recursiva.
- **Mediator:** El evento dispatcher del componente EventDispatcher de Symfony 2 administra las conexiones que existen entre un sujeto y observadores asociados. Por ejemplo la clase EventDispatcher.php es la encargada de gestionar las conexiones entre los observadores (*listeners*) y un evento determinado.
- **Fachada**: El uso de este patrón se evidencia en la implementación de las interfaces que brindarán servicios al módulo Portal y al módulo Seguridad, con

el objetivo de lograr la integración con el módulo Estructura y Composición para el nuevo marco de trabajo desarrollado en Symfony 2.

- **Command**: Este patrón se observa en las clases “app.php” y “app_dev.php”, son las encargadas de establecer el módulo y la acción que se va a usar según la petición del usuario. Esto se aplica en la clase “RoutingManipulator.php”, que está desactivada por defecto y procede según las necesidades del administrador del sistema donde se aplique el marco de trabajo, la cual se puede activar o desactivar. En este método se comprueba la URL con el objetivo de precisar los parámetros de la misma y de esta forma saber el action que debe responder a la petición.
- **Decorator**: Este patrón se observa en la clase “base.html.twig”, padre de todas las vistas, que contienen un decorador para permitir agregar funcionalidades dinámicamente. El archivo nombrado “layout.php.twig” es el que contiene el layout de la página. Este archivo, conocido también como plantilla global, guarda el código HTML que es usual en todas las páginas del sistema, para no tener que repetirlo en cada página. El contenido de la plantilla se integra en el layout, o si se mira desde el otro punto de vista, el layout decora la plantilla. Este procedimiento es una implementación del patrón Decorator.
- **Registry**: Este patrón se aplica en la clase “GlobalVariables.php”, que es la encargada de acumular todas las variables de uso global en el sistema (Hamon, 2014).

2.3.2. Modelo arquitectónico

Según Sommerville existen 3 modelos arquitectónicos para representar la arquitectura de software:

- Modelo Repositorio de datos.
- Modelo Cliente-Servidor.
- **Modelo de Capas.**

“Un modelo arquitectónico refleja la estrategia básica para estructurar la organización de un sistema, tomando decisiones sobre la totalidad del modelo organizacional al principio del diseño arquitectónico” (Sommerville, 2005).

El Modelo de Capas (en ocasiones denominada modelo de máquina abstracta) organiza el sistema en capas cada una de las cuales proporciona un conjunto de servicios a las capas inmediatas. Este modelo arquitectónico es portable y soporta los cambios, en la

medida que su interfaz lo admita, una capa puede remplazarse por otra. La figura 7 representa gráficamente la estructura del modelo arquitectónico de la solución del sistema.

Figura 7: Modelo arquitectónico en capas del sistema.

2.3.3. Arquitectura orientada a componentes

Es un estilo arquitectónico impuesto al diseño del sistema, el objetivo es establecer una estructura para todos los componentes del mismo, ya que describe una aproximación de ingeniería de software al diseño y desarrollo de un sistema. Está enfocado a la descomposición del diseño en componentes funcionales o lógicos que expongan interfaces de comunicación bien definidas y proporcionen los servicios requeridos en la aplicación. En esta arquitectura la interfaz es el elemento básico de comunicación, por lo que cada componente debe describir de forma completa las interfaces que ofrece, así como las interfaces que requiere para su operación. “La modularidad, la reusabilidad y componibilidad son características muy relevantes de una arquitectura basada en componentes” (Pressman, 2010).

Este estilo arquitectónico trae consigo una serie de beneficios como la facilidad de instalación, o sea, cuando una nueva versión de la aplicación esté disponible puede reemplazarse la versión existente sin impactar otros componentes o el sistema como un todo. También permite la reusabilidad de los componentes entre diversas aplicaciones y por consiguiente, reducir los costos de desarrollo y mantenimiento, posibilita la mitigación de la complejidad técnica mediante el uso de contenedores de componentes y sus servicios y brinda facilidad de desarrollo.

Existen varios conceptos del término componente, para este trabajo se utilizará la definición adoptada y publicada por el Software Engineering Institute (SEI) la cual propone que: “Un componente es un fragmento de un sistema de software que puede ser ensamblado con otros fragmentos para formar piezas más grandes o aplicaciones completas”.

"Esta definición se basa en tres perspectivas de un componente: la perspectiva de empaquetamiento que considera un componente como unidad de empaquetamiento, la perspectiva de servicio que considera un componente como proveedor de servicios y la perspectiva de integridad que considera un componente como un elemento encapsulado" (Centro de Informatización de Gestión de Entidades, 2013).

Un componente debe cumplir una serie de requisitos:

Identificable: Un componente debe tener una identificación clara y consistente que facilite su catalogación y búsqueda en repositorios de componentes.

Accesible: Sólo a través de su interfaz el componente debe exponer únicamente el conjunto de operaciones que lo caracteriza y ocultar sus detalles de implementación. Esta característica permite que un componente sea reemplazado por otro que implemente la misma interfaz.

Documentado: Un componente debe tener una documentación adecuada que facilite su búsqueda en repositorios de componentes, evaluación, adaptación a nuevos entornos, integración con otros componentes y acceso a información de soporte.

Estandarizado: Significa que estos deben ajustarse a algún modelo de componentes.

Independiente: Debe poder componerse y desplegarse sin tener que utilizar otros componentes.

Componible: Las interacciones externas deben ser a través de interfaces definidas públicamente.

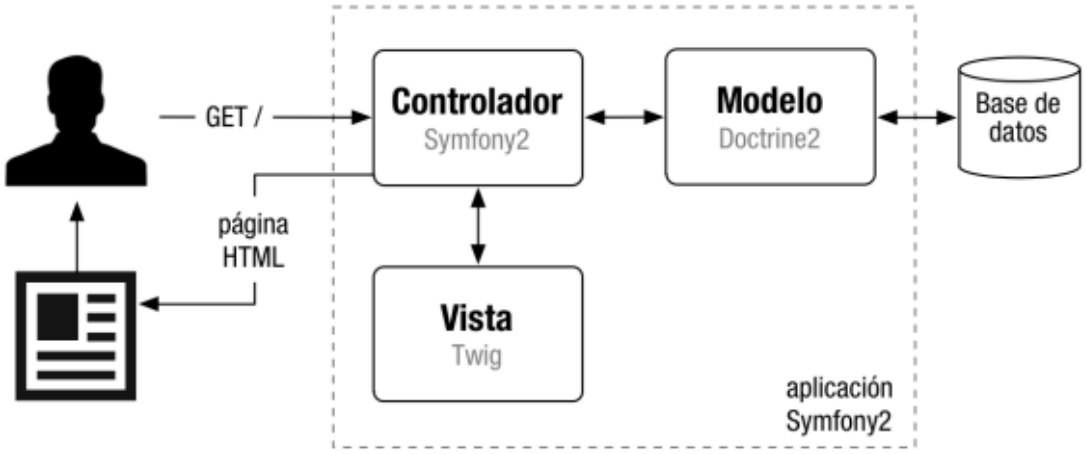
2.3.4. Patrón arquitectónico

Describe un problema del diseño arquitectónico que se origina en un contexto específico, y presenta una solución general y probada. La solución se especifica mediante la descripción de los componentes que la constituyen, sus responsabilidades, las relaciones entre ellos y la forma en que colaboran. El **Modelo Vista Controlador (MVC)** es un patrón arquitectónico para el desarrollo de software que delimita los datos y la lógica de negocio de una aplicación de la interfaz de usuario y el responsable de administrar los eventos, servicios y comunicaciones. Este patrón está estructurado en tres capas que son el **modelo**, la **vista** y el **controlador**, es decir, por un lado define mecanismos para la representación de la información, y por otro lado para la interacción con el usuario. MVC se basa en la reutilización de código y la separación de conceptos y permite dividir la propuesta de solución en las capas (Quintero, 2010).

- **Modelo**: La capa del modelo representa la información con la que trabaja la aplicación, es decir la lógica de negocio (la base de datos pertenece a esta capa). Esta se puede dividir en la capa de acceso a los datos y en la capa de abstracción de la base de datos, utiliza Doctrine que se encarga de la generación automática de las clases que pertenecen a esta capa, en dependencia de la estructura de la base de datos. Symfony 2 guarda todas las clases y archivos relacionados con el modelo en el directorio EstructuraComposicionBundle\Entity, ejemplos de estas clases son DatCargo.php y DatEstructura.php.
- **Vista:** La vista es lo que utilizan los usuarios para interactuar con la aplicación (el motor de plantillas Twig pertenece a esta capa), es decir transforma el modelo en una página web. En Symfony 2 la capa de presentación está formada principalmente por plantillas (un ejemplo es index.html.twig) y el layout (base.html.twig) que contiene la información visual común a todas las páginas. Estas plantillas se guardan en el directorio llamado Resources\views\Default.
- **Controlador:** El controlador es el que realiza llamadas al modelo para obtener los datos y se los pasa a la vista para que los muestre al usuario. En Symfony 2 todas las peticiones se canalizan a través de los controladores frontales (app.php y app_dev.php). Estos controladores frontales realmente delegan todo el trabajo en las "action", son métodos con el nombre insertAction, updateAction, entre otros de una clase entidad determinada. El controlador que tiene las acciones de una entidad (tabla de la base de datos) se encuentra en el directorio EstructuraComposicionBundle\Controller.

El cliente envía una señal llamada *REQUEST* o Petición, ésta es interceptada por el **Controlador** quien realiza las validaciones necesarias, procesamiento de dichos datos y lógica de negocio asociadas a esa petición del cliente. El **Controlador** hace uso del **Modelo** para buscar la información necesaria en la base de datos., y la obtiene dependiendo de la solicitud del usuario para finalmente enviarlos a la **Vista** para ser mostrados nuevamente al cliente a través de un *RESPONSE* o respuesta. Symfony 2 basa su funcionamiento interno en este patrón arquitectónico, aunque tiene su propia forma de trabajo, ya que implementa una variante del MVC llamado **controlador frontal**, que se relaciona con el diseño de aplicaciones web (Quintero, 2010). Ofrece un punto de acceso centralizado para el tratamiento de todas las solicitudes y canaliza todas las peticiones en un único controlador. En la figura 8 se muestra lo anteriormente explicado.

Figura 8: Imagen que aparece en manual oficial de Symfony 2.



2.3.5. Diagrama de clases del diseño

Con el objetivo de obtener clases más independientes, reutilizables y fáciles de mantener. Las clases están distribuidas en capas siguiendo el diseño que impone el patrón arquitectónico MVC. Los datos de la tabla 5 describen las clases que se muestran en el diagrama de clases basado en estereotipos web de la figura 9 que fueron confeccionados durante el proceso de desarrollo correspondiente a la funcionalidad Definir nivel estructural.

Tabla 5: Descripción del diseño de clases Definir nivel estructural.

Capa de acceso a datos	
RunSqlDoctrineCommand.php	Clase perteneciente a la versión de Doctrine que usa Symfony 2 encargada de ejecutar las consultas SQL a la base de datos.
Doctrine_Command.php	Tiene la responsabilidad de gestionar la conexión a la base de datos.
NomNomencladoreavestruc.php	Clase entidad que representa la tabla respectiva en la base de datos. Posee los atributos de los niveles estructurales y los métodos de acceso.
NomCampoestruc.php	Clase entidad que representa la tabla respectiva en la base de datos. Tiene los atributos de los campos que puede poseer una estructura determinada.

Capa de lógica de negocio	
GestionarNivelEstructuralController.php	Es la clase controladora de la entidad NomNomencladoreavestruc.php y tiene los métodos con las responsabilidades de crear, eliminar y mostrar los niveles estructurales.
GestionarNomCampoestruc.php	Es la clase controladora de la entidad NomCampoestruc.php y contiene a los métodos con las responsabilidades de crear, eliminar y mostrar los campos de una estructura.
Controller.php	Clase controladora que provee el marco de trabajo Symfony 2, de la misma heredan todas las demás clases controladoras de un proyecto.
Capa de presentación	
GestionarNivelEstructural.js	Formulario que sirve para adicionar o modificar los datos de los niveles estructurales.
GestionarCampos.js	Formulario encargado de gestionar los campos de los niveles estructurales que se hayan adicionado.
DefinirNivelEstructural.js	Página principal que muestra los distintos niveles estructurales y las relaciones entre ellos.

Los datos de la tabla 6 describen las clases que se muestran en el diagrama de clases basado en estereotipos web de la figura 10 que fueron confeccionados durante el proceso de desarrollo correspondiente a la funcionalidad Gestionar estructuras.

Tabla 6: Descripción del diseño de clases Gestionar estructuras.

Capa de acceso a datos	
RunSqlDoctrineCommand.php	Clase perteneciente a la versión de Doctrine que usa Symfony 2 encargada de ejecutar las consultas SQL a la base de datos.

Doctrine_Command.php	Tiene la responsabilidad de gestionar la conexión a la base de datos.
DatEstructura.php	Clase entidad que representa la tabla respectiva en la base de datos. Posee los atributos de las estructura externas y los métodos de acceso.
DatEstructuraop.php	Clase entidad que representa la tabla correspondiente en la base de datos. Posee los atributos de las estructuras internas y los métodos de acceso.
Capa de lógica de negocio	
GestionarEstructurasController.php	Es la clase controladora que tiene los métodos con las responsabilidades de crear, eliminar y mostrar las estructuras organizacionales.
Controller.php	Clase controladora que provee el marco de trabajo Symfony 2, de la misma heredan todas las demás clases controladoras de un proyecto.
Capa de presentación	
GestionarEstructuraExterna.js	Formulario que sirve para adicionar o modificar los datos de las estructuras externas.
GestionarEstructuraInterna.js	Formulario que sirve para adicionar o modificar los datos de las estructuras internas.
GestionarEstructuras.js	Página principal que muestra las estructuras organizacionales con la composición interna respectiva.

Figura 9: Diagrama de clases del diseño Definir nivel estructural.

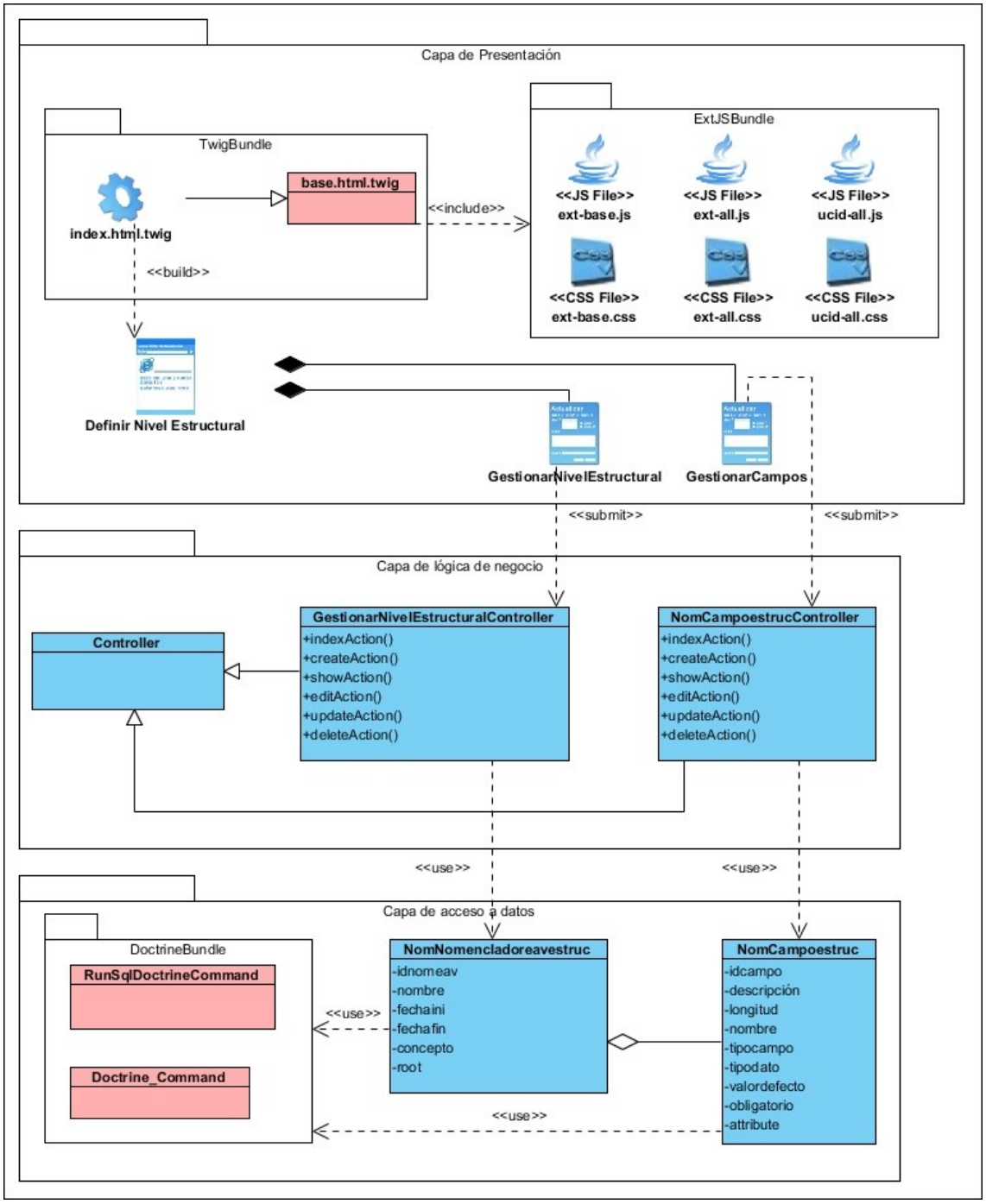
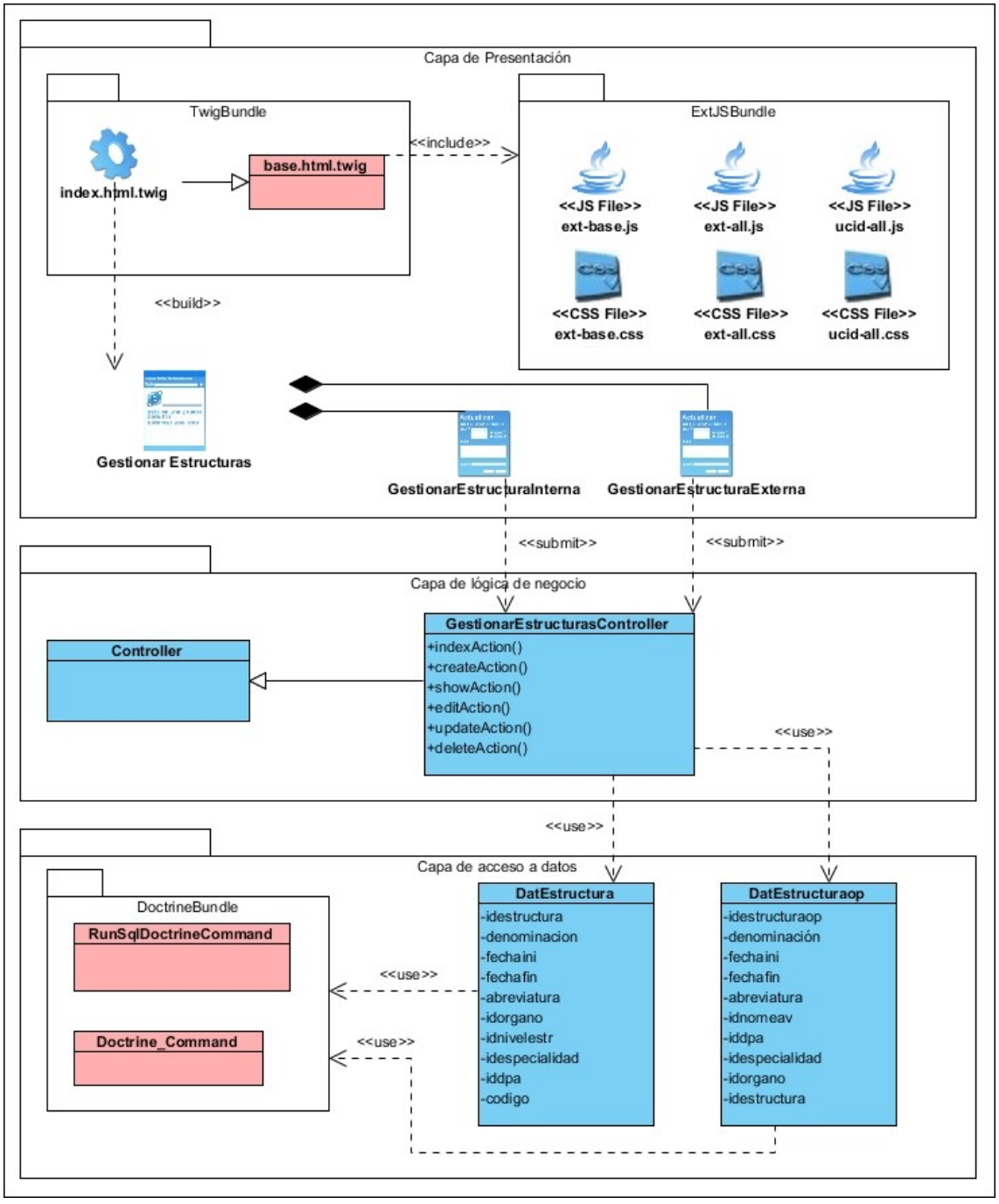


Figura 10: Diagrama de clases del diseño Gestionar estructuras.



2.4. Modelado de la Base de Datos

El Modelo de datos correspondiente al módulo Estructura y Composición tiene un total de 24 tablas, de las cuales 18 son nomencladores, el resto constituyen las tablas que almacenan las características propias de las estructuras. La tabla 7 detalla las principales tablas de la base de datos.

2.4.1. Descripción del Modelo de datos

Tabla 7: Tablas de la base de datos del módulo Estructura y Composición

dat_agrupacionest	Guarda la relación entre las tablas dat_estructura y nom_agrupacion.
dat_agrupacionestop	Guarda la relación entre las tablas dat_estructuraop y nom_agrupacion.
dat_estructura	Guarda los datos correspondientes a las estructuras externas.
dat_estructuraop	Guarda los datos correspondientes a las estructuras internas.
dat_cargo	Guarda los datos comunes de los cargos.
dat_cargocivil	Guarda los datos de los cargos que se asocian a las estructuras internas.
nom_aristaeav	Guarda la relación entre los niveles estructurales.
nom_agrupacion	Guarda los datos correspondientes al nomenclador agrupación lógica.
nom_calificador_cargo	Guarda los datos correspondientes al nomenclador calificador de los cargos.
nom_campoestruc	Guarda los datos de los campos dinámicos correspondientes a cada valor en la tabla nom_nomencladoreavestruc.
nom_cargocivil	Guarda los datos correspondientes al nomenclador cargo.
nom_categcivil	Guarda los datos correspondientes al nomenclador responsabilidad.
nom_categocup	Guarda los datos correspondientes al nomenclador categoría ocupacional.
nom_clasificacion_cargo	Guarda los datos correspondientes al nomenclador tipo de plantilla.
nom_escalasalarial	Guarda los datos correspondientes al nomenclador de tipo escala salarial.
nom_grupocomple	Guarda los datos correspondientes al nomenclador grupo de complejidad.

nom_nivel_utilización	Guarda los datos correspondientes al nomenclador nivel de utilización.
nom_nivelestr	Guarda los datos correspondientes al nomenclador nivel jerárquico.
nom_nomencladoreavestruc	Nomenclador de los datos asociados a los distintos niveles estructurales.
nom_organo	Guarda los datos del nomenclador órgano.
nom_salario	Guarda los datos correspondientes al nomenclador escala salarial.
nom_tipo_calificador	Guarda los datos correspondientes al nomenclador tipo de calificador.
nom_tiipocifra	Guarda los datos correspondientes al nomenclador tipo de cifra.
nom_valor_defecto	Guarda los datos correspondientes a los valores por defecto de los campos.

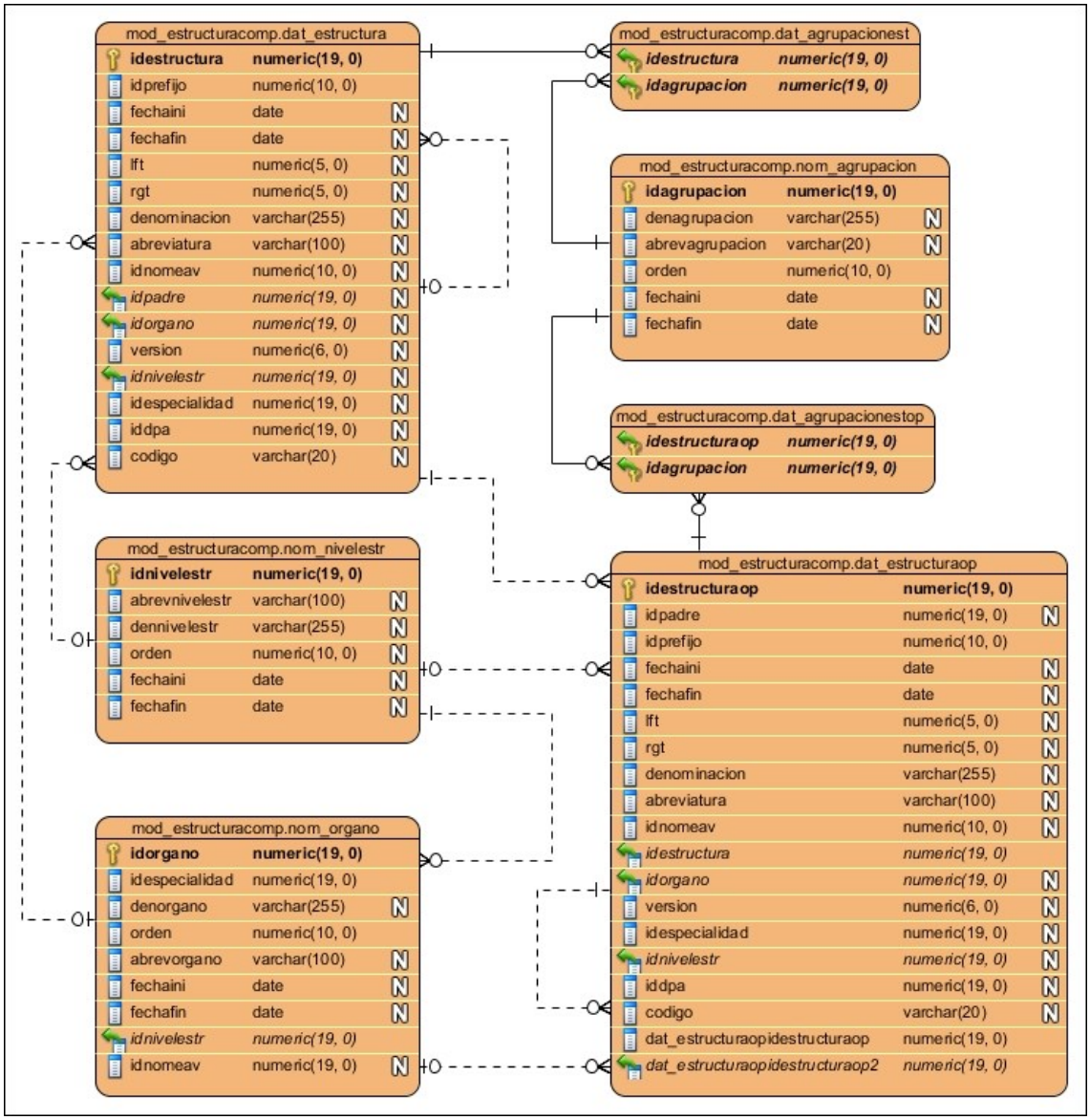
2.4.2. Diagrama de Entidad-Relación

En la figura 11 correspondiente a un fragmento del diagrama de entidad-relación se muestran las tablas dat_estructura, dat_estructuraop, nom_organo, nom_agrupacion, nom_nivelestr, dat_agrupacionest y dat_agrupacionestop, así como la relación existente entre estas. La tabla dat_estructura almacena los datos de las estructuras externas que se estén gestionando. Contiene como atributos el identificador de la estructura, que es la llave primaria de la tabla, la denominación de la estructura, la abreviatura, que representa las siglas de dicha denominación; el código que se le asigna, el identificador del órgano asociado a la misma, el identificador del nivel estructural que tiene, las fechas de inicio y de fin del uso de la estructura, y el identificador de la estructura padre de dicha entidad.

La tabla dat_estructuraop almacena los datos referentes a las estructuras internas. La misma tiene definidos los mismos campos que la tabla anterior, con la diferencia de que el identificador en este caso es de la estructura interna y se incluye como un atributo más, el identificador de la estructura externa, a la cual se asocia la estructura interna gestionada. La estructura externa como la estructura interna están en agrupaciones lógicas determinadas, las cuales se encuentran establecidas en el nomenclador nom_agrupacion. Sus atributos constituyen el identificador de la agrupación, que se corresponde con la llave primaria del nomenclador, el nombre de la agrupación, la

abreviatura que responde a dicho nombre, el orden que tiene la agrupación y las fechas de inicio y fin del uso del nomenclador.

Figura 11: Fragmento del Diagrama Entidad-Relación de Estructura y Composición



Las relaciones entre estas tres tablas se establecen mediante las tablas dat_agrupacionest y dat_agrupacionestop, las cuales permiten determinar y mostrar la agrupación a la cual pertenecerán las estructuras que están siendo gestionadas. Es por ello que comprenden el identificador de la agrupación y el identificador de la estructura, en el caso de dat_agrupacionest, para el caso de dat_agrupacionestop, es el identificador de la estructura interna. El nomenclador nom_organo especifica los distintos órganos que forman parte de las estructuras organizacionales, acorde al nivel estructural al cual pertenecen. Es por ello que guarda relación con las tablas que representan a ambas estructuras. En sus campos engloba el identificador del órgano, que también es la llave primaria, el identificador del nivel estructural al que pertenece,

su nombre y la abreviatura que responde a este, el orden que tiene el nomenclador y las fechas de inicio y fin del uso del mismo. Este nomenclador se relaciona con el nomenclador nom_nivelestr, debido a que cada órgano pertenece a un nivel determinado. Este guarda los datos de los niveles estructurales que pueden presentar las estructuras organizacionales, por ejemplo nivel 1, nivel 2, nivel 4, etc. Dentro de sus atributos se encuentran la denominación del nivel y la abreviatura que recibe la misma, el orden del nomenclador y las fechas de inicio y fin del uso del mismo.

2.5. Modelado de la implementación

El modelo de implementación describe cómo los elementos del modelo de diseño, se implementan en términos de componentes. Detalla también cómo se organizan las aplicaciones de acuerdo con los mecanismos de estructuración disponibles en el escenario de implementación y los lenguajes de programación utilizados, y cómo interactúan los componentes unos de otros, además de los recursos necesarios para poder ejecutar el sistema desarrollado.

Symfony 2 proporciona facilidades para el trabajo, y acentúa una arquitectura que promueve el código reutilizable y disociado (Potencier, 2011). La estructura de directorios de una aplicación desarrollada en Symfony 2 es la siguiente:

- app\: Configuración de la aplicación.
- src\: El código PHP del proyecto.
- vendor\: Las dependencias de terceros.
- web\: El directorio raíz del servidor web.

La configuración de todo el proyecto se encuentra en el directorio app\. En src se encuentra el código de la aplicación, la cual estará formada por módulos denominados bundles que constituyen un conjunto de archivos que implementan una única funcionalidad. La configuración de un *bundle* se puede realizar a través del archivo config.yml en la carpeta config\ del proyecto, o a través del archivo services.yml que se encuentra en la carpeta config\ dentro del bundle; de cualquier modo en esta configuración solo se explicitan los servicios que provee un bundle, pudiéndose consumir dichos servicios desde cualquier parte del sistema. Un servicio es cualquier objeto PHP que realiza alguna tarea “global”, un objeto creado para un propósito determinado. El servicio se utiliza en el sistema siempre que se necesite la funcionalidad específica que este proporciona.

Al declarar un servicio en alguno de los archivos de configuración también se detallan los parámetros que debe recibir este servicio así como la ubicación de la clase que

implementa el servicio. El componente o bundle del módulo Estructura y Composición estará dentro de la carpeta src\ y debe existir un fichero **composer.json** en la raíz de la aplicación que contenga los metadatos del componente, servicios y dependencias. Este fichero tiene la siguiente estructura:

{“services”: [

“identificador servicio”:

{“interface”: “namespace relativa a la interfaz”,

“impl:” “namespace relativa a la implementación”}

], “dependencies”: [“identificador dependencia”:

{“interface”: “namespace relativa a la interfaz”,

“optional”: “valor booleano”,

“use”: “nombre componente que resuelve”

“versión”: “X.X.X”}]}

En las dependencias, el uso de las propiedades **optional**, **use**, y **versión** son opcionales. La primera si está en false, garantiza que si no se resuelve esa dependencia el componente no funciona. La segunda es para especificar el componente y la tercera que versión del componente puede resolver la dependencia. La versión puede no escribirse, y también puede usar comodines por ejemplo: >=1.0.*.

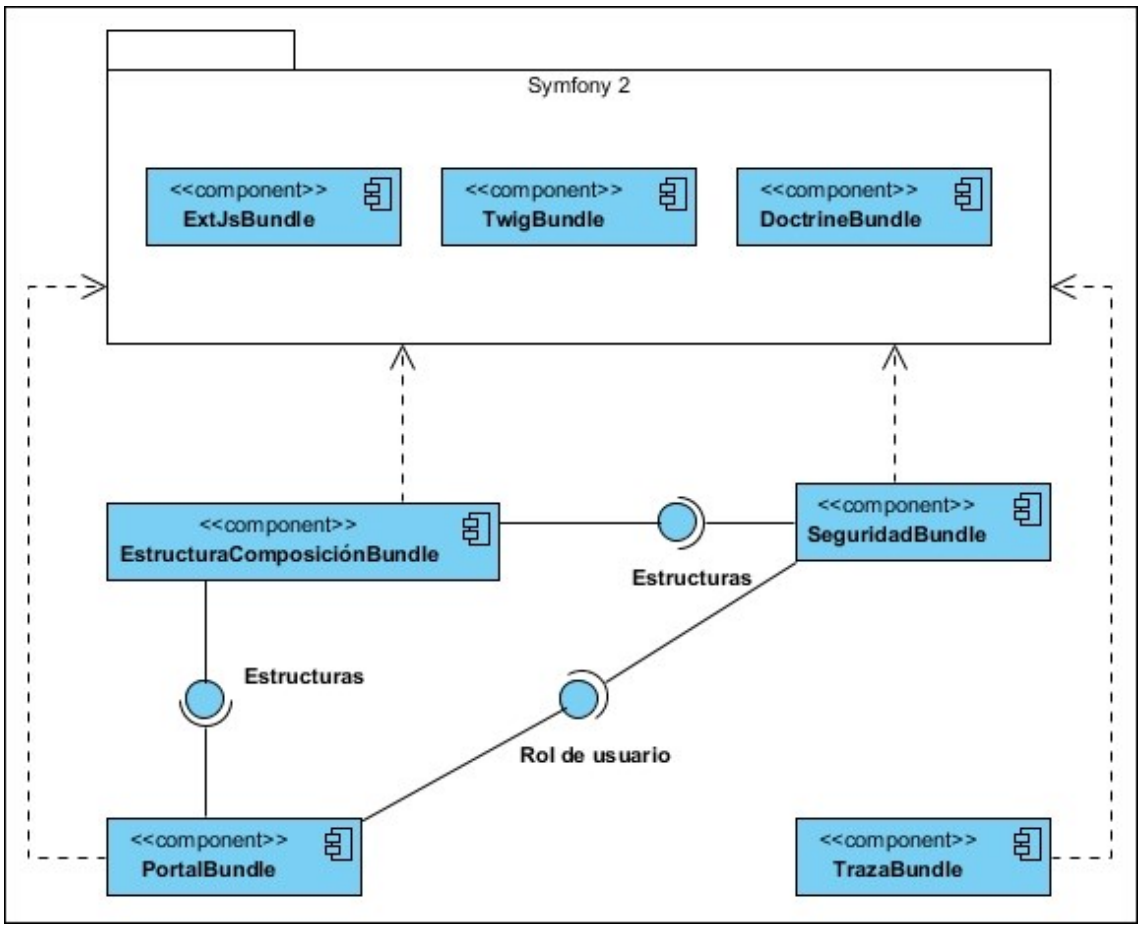
Cada componente que brinde servicios o tenga dependencias debe tener una interfaz PHP donde se definan los métodos que va a brindar el mismo. Los métodos deben estar descritos usando PHPNotation. Las interfaces y clases tienen que tener un namespace para que sean únicas, el namespace coincide con la ruta del archivo a partir del directorio src\ del proyecto, pero sin incluir el propio nombre del archivo, lo que lleva es el nombre de la clase.

Debido al mecanismo de integración de componentes en Symfony 2 es posible el consumo de servicios desde un bundle de una aplicación hacia otro, los componentes o bundles consumen servicios que brindan otros componentes o bundles, sin embargo en el marco de trabajo Sauxe esta comunicación se realiza mediante el consumo de métodos que se encuentran en otros componentes. El módulo de Estructura y Composición identificado como un objeto bundle hará uso del patrón Fachada ya que brinda servicios que solo se accederán a través de las interfaces de clases en las que se encuentran declarados.

2.5.1. Diagrama de Componentes

En el diagrama de componentes se muestran los elementos de diseño de un sistema de software, permite visualizar con más facilidad la estructura general del sistema y el comportamiento de los servicios que estos componentes proporcionan y utilizan a través de las interfaces. La propuesta de solución cuenta con una serie de componentes que interactúan entre sí. La figura 12 muestra el Diagrama de componentes elaborado.

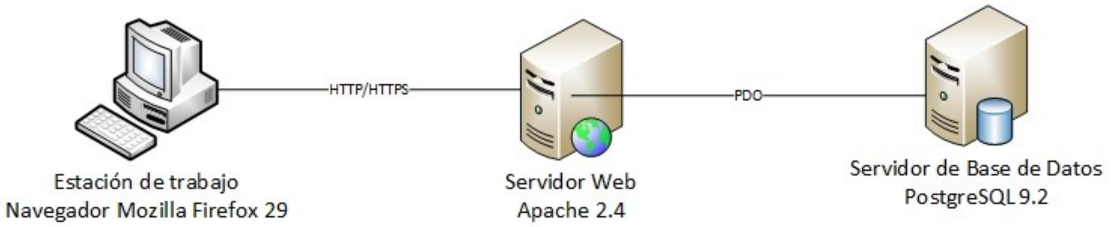
Figura 12: Diagrama de componentes



2.5.2. Diagrama de despliegue

El usuario desde una estación de trabajo podrá acceder al sistema a través de los protocolos HTTP o HTTPS utilizando un navegador web, el sistema estará alojado en un servidor web Apache. Dicho servidor se conectará a un servidor de bases de datos PostgreSQL mediante la extensión de acceso a datos PDO, en el cual se almacenará la información de interés para la solución. La figura 13 muestra el Diagrama de despliegue desarrollado.

Figura 13: Diagrama de despliegue



2.6. Conclusiones parciales

En este capítulo fueron mostrados los artefactos generados durante el diseño de la solución propuesta. En el mismo se identificaron y describieron los requisitos funcionales y no funcionales, se conformó la arquitectura del sistema que rige el diseño, el modelo de datos, el diagrama de clases del diseño propuesto, el diagrama de componentes y el de despliegue. Los requisitos fueron identificándose en el transcurso del desarrollo de la solución propuesta. Con el desarrollo de esta solución se adquirieron conocimientos sobre la gestión de estructuras organizacionales en las empresas y el desarrollo de aplicaciones web de gestión sobre el marco de trabajo Symfony 2. Una vez concluido el diseño e implementación de la solución se prosigue a la validación de la solución.

3. CAPÍTULO 3 VALIDACIÓN DE LA SOLUCIÓN

Este capítulo tiene como objetivo evaluar y validar la calidad del sistema aplicando un conjunto de técnicas como las que se explicarán seguidamente. Se evaluará el diseño de la aplicación empleando métricas de software para conocer cómo se afectan los valores de reutilización, acoplamiento, responsabilidad de las clases y complejidad de la implementación. También para evaluar la calidad del sistema se aplicarán pruebas de caja negra con el objetivo de detectar y corregir el máximo de errores, antes de su entrega al cliente.

La IEEE plantea que la validación de software es: *"el proceso de evaluar un sistema o componente, durante o al final del proceso de desarrollo, con el fin de determinar si satisface los requerimientos especificados".*

3.1. Métricas de software orientadas a clases para evaluar el diseño

Las métricas de software son una medida cuantitativa que permite a los desarrolladores tener una visión profunda de la eficacia del proceso del software y de los proyectos que dirigen utilizando el proceso como un marco de trabajo. Estos datos son analizados, comparados con promedios anteriores y evaluados, para determinar las mejoras en la calidad y productividad. Las métricas son también utilizadas para señalar áreas con problemas, de manera que se puedan desarrollar los remedios y mejorar el proceso del software (Pressman, 2010).

Debido a que este sistema se realizó bajo la programación orientada a objetos (POO) y las clases constituyen la unidad básica y fundamental en este tipo de programación, resulta viable realizar la validación del mismo con la aplicación de métricas dirigidas a sus clases de forma individual, sus jerarquías y colaboraciones.

Las métricas empleadas están diseñadas para evaluar los siguientes atributos de calidad:

- *Responsabilidad*. Consiste en la responsabilidad asignada a una clase en un marco de modelado de un dominio o concepto, de la problemática propuesta.
- *Complejidad de la implementación*: Consiste en el grado de dificultad que tiene implementar un diseño de clases determinado.
- *Reutilización*: Consiste en el grado de reutilización presente en una clase o estructura de clase, dentro de un diseño de software.
- *Acoplamiento*: Consiste en el grado de dependencia o interconexión de una clase o estructura de clase con otras. Está muy ligada a la característica de reutilización.

- *Complejidad del mantenimiento*: Consiste en el grado de esfuerzo necesario a realizar para desarrollar un arreglo, una mejora o una rectificación de algún error de un diseño de software. Puede influir indirectamente, pero fuertemente en los costes y la planificación del proyecto.
- *Cantidad de pruebas*: Consiste en el número o el grado de esfuerzo para realizar las pruebas de calidad (unitarias) del producto (componente, módulo, clase, conjunto de clases, entre otras) diseñado.

Las métricas concebidas como herramienta para evaluar la calidad del diseño del módulo Estructura y Composición y su relación con los atributos de calidad definidos en esta investigación son las siguientes:

3.1.1. Tamaño operacional de clases (TOC)

El tamaño operacional de las clases está dado por el número de métodos asignados a una clase. Mediante esta métrica se calcula el nivel de responsabilidad de los métodos, la complejidad de la implementación de los mismos y su reutilización, a fin de inspeccionar la efectividad del diseño, existiendo una relación directa con los dos primeros e inversa con el último antes mencionado.

Tabla 8: Atributos de calidad evaluados por la métrica TOC

Atributo de calidad	Modo en que lo afecta
Responsabilidad.	Un aumento del TOC implica un aumento de la responsabilidad asignada a la clase.
Complejidad de la implementación.	Un aumento del TOC implica un aumento en la complejidad de la implementación de la clase.
Reutilización.	Un incremento del TOC implica una disminución del grado de reutilización de la clase.

Tabla 9: Criterios de evaluación para la métrica TOC

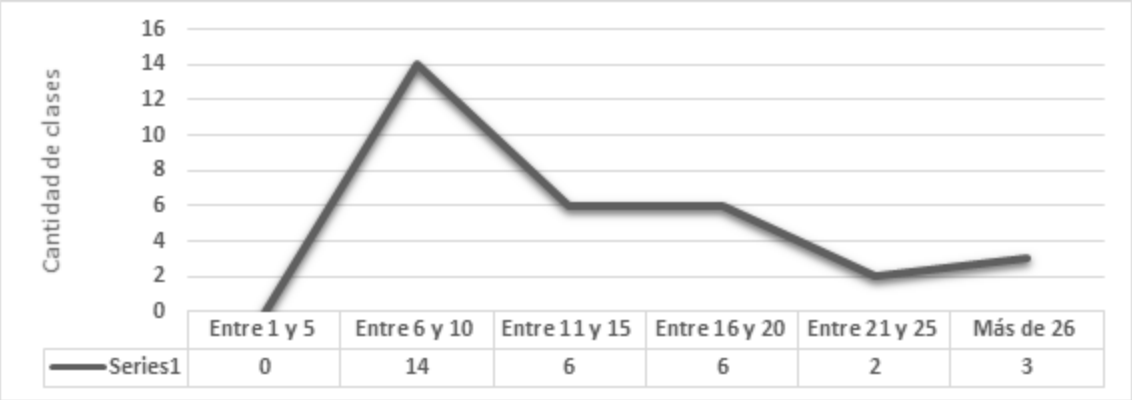
Atributo	Categoría	Criterio
Responsabilidad.	Baja.	< =Prom.
	Media.	Entre Prom y 2* Prom.
	Alta.	> 2* Prom.
Complejidad de la implementación.	Baja.	< =Prom.
	Media.	Entre Prom y 2* Prom.
	Alta.	> 2* Prom.

Reutilización.	Baja.	> 2*Prom.
	Media.	Entre Prom. y 2* Prom
	Alta.	<= Prom.

3.1.2. Resultados obtenidos de la aplicación de la métrica TOC

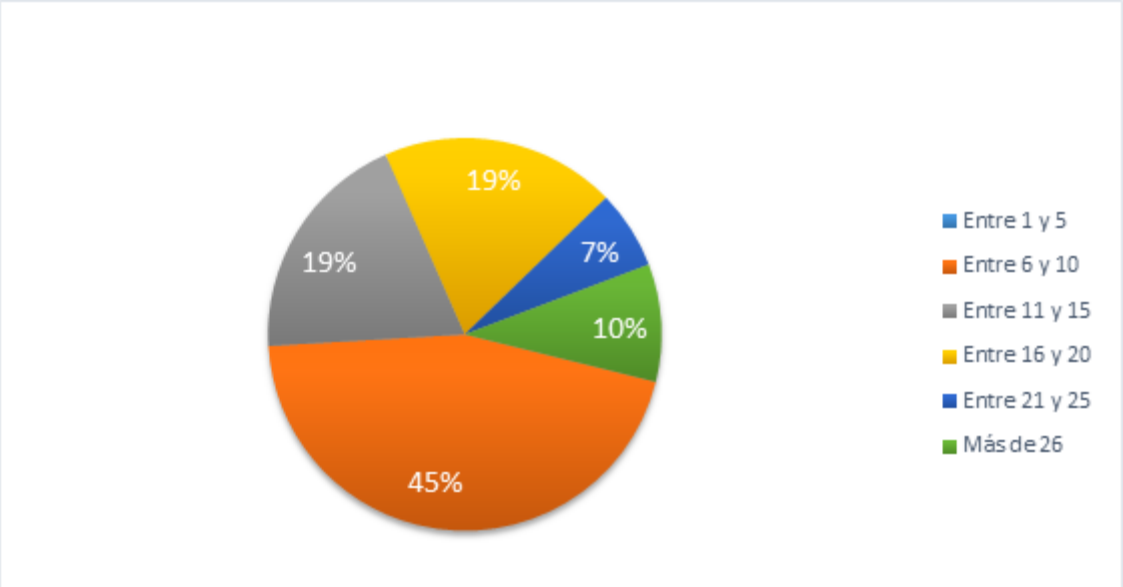
El gráfico 1 muestra la representación de los resultados obtenidos en la herramienta agrupados en los intervalos definidos. El gráfico refleja que la mayoría de las clases tienen de 6 a 10 procedimientos, lo que demuestra que el funcionamiento general del sistema está distribuido equitativamente entre la mayoría de las clases, de esta forma se garantiza que en caso de ocurrir algún cambio en el sistema de clases del componente, estaría menos comprometido el funcionamiento correcto del mismo.

Gráfico 1: Representación de la evaluación de la métrica TOC.



El gráfico 2 muestra la representación en % de los resultados obtenidos en la herramienta agrupados en los intervalos definidos.

Gráfico 2: Representación en % de la evaluación de la métrica TOC.



- Responsabilidad.

Los datos mostrados por la tabla 10 y el gráfico 3 muestran la incidencia de los resultados de la evaluación de la métrica TOC en este atributo, donde quedó demostrado que el 68 % de las clases tienen una baja responsabilidad, pues este atributo se distribuyó de manera equitativa entre todas las clases del sistema.

Tabla 10: Responsabilidad

Responsabilidad	Cantidad de clases	Promedio
Baja.	21	67,74193548
Media.	8	25,80645161
Alta.	2	6,451612903

Gráfico 3: Representación de la evaluación de la métrica TOC en el atributo Responsabilidad.

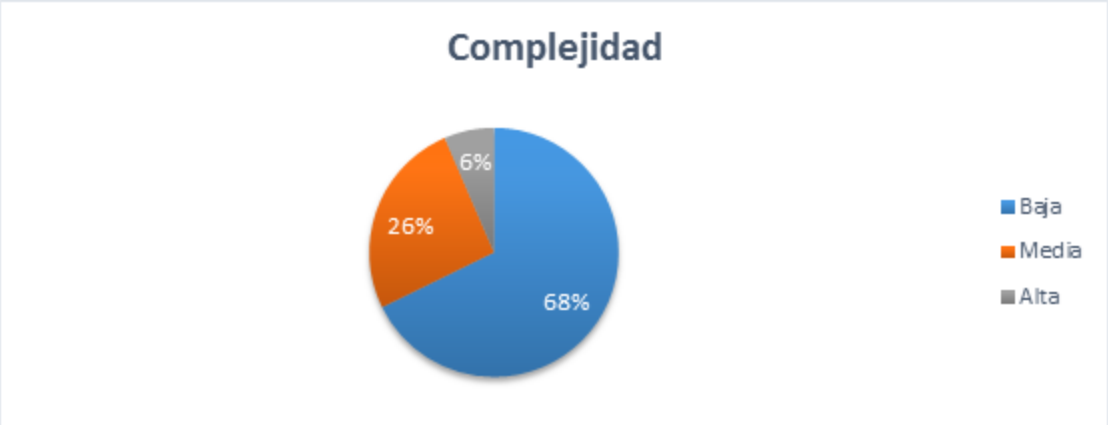
- Complejidad de la implementación.

Los datos de la tabla 11 y el gráfico 4 muestran la representación de la evaluación de la métrica TOC en este atributo, lo que demuestra que el 68% de las clases tienen una baja complejidad, este atributo está distribuido equitativamente entre todas las clases. Esta característica permite mejorar el mantenimiento y el soporte de la aplicación.

Tabla 11: Complejidad de implementación

Complejidad de la implementación	Cantidad de clases	Promedio
Baja.	21	67,74193548
Media.	8	25,80645161
Alta.	2	6,451612903

Gráfico 4: Representación de la evaluación de la métrica TOC en el atributo Complejidad de la implementación.



- Reutilización.

En la tabla 12 y en el gráfico 5 se muestra la representación de la incidencia de los resultados de la evaluación de la métrica TOC en este atributo, quedando demostrado que el diseño de la solución es aceptable ya que la mayoría de las clases tienen un alto grado de reutilización.

Tabla 12: Reutilización

Reutilización	Cantidad de clases	Promedio
Alta.	21	67,74193548
Media.	8	25,80645161
Baja.	2	6,451612903

Gráfico 5: Representación de la evaluación de la métrica TOC en el atributo Reutilización.

Cuando existe un TOC alto se afectan los parámetros de calidad definidos por esta métrica. Se reduce la reutilización de las clases, la implementación se hace más compleja, las pruebas son difíciles de realizar y aumenta la responsabilidad de las clases.

Haciendo un análisis de los resultados obtenidos en la herramienta de medición de la métrica TOC, se puede concluir que el diseño del módulo Estructura y Composición tiene una calidad aceptable, teniendo en cuenta que el 93 % de las clases incluidas en este sistema poseen menos cantidad de procedimientos que la mitad del valor máximo registrado en las mediciones. Además este mismo por ciento de clases posee evaluaciones positivas en los atributos de calidad (responsabilidad, complejidad de la implementación y reutilización) en el diseño propuesto, evidenciando que las clases no tienen tanta responsabilidad, no son tan complejas, y poseen un elevado grado de reutilización. Por lo que se concluye que los resultados obtenidos según esta métrica son positivos.

3.1.3. Relaciones entre clases (RC)

Las relaciones entre las clases están dadas por el número de relaciones de uso que tenga una clase con otras, o sea el número de dependencias que una clase tiene con otra. A través de esta métrica se calcula el acoplamiento, la complejidad de mantenimiento, la reutilización y la cantidad de pruebas a fin de inspeccionar la efectividad del diseño, existiendo una relación directa con los tres primeros e inversa con el último antes mencionado.

Tabla 13: Atributos de calidad evaluados por la métrica RC

Atributo de calidad	Modo en que lo afecta
Acoplamiento.	Un aumento del RC implica un aumento del acoplamiento de la clase.
Complejidad de mantenimiento.	Un aumento del RC implica un aumento de la complejidad del mantenimiento de la clase.
Reutilización.	Un aumento del RC implica una disminución en el grado de reutilización de la clase.
Cantidad de pruebas.	Un aumento del RC implica un aumento de la cantidad de pruebas de unidad necesarias para probar una clase.

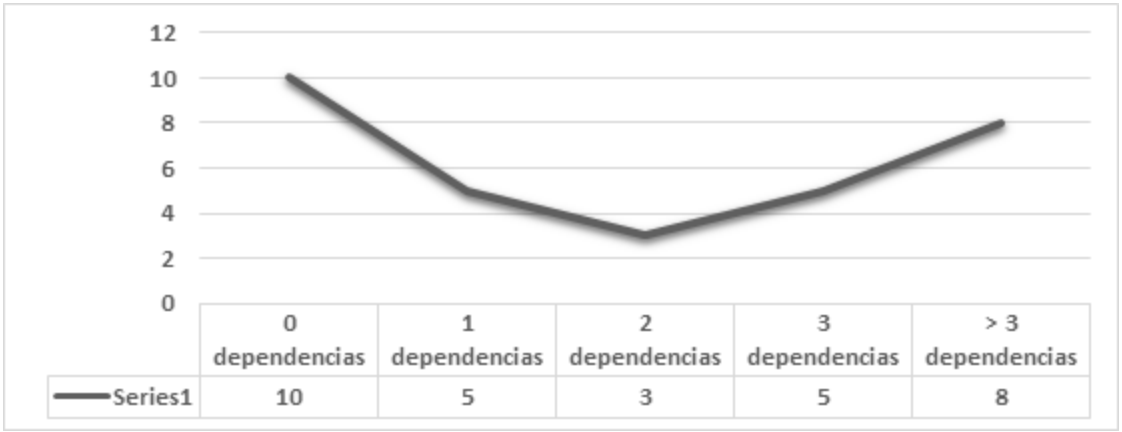
Tabla 14: Criterios de evaluación para la métrica RC

Atributo	Categoría	Criterio
Acoplamiento.	Ninguno.	0
	Bajo.	1
	Medio.	2
	Alto.	> 2
Complejidad de mantenimiento.	Baja.	< =Prom.
	Media.	Entre Prom y 2* Prom.
	Alta.	> 2* Prom.
Reutilización.	Baja.	> 2*Prom.
	Media.	Entre Prom y 2* Prom
	Alta.	<= Prom.
Cantidad de pruebas.	Baja.	< =Prom.
	Media.	Entre Prom y 2* Prom.
	Alta.	> 2* Prom.

3.1.4. Resultados obtenidos de la aplicación de la métrica RC

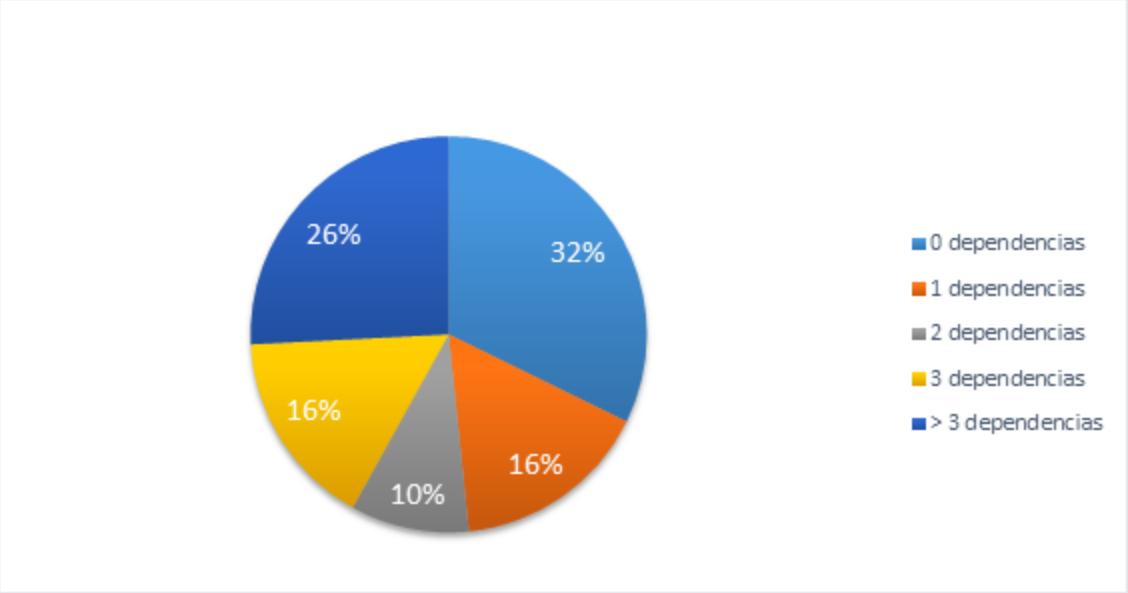
El gráfico 6 muestra la representación de los resultados obtenidos en la herramienta de medición agrupados en los intervalos definidos. Se obtiene que el 58% de las clases tienen menos de 3 dependencias por lo que encapsulan la responsabilidad de su funcionamiento. Existe un 42% que dependen de otras clases para su funcionamiento, después de un análisis se decidió que no es posible eliminar estas dependencias por la responsabilidad arquitectónica que deben mantener las clases.

Gráfico 6: Representación de la evaluación de la métrica RC.



El gráfico 7 muestra la representación en % de los resultados obtenidos en la herramienta agrupados en los intervalos definidos.

Gráfico 7: Representación en % de la evaluación de la métrica RC.



- Acoplamiento.

La tabla 15 y el gráfico 8 muestran la representación de la incidencia de los resultados de la evaluación de la métrica RC en el atributo acoplamiento, evidenciando un diseño aceptable. Sin embargo existe un 42% que debe ser analizado profundamente para identificar si es posible o no disminuir el acoplamiento y así ganar en modularidad.

Tabla 15: Acoplamiento

Acoplamiento	Cantidad de clases	Promedio
Ninguno.	10	32,25806452
Bajo.	5	16,12903226
Medio.	3	9,677419355
Alto.	13	41,93548387

Gráfico 8: Representación de la evaluación de la métrica RC en el atributo Acoplamiento.

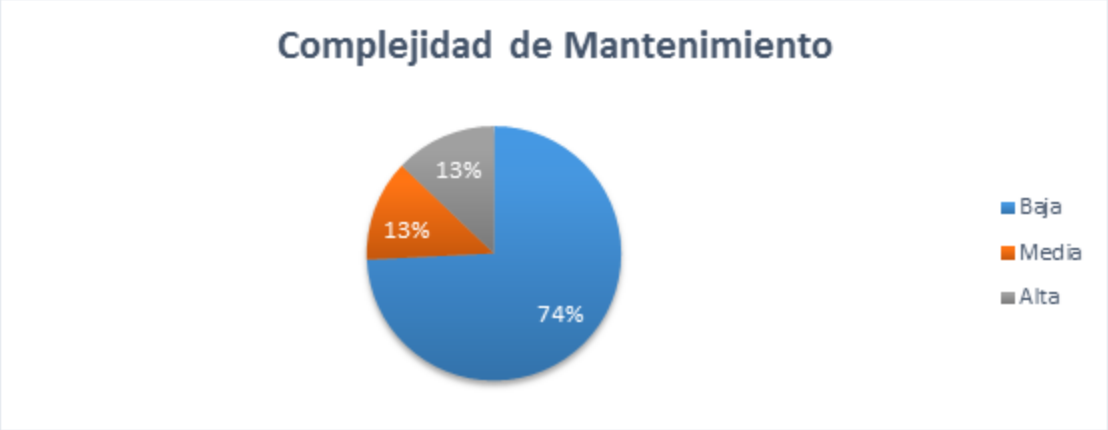
- Complejidad de mantenimiento.

La tabla 16 y el gráfico 9 exponen la representación de la incidencia de los resultados de la evaluación de la métrica RC en este atributo, mostrando la eficiencia de la arquitectura del sistema al evidenciarse una baja complejidad de mantenimiento o soporte.

Tabla 16: Complejidad de mantenimiento

Complejidad de mantenimiento	Cantidad de clases	Promedio
Baja.	23	74,19354839
Media.	4	12,90322581
Alta.	4	12,90322581

Gráfico 9: Representación de la evaluación de la métrica RC en el atributo Complejidad de mantenimiento.



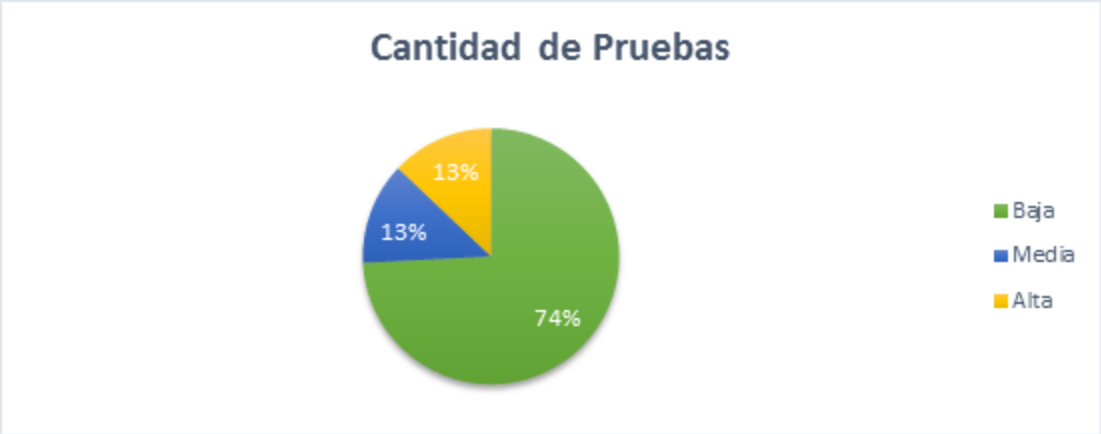
- Cantidad de pruebas.

La tabla 17 y el gráfico 10 muestran la representación de la incidencia de los resultados de la evaluación de la métrica RC en este atributo, se puede apreciar que un 74% de las clases requieren un bajo grado de esfuerzo para efectuarles las pruebas de software, ahorrando tiempo en el desarrollo del sistema.

Tabla 17: Cantidad de pruebas

Cantidad de pruebas	Cantidad de clases	Promedio
Baja.	23	74,19354839
Media.	4	12,90322581
Alta.	4	12,90322581

Gráfico 10: Representación de la evaluación de la métrica RC en el atributo Cantidad de pruebas.



- Reutilización.

La tabla 18 y el Gráfico 11 muestran la representación de la incidencia de los resultados de la evaluación de la métrica RC en este atributo. Se evidencia que un 74% de las clases tienen un grado de reutilización alto, permitiendo que el diseño del sistema sea aceptable.

Tabla 18: Reutilización

Reutilización	Cantidad de clases	Promedio
Baja.	4	12,90322581
Media.	4	12,90322581
Alta.	23	74,19354839

Gráfico 11: Representación de la evaluación de la métrica RC en el atributo Reutilización

Haciendo un análisis de los resultados obtenidos en la herramienta de medición de la métrica RC, se puede concluir que el diseño del módulo Estructura y Composición está entre los niveles de calidad requeridos, pudiéndose observar que el 58% de las clases

posee menos de 3 dependencias con otras clases, este mismo por ciento posee índices aceptables en cuanto al atributo acoplamiento. Así mismo los atributos de calidad complejidad de mantenimiento, reutilización y cantidad de pruebas se comportan satisfactoriamente con un 74% de valor en las clases, ratificando el diseño aceptable de la solución propuesta.

3.2. Pruebas de software

Las pruebas son un conjunto de actividades que se pueden planificar por adelantado y llevar a cabo sistemáticamente. Por esta razón, se define en el proceso de ingeniería de software una plantilla para las pruebas como: "un conjunto de pasos en los que podamos situar los métodos específicos de diseño de casos de prueba" (Pressman, 2010).

Es importante considerar que las pruebas no pueden demostrar que el software está libre de defectos o que se comportará en todo momento como está especificado. Siempre es posible que una prueba que se haya pasado por alto pueda descubrir problemas adicionales con el sistema. *"Las pruebas sólo pueden demostrar la presencia de errores, no su ausencia"* (Dijkstra, 1972).

"Un caso de prueba es un conjunto de entradas, condiciones de ejecución y resultados esperados, desarrollado para conseguir un objetivo particular o condición de prueba como, por ejemplo, verificar el cumplimiento de un requisito específico" (Laboratorio Nacional de Calidad del Software, 2009).

Los casos de prueba especifican una forma de probar el sistema, incluyendo las entradas con las que se ha de probar, las condiciones bajo las que ha de probarse, así como los resultados esperados. Verifican el resultado de la interacción entre los actores y el sistema, validando así los requisitos funcionales que debe satisfacer el mismo. Este artefacto es generado durante el transcurso del desarrollo del software.

El diseño de casos de prueba para la verificación del software puede significar un esfuerzo considerable (cerca del 40% del tiempo total de desarrollo). Los requisitos son la fuente principal para obtener los casos de prueba, para el desarrollo del módulo Estructura y Composición se obtuvieron un total de 27 casos de prueba respondiendo cada uno a los requisitos funcionales definidos.

3.2.1. Pruebas de caja negra

El enfoque de prueba aplicado en la presente investigación fue el funcional o de caja negra, el cual se refiere a las pruebas que se llevan a cabo sobre la interfaz del software, sin tener mucho en cuenta la estructura interna del software, es decir el código de la

solución. Dichas pruebas tratan al software como una caja negra con entradas y salidas, pero no tienen conocimiento de cómo está estructurado el programa o componente dentro de la caja. Este enfoque pretende demostrar, mediante los casos de prueba, que las funciones del software son operativas, que la entrada se acepta de forma adecuada, que se produce un resultado correcto, y la mantención de la integridad de la información externa. Este tipo de prueba es un enfoque complementario que intenta descubrir diferentes tipos de errores como por ejemplo errores de interfaz, errores en estructuras de datos o acceso a las bases de datos entre otros.

Las pruebas de caja negra se centran en los requisitos funcionales del software, intentan encontrar errores de los siguientes tipos:

- Funciones incorrectas o inexistentes.
- Errores relativos a las interfaces.
- Errores en estructuras de datos o en accesos a bases de datos externas.
- Errores debidos al rendimiento.
- Errores de inicialización o terminación.

Figura 14: Modelación de la prueba de caja negra

Para este tipo de pruebas se empleó la técnica de Partición de Equivalencia la cual consiste en dividir un conjunto de condiciones de prueba en grupos o conjuntos que puedan ser considerados iguales por el sistema. Esta técnica requiere probar sólo una condición para cada partición. Esto es así porque se asume que todas las condiciones de una partición serán tratadas de la misma manera por el software. Si una condición en una partición funciona, se asume que todas las condiciones en esa partición funcionarán. De la misma forma, si una de las condiciones en una partición no funciona, entonces se asume que ninguna de las condiciones en esta partición en concreto funcionará. Una partición de equivalencia representa un conjunto de estados válidos y no válidos para una condición de entrada. El diseño de los casos de prueba para la

partición de equivalencia se basa en la evaluación de las clases de equivalencia (Laboratorio Nacional de Calidad del Software, 2009).

Dentro de las pruebas de caja negra la técnica de la Partición de Equivalencia es una de las más efectivas, pues permite examinar los valores válidos e inválidos de las entradas existentes en el software, descubre de forma inmediata una clase de errores que, de otro modo, requerirían la ejecución de muchos casos antes de detectar el error genérico. La Partición de Equivalencia se dirige a la definición de casos de pruebas que descubran clases de errores, reduciendo así el número de casos de prueba que hay que desarrollar.

3.2.2. Resultados de las pruebas de caja negra

En las pruebas realizadas al sistema se detectaron un total de 34 no conformidades. La tabla 19 refleja la cantidad de no conformidades detectadas por iteración.

Tabla 19: Resumen de No Conformidades por iteraciones

Iteraciones	Cantidad de no conformidades	Significativas	No significativas
Primera Iteración	27	7	20
Segunda Iteración	7	2	5
Tercera Iteración	0	0	0
Total	34	9	25

Las no conformidades fueron mayormente errores de faltas de ortografías en las interfaces, de validaciones, los mensajes de información estaba mal elaborados, faltaban los iconos de los botones, entre otros. Después de corregir los errores encontrados en las pruebas, se pudo evidenciar que el flujo de trabajo de las interfaces estaba correcto ya que cumplía con las condiciones necesarias que se habían definido para las funcionalidades descritas.

3.3. Conclusiones parciales

En este capítulo se definieron algunos elementos que intervienen en el desarrollo de la solución propuesta, se aplicaron las métricas: Tamaño Operacional de la Clase y Relaciones entre Clases para evaluar la calidad el diseño propuesto. Los resultados arrojados por dichas métricas permitieron evaluar varios factores claves como la reutilización, complejidad de la implementación, entre otros, de los cuales se obtuvo para las métricas TOC y RC una calidad aceptable.

También se efectuaron pruebas de caja negra al módulo Estructura y Composición para verificar su correcto funcionamiento, empleando la técnica de Partición de Equivalencia. Después de 3 iteraciones de realización de pruebas funcionales se detectaron un total de 34 no conformidades. Estas fueron resueltas, permitiendo cumplir con los requisitos capturados en la primera etapa de desarrollo.

Los resultados de las pruebas realizadas al sistema y la aplicación de las métricas en ambos casos descritos anteriormente dan lugar a que la implementación realizada presenta una calidad admisible, siendo satisfactoria la validación de la solución propuesta.

3.4. Conclusiones generales

Durante el desarrollo del presente trabajo investigativo se llevaron a cabo una serie de fases que permitieron dar cumplimiento al objetivo general planteado y que abarcaron todas las tareas investigativas propuestas.

Se elaboró el marco teórico referencial de la investigación mediante la consulta de bibliografía nacional e internacional relacionada con la gestión de estructuras organizacionales, evidenciándose de esta manera la no existencia de una solución informática para el marco de trabajo Symfony 2, capaz de ejecutar las funcionalidades referentes al proceso, ni de cumplir con los requisitos establecidos a nivel nacional en las empresas cubanas.

Se obtuvo una nueva versión del módulo Estructura y Composición desarrollada en Symfony 2, la cual ayudará al cliente a definir y organizar mejor la estructura organizacional de una empresa, garantizando que el mismo tenga el conocimiento suficiente de las relaciones existentes en cada una de las estructuras que se gestionan. La modelación y construcción de la solución propuesta viabiliza la creación, modificación y eliminación de estructuras organizacionales de manera dinámica, posibilitando la gestión eficiente de las mismas.

Para validar el correcto funcionamiento del módulo se aplicaron pruebas de caja negra, específicamente se empleó la técnica de Partición de Equivalencias, arrojando resultados satisfactorios. También se realizó la validación del diseño a través de las métricas TOC y RC, con el objetivo de comprobar la calidad del mismo, obteniéndose resultados positivos. Esto apoya el hecho de que la implementación desarrollada se puede considerar como aceptable, y el código puede ser reutilizado en futuras implementaciones.

4. RECOMENDACIONES

Se recomienda incluir en próximas versiones del sistema los siguientes aspectos:

- Permitir la gestión de nuevos nomencladores que se puedan asociar a los niveles estructurales de manera dinámica.
- Implementar en versiones futuras del sistema las funcionalidades relacionadas con las Subordinaciones y Recuperaciones, las cuales posee el marco de trabajo Sauxe.
- Implementar en versiones futuras del sistema una funcionalidad que permita mostrar gráficamente el organigrama de una empresa determinada.

5. BIBLIOGRAFÍA

1. **Apache Software Foundation.** Sitio oficial de Apache. [En línea] http://www.apache.org.

2. **Baryolo, Oiner Gómez. 2012.** *Modelo de control de acceso para sistemas de información en entronos multidominios.* La Habana : s.n., 2012.

3. **—. 2011.** *SOLUCIÓN INFORMÁTICA DE AUTORIZACIÓN EN ENTORNOS MULTIENTIDAD Y MULTISISTEMA.* Ciudad de la Habana : s.n., 2011.

4. **Centro de Informatización de Gestión de Entidades. 2012.** *MODELO DE DESARROLLO DE SOFTWARE.* La Habana : s.n., 2012.

5. **Clements, Paul. 1996.** *Coming attractions in Software Architecture.* 1996.

6. **Colectivo de autores. 2007.** Conceptos sobre estructura organizacional. [En línea] 10 de 2007. http://admindeempresas.blogspot.com/2007/10/conceptos-sobre-estructura.html.

7. **—. 2014.** Definiciones. [En línea] 2014. http://definicion.de/organizacion/.

8. **—. 2012.** *Doctrine ORM for PHP. Guide to Doctrine for PHP.* 2012.

9. **—. 2011.** *Manual de PHP.* s.l. : PHP Documentation Group , 2011.

10. **—. 2010.** *Tutorial de paAdmin.* 2010.

11. **Comité de la Práctica Profesional del IEEE Computer Society. 2004.** *Swebok.* Los Alamitos, California : IEEE Computer Society, 2004. ISBN 0-7695-2330-7.

12. **Comunidad de software libre de la UCI.** humanos. [En línea] http://humanos.uci.cu/.

13. **Comunidad Mozilla Firefox de Cuba.** Firefoxmanía. [En línea] http://firefoxmania.uci.cu/.

14. **Cortes, Hector. 2006.** *SAP Modulo RH.* BOGOTÁ : Colombia Creative Commons, 2006.

15. *Definición de pautas para la creación de un modelo de componentes.* **Torres, Abraham Calás. 2012.** 12, La Habana : Serie Científica de la Universidad de las Ciencias Informáticas, 2012, Vol. 5. ISSN/RNPS.

16. *Desarrollo de funcionalidades que faciliten al docente su preparación y el control del aprendizaje de los estudiantes en la plataforma educativa Zera.* **Pérez, Irina Ivis Santiesteban, y otros. 2011.** 11, La Habana : Serie Científica de la Universidad de las Ciencias Informáticas, 2011, Vol. 4. ISSN/RNPS .

17. **Dijkstra, Edsger. 1972.** *Notes on Structured Programming.* 1972.

18. **Eguiluz, Javier. 2011.** *Desarrollo web ágil con Symfony2.* 2011.

19. *Extensión del modelo RBAC para sistemas ERP en entornos multientidad.* **Baryolo, Oiner Gómez, Mendoza, Yoel Hernández y Pérez, Mileidy M. Sarduy. 2011.** 8, La

Habana : Serie Científica de la Universidad de las Ciencias Informáticas, 2011, Vol. 4. ISBN/ISSN.

20. **2014.** FORMAS Y TIPOS DE ESTRUCTURAS ORGANIZACIONALES. [En línea] 2 de 2014. http://ssfe.itorizaba.edu.mx/bvirtualindustrial/index.php/administracion-gerencial/1924-112-formas-y-tipos-de-estructuras-organizacionales.

21. **Fundación de código libre dominicano. 2007.** *Introduccion a Base de Datos con PostgreSQL.* 2007.

22. **Gamma, Erich, y otros. 1995.** *Design Patterns: Elements of Reusable Object-Oriented Software.* 1995. ISBN 0-201-63361-2.

23. **Gómez, Dr. Orlando Salinas. 2012.** *COMPONENTE PROCESOS ORGANIZACIONALES.* Bogotá : s.n., 2012.

24. **Hamon, Hugo. 2014.** *Identifying Design Patterns in the Symfony framework.* Estambul, Turkía : s.n., 2014.

25. **Hintze, Dr. Jorge. 2011.** *Administración de estructuras organizativas.* 2011.

26. **Inscale GmbH & Co. KG. 2007.** *Patrones en la arquitectura.* 2007.

27. **Iranmanesh, Z. 2008.** *A Logic for Multi-Domain Authorization Considering Administrators.* NY, USA : Workshop on Policies for Distributed Systems and Networks, Palisades, IEEE Computer Society, 2008.

28. **Joshi, J. y Ghafoor, A. 2001.** *"Digital Government Security Infrastructure Design Challenges".* NY, USA : IEEE Computer Society, 2001.

29. **Küng, Stefan, Onken, Lübbe y Large, Simon. 2013.** *RapidSVN a Subversion graphics client for Linux.* 2013.

30. **Laboratorio Nacional de Calidad del Software. 2009.** *GUÍA DE VALIDACIÓN Y VERIFICACIÓN.* España : s.n., 2009.

31. **Larman, Craig. 1999.** *UML y Patrones.Introducción al análisis y diseño orientado a objetos.* México : Prentice Hall Hispanoamericana, S.A, 1999. ISBN.

32. **Morales, Yanet Rosales, Marrero, Michael Eduardo y Oliva, Adrian Trujillo. 2013.** *Extensión de la herramienta Visual Paradigm para la generación de clases de acceso a datos con Doctrine 2.0.* La Habana : Serie Científica de la Universidad de las Ciencias Informáticas, 2013. ISSN: 2306-2495 | RNPS: 2343.

33. *NetBeans IDE. La alternativa a Eclipse.* **Domínguez-Dorado, M. 2005.** 13, Madrid : Editorial Iberprensa, 2005.

34. **Pacheco, Nacho. 2011.** *Manual de Twig .* 2011.

35. **Piñera, Yadira, Pérez, Yinet y Pérez, Terán. 2011.** *Módulo Estructura y Composición. Manual de usuario.* La Habana : s.n., 2011.

36. **Potencier, Fabien. 2011.** *Guía de inicio rápido.* 2011.

37. **Pressman, Roger S. 2010.** *Ingeniería de Software. Un enfoque práctico.* New York : Mac Graw Hill. Higher Education, 2010. ISBN.

38. **Quintero, Juan Bernardo. 2010.** *Arquitectura de Software. Patrones en la arquitectura.* [Documento pdf] 2010.

39. **Reynoso, Carlos Billy. 2004.** *Introducción a la arquitectura de software.* Buenos Aires, Argentina : s.n., 2004.

40. **Robaina, Leydisbel Jaime y Zamora, Yaniris Blanco. 2009.** *Solución Informática para el Módulo Estructura y Composición del sistema Cedrux.* Ciudad de la Habana : s.n., 2009.

41. **Rodríguez, Juanda. 2014.** Tutorial de Symfony2. [En línea] 2014. http://juandarodriguez.es/tutoriales/tutorial-de-symfony2.

42. **Sánchez, Jorge. 2003.** *Manual de referencia.* 2003.

43. **Schmuller, Joseph. 2000.** *Aprediendo UML en 24 horas.* México : Pearson Educacion, 2000. ISBN.

44. **Shafiq, B. y Ghafoor, Professor Arif. 2006.** *Access Control Management and Security in Multi-Domain Collaborative.* West Lafayette, Indiana, USA, Purdue University : Center for Education and Research in Information Assurance and Security, 2006.

45. **Sommerville, Ian. 2005.** *Ingeniería de Software.* Madrid : Pearson Educacion S.A., 2005. ISBN.

46. **SOPORTE LOGICO Ltda. 2010.** *MANUAL DE USUARIO MÓDULO ESTRUCTURA ORGANIZACIONAL.* Bogota : s.n., 2010. PBX 3403270.

47. **Torres, Abraham Calás. 2012.** *Definición de pautas para la creación de un modelo de componentes para el marco de trabajo sauxe.* La Habana : Serie Científica de la Universidad de las Ciencias Informáticas, 2012. ISSN:2306-2495/ RNPS:2343.

48. **Yong, H. 2010.** *Reputation and Role Based Access Control Model for Multi-domain Environments.* s.l. : International Symposium on Intelligence Information Processing and Trusted Computing, Huanggang, IEEE Computer Society, 2010.

Agrupación lógica: Es una clasificación de estructuras que cumplen una condición común, Ej. Hospitales, Ministerios.

Agrupación: Una Agrupación es un grupo empresarial, unión o estructuras similares adscriptas a un Nivel 1, a las que se le subordinan entidades o unidades las cuales están relacionadas entre sí.

Área: Las áreas son divisiones internas de las unidades, es una forma de organizar el personal de la empresa que se relacionan entre sí, que hacen actividades similares o que cumplen un objetivo específico.

Campo: Un campo es una característica inherente de los niveles estructurales que se crean dinámicamente los cuales son necesarios guardar en la base de datos, por lo que es necesario gestionarlos dinámicamente también.

CAP: Consejos de la Administración Provinciales.

Composición: Los elementos que puede tener una estructura que no sea concepto. Una estructura tiene composición si puede tener asociado estructuras internas.

Concepto: Son niveles de la estructura que no tienen composición.

Entidad: Una entidad es una empresa, unidad presupuestada u otro tipo de organización similar tiene gestión económica, financiera, organizativa, técnica, productiva, comercial, laboral y contractual con autonomía controlada en cumplimiento de lo establecido por el Gobierno y el Estado.

Escala salarial: La escala salarial es un concepto que define como se comporta el salario de un trabajador, según la escala salarial que se le asigne.

Estructura externa: Son las estructuras que no tienen asociado directamente los cargos.

Estructura interna: Son las estructuras que tienen asociado directamente los cargos y medios.

Nivel 1: Es el primer nivel en el árbol de estructuras, puede tomar forma de OACE, Órganos estatales, CAP, organizaciones políticas, organizaciones de masas, asociaciones profesionales, otras asociaciones nacionales.

OACE: Organismos de la Administración Central del Estado.

ONAT: Oficina Nacional de Administración Tributaria.

ONE: Oficina Nacional de Estadística.

Raíz: Es el nivel de la estructura que no se subordina a ningún otro.

Responsabilidad: Una responsabilidad es una función de dirección intermedia asignada a un cargo. Ejemplo: Jefe de brigada, especialista principal etc.

Unidad: Las unidades son divisiones internas, que se crean para organizar los procesos de producción de bienes y servicios, actúan con independencia relativa y no cuentan con personalidad jurídica propia.