Universidad de Ciencias Informáticas
Facultad 4

“Vistas de arquitectura de sistema y de datos de las soluciones Estructura y Composición, Configuración y Multimoneda del proyecto ERP Cuba”

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

Autora: Jeanne Omara González Rivas
Tutor(es): Ing. Nemury Silega Martínez
Ing. Leandro Perezborroto Vivero

Ciudad de la Habana, junio del 2010

Lo fundamental es que seamos capaces de hacer cada día algo que perfeccione lo que hicimos el día anterior.

Declaración de autoría

Declaro que soy el único autor de este trabajo y autorizo al Centro para la Informatización de la Gestión de Entidades, de la Universidad de las Ciencias Informáticas, así como a dicho centro, a hacer uso que estimen pertinente del mismo.
Para que así conste firmo la presente a los _____ días del mes de _____ del año _____.

Autora:

_____

Jeanne O. González Rivas

Tutor(es):

_____

Ing. Nemury Silega Martínez

_____

Ing. Leandro Perezborroto Vivero

Agradecimientos

Dedicatoria

Resumen

La descripción de la arquitectura de software constituye la especificación de los diferentes elementos arquitectónicos que conforman un sistema, de las relaciones de integración entre los mismos y de los patrones arquitectónicos utilizados en su diseño y desarrollo. Es por ello que la misma resulta ser de gran importancia, pues mediante sus vistas, sirve de apoyo para comprender mejor el sistema, darle soporte y facilita la comunicación entre las partes interesadas sobre el mismo.

En el presente trabajo se detallaron específicamente las Vistas de arquitectura de sistema y de datos de las soluciones Estructura y Composición, Configuración y Multimoneda, del proyecto ERP Cuba, por medio de la realización de una serie de artefactos que describen la organización de estos subsistemas, en función de los componentes que contienen, especificando a su vez, las funcionalidades y la integración interna que presentan dichos componentes.

El desarrollo de estas vistas permite elevar el entendimiento y comprensión de estas soluciones, por parte del personal involucrado en el proyecto, y por consiguiente brindar un mejor mantenimiento al sistema y realizar nuevas versiones del mismo, elevando a su vez, la calidad de desarrollo.

Palabras claves: descripción de la arquitectura, vistas, componentes, integración.

Índice

Introducción

En la actualidad, el uso de sofisticados sistemas de información integrados para la planificación de recursos empresariales, conocidos como los sistemas ERP, adquiere mayor auge y relevancia en el ámbito empresarial, puesto que constituyen sistemas integrales de gestión de información que integran y automatizan muchos de los negocios asociados con los aspectos operativos o productivos de una empresa ligada a la producción de bienes o servicios.

Estos sistemas brindan soluciones prácticas e integrales a problemas reales, se configuran como una herramienta de gestión y otorgan ventajas competitivas a las organizaciones dentro del mercado empresarial, revolucionando por completo la manera de hacer negocios en el mismo. El hecho de que estén compuestos por diferentes módulos integrados en una única aplicación, donde a su vez queda integrada toda la información que se utiliza para diferentes funciones y departamentos, en una entidad, contribuye a obtener un acceso más completo a la información, a la disminución de los errores y a una mayor rapidez y eficiencia.

Para Cuba también es muy importante la planificación de los recursos y gestión de la información, además de que se necesita aumentar la productividad, minimizando los costos y mejorar la calidad y el servicio al cliente. Por tanto la dirección del país ha decidido desarrollar un sistema ERP propio que pueda aplicarse a las distintas ramas de la economía y de la sociedad cubana.

Algunas de las peculiaridades que debe cumplir el Sistema Integral de Gestión de Entidades (en lo adelante CEDRUX) son: que sea multiendidad, o sea, que tenga la capacidad de soportar una estructura que contenga muchas entidades; que sea adaptable a las características de cada empresa y que permita el uso de la doble moneda en las transacciones económicas y su registro, pues el país difiere de otras naciones, en cuanto al manejo de los temas monetarios y el uso de la dualidad monetaria. A estas características responden las soluciones de Estructura y Composición, Configuración y Multimoneda, respectivamente. Es importante destacar que la inclusión del módulo Estructura y Composición en el sistema CEDRUX, es novedosa, pues otras aplicaciones de este tipo a nivel internacional no presentan esta solución dentro de sus módulos básicos.

Para todo sistema software grande y complejo, como lo es CEDRUX, es necesario definir y diseñar una arquitectura sólida que pueda soportar el ámbito del sistema. Si el diseño arquitectónico de los subsistemas referenciados no cuenta con una descripción coherente y precisa, no puede comprobarse si el diseño es escalable, flexible. Tampoco es posible proporcionar un adecuado mantenimiento a la base de datos, a los subsistemas y sus componentes. Se tendría desconocimiento sobre el nivel de complejidad de dichos componentes, por tanto estos no se podrían reutilizar en nuevas soluciones similares a estas. También resulta difícil revisar la integración de los mismos, así como el acoplamiento; lo cual no permite dar soporte a un mínimo de dependencia y a un aumento de reusabilidad. De igual modo se entorpece la revisión del nivel de integración de los datos y del flujo de información. Por otra parte si se quisiera hacer un rediseño de estas soluciones, se tornaría un proceso complejo y costoso, igualmente lo sería, desarrollar nuevas versiones del sistema y proveerles soporte en el futuro. Además se dificultan las relaciones comunicativas dentro del equipo de trabajo. En fin, impide realizar un adecuado seguimiento al desarrollo y evolución de la aplicación.

Por todo lo anteriormente expuesto, el **problema a resolver** por esta investigación se centra en que el estado actual de la descripción arquitectónica de sistema y de datos de las soluciones Estructura y Composición, Configuración y Multimoneda, afecta la capacidad de entendimiento del personal involucrado y por consiguiente el soporte y desarrollo de nuevas versiones del sistema.

Se propone como **objeto de estudio** la Arquitectura de sistema y de datos y se especifica como **campo de acción** la Arquitectura de sistema y de datos de la soluciones Estructura y Composición, Configuración y Multimoneda del proyecto ERP Cuba.

El presente trabajo de diploma tiene como objetivo general realizar la vista de arquitectura de sistema y de datos de las soluciones Estructura y Composición, Configuración y Multimoneda del proyecto ERP Cuba; y se desglosa en los siguientes objetivos específicos:

- ✓ Determinar el marco teórico de la investigación.
- ✓ Realizar la vista de arquitectura de sistema de las soluciones Estructura y Composición, Configuración y Multimoneda.

- ✓ Realizar la vista de arquitectura de datos de las soluciones Estructura y Composición, Configuración y Multimoneda.

Idea a defender: Si se realizan la vista de sistema y de datos de las soluciones Estructura y Composición, Configuración y Multimoneda se facilita el entendimiento de estas soluciones por el personal involucrado y apoyan el soporte y desarrollo de nuevas versiones del sistema.

Tareas de investigación:

1. Buscar y sintetizar bibliografía sobre Arquitectura de software, Metodologías de desarrollo, Patrones arquitectónicos, etcétera.
2. Estudiar los temas y buenas prácticas que apoyan el diseño arquitectónico de un sistema.
3. Estudiar elementos necesarios y fundamentales para describir una arquitectura.
4. Realizar un análisis de los artefactos generados en el negocio.
5. Describir los artefactos de sistema y de datos.
6. Realizar la línea base de subsistema.
7. Realizar la matriz de complejidad y criticidad de los componentes de estos subsistemas.
8. Realizar la especificación de los componentes de estos subsistemas.
9. Realizar la matriz de integración entre los componentes dentro de cada uno de estos subsistemas.
10. Describir los modelos de datos de los subsistemas referenciados.
11. Realizar el diccionario de datos.
12. Realizar la matriz de trazabilidad de los datos.

Métodos teóricos de investigación utilizados:

- ⇨ Histórico – Lógico: se empleó para recopilar información, estudiar y conocer las tendencias más relevantes que existen en cuanto al proceso de desarrollo y los elementos que integran el diseño arquitectónico de un sistema; que permitiera valorar y determinar la soluciones óptimas para el problema en cuestión.

El contenido del trabajo está estructurado en tres capítulos. En el capítulo I se realiza la fundamentación teórica de la investigación; incluye un estudio del estado del arte, sobre los estilos arquitectónicos, los

patrones de diseño y algunas de las metodologías mayormente utilizadas en el diseño de arquitectura en los últimos tiempos; se tratan aspectos relevantes sobre el enfoque de desarrollo orientado a componentes, y sobre los métodos que se emplean para la descripción de una arquitectura. El capítulo II abarca la vista de arquitectura de sistema de los subsistemas Estructura y Composición, Configuración y Multimoneda, y el capítulo III comprende la vista de arquitectura de datos de los subsistemas Estructura y Composición, Configuración y Multimoneda.

Capítulo I Fundamentación Teórica

1.1. Introducción

En este capítulo se tratan aspectos importantes relacionados con el proceso de desarrollo y la arquitectura de software. Se describen brevemente algunas de las metodologías de desarrollo y estilos arquitectónicos más utilizados en la actualidad, así como los patrones de diseño; caracterizando de igual modo la metodología, los patrones de diseño y los estilos empleados en el diseño arquitectónico del sistema CEDRUX. También se abordan características importantes sobre el desarrollo basado en componentes, siendo esta una de las tecnologías más novedosas en la construcción de aplicaciones software, y sobre la descripción de una arquitectura.

1.2. Proceso de desarrollo de software

Un proceso de desarrollo de software tiene como propósito la producción eficaz y eficiente de un producto software que cumpla con los requisitos del cliente. Se define como un marco de trabajo de las tareas que se requieren para construir software de alta calidad. El proceso consiste en traducir las necesidades del usuario en requisitos de software, la transformación de los requisitos en el diseño, la implementación del diseño en el código, la verificación del código y en ocasiones la instalación y verificación del software para uso operativo. (1). Estas actividades están implícitas en todos los procesos software a pesar de la diversidad que existe sobre estos. También está presente como otra actividad fundamental, la evolución del software, para que pueda adaptarse a las necesidades del usuario. Dichas actividades pueden realizarse de forma iterativa (1) y son aplicables a todos los proyectos independientemente de su tamaño o complejidad.

Algunas personas consideran que las empresas profesionales deberían organizarse en torno a las habilidades de individuos altamente capacitados, que hacen muy bien el trabajo y raramente requieren dirección sobre los procedimientos de la organización en que trabajan. Pero en la generalidad de los casos es un gran desacierto y más grave aún, en el caso del desarrollo de software. Aunque los desarrolladores estén altamente cualificados necesitan un modelo o método común que brinde un conjunto de procedimientos, técnicas y herramientas que los ayuden en la construcción del software; o sea una metodología de desarrollo.

1.2.1. Metodología de desarrollo

Las metodologías no son más que marcos de trabajo utilizados para estructurar, planificar y controlar el proceso de desarrollo en los sistemas de información. Se basan en los modelos de procesos genéricos y adicionalmente brindan una guía para ordenar las actividades del equipo de trabajo, dirigir las tareas, definir los artefactos que se deben realizar y establecer criterios para el control y medición de los productos y actividades del proyecto. Por tanto de ellas también depende el éxito en el desarrollo de software. Existen disímiles propuestas metodológicas que pueden emplearse en la construcción de software. Por ejemplo:

- **Metodología: Programación Extrema**

La Programación Extrema (en lo adelante XP) es una de las más populares, dentro de los procesos ágiles de desarrollo de software. Consiste en una programación rápida o extrema y es un método útil para proyectos de alto riesgo, donde los requisitos son aún ambiguos o cambian rápidamente. Establece la integración del cliente en todo el proceso de desarrollo, la programación en parejas y el trabajo en equipo, así como la adaptabilidad de los procesos en desarrollo. Para ello XP se basa en los principios de la retroalimentación continua entre el cliente y el equipo de desarrollo, de la simplicidad de las soluciones implementadas, del valor para enfrentar los cambios, de una comunicación fluida y del respeto entre todo el personal involucrado, así como el respeto por su trabajo. Por tanto XP se considera una metodología eficiente, flexible y de bajo riesgo.

- **Metodología: Proceso Unificado de Desarrollo (RUP)**

El Proceso Unificado de Desarrollo (*Rational Unified Process* en inglés) constituye la metodología estándar más utilizada en el análisis, diseño, implementación y documentación de sistemas orientados a objetos. Se caracteriza por ser un proceso dirigido por casos de uso, centrado en la arquitectura, iterativo e incremental. En RUP se puede ver la evolución del software en cuatro fases, al final de las cuales y tras una serie de iteraciones, establece objetivos a alcanzar muy bien definidos. Además es un proceso configurable al contexto y necesidades de cada organización y proyecto. Por otra parte, RUP se basa en lo que se consideran las seis mejores prácticas del desarrollo de software: Desarrollo de software de

forma iterativa, la Gestión de requerimientos, el Uso de Arquitecturas basadas en componentes, la Modelización visual del software, la Verificación de calidad del software y el Control de cambios. No en vano es calificado como el proceso de desarrollo más general de los existentes actualmente.

- **Modelo de desarrollo adoptado para el proyecto ERP Cuba**

Para el desarrollo del sistema CEDRUX se requería de una metodología específica que respondiera a sus peculiaridades y magnitud. Por tanto se utilizó el Modelo de desarrollo orientado a componentes del proyecto ERP Cuba, definido por el departamento de Tecnología del Centro para la Informatización de la Gestión de Entidades (en lo adelante CEIGE), en colaboración con las líneas de desarrollo del proyecto, acorde a las necesidades que han presentado cada una de ellas y teniendo en cuenta los principales riesgos con los que se cuentan en el proyecto. (2) Este modelo está basado en varias metodologías como RUP y XP. Se caracteriza por ser:

- Centrado en la arquitectura. La arquitectura determina la línea base, los elementos de software estructurales a partir de los elementos de la arquitectura de negocio. Interviene en la gestión de cambios y diseña la evolución e integración del producto. La arquitectura orienta las prioridades del desarrollo y resuelve las necesidades tecnológicas y de soporte para el desarrollo. (3)
- Orientado a componentes. Las iteraciones son orientadas por el nivel de significancia arquitectónicas de los componentes, los mismos son abstracciones arquitectónicas de los procesos de negocio y requisitos asociados que modelan, el componente es la unidad de medición y ordenamiento de las iteraciones. (3)
- Iterativo e incremental. Las iteraciones son planificadas y coordinadas con el equipo de arquitectura, los clientes y la alta gerencia. Cada iteración constituye el desarrollo de componentes, los cuales son integrados al término de la integración, permitiendo de esta manera la evolución incremental del producto. (3)
- Ágil y adaptable al cambio. El desarrollo de las partes formaliza solamente las características principales de la solución, priorizando los talleres y las comunicaciones entre las personas. Los clientes y funcionales están involucrados en el proyecto y poseen parte de las responsabilidad del éxito del mismo. Los cambios son conciliados semanalmente, discutidos y aprobados. (3)

Este modelo concuerda con el propósito fundamental que define una metodología, que especifica **quién**[1], debe hacer **qué**[2], **cómo** y **cuándo**[3] debe hacerlo; puesto que en el mismo se encuentran definidos, entre otras cosas, los diferentes roles involucrados y las responsabilidades de cada uno de ellos, las actividades a realizar, el flujo de las mismas y los artefactos que deben ser generados, conjuntamente a los roles que intervienen en el cumplimiento de ambos. Además se ajusta a la dinámica del proyecto y rige adecuadamente el desarrollo y evolución del mismo.

1.3. Estudio de sistemas ERP existentes en el mundo

- **SAP R/3:**

Sistema creado por SAP AG[4], que se define como un software abierto, basado en la tecnología cliente/servidor, diseñado para manejar las necesidades de información de una empresa. Es un sistema de gestión altamente integrado, flexible y adaptable, que coordina las distintas estructuras, procesos y eventos de todos los departamentos y áreas funcionales de una empresa. Provee herramientas para el desarrollo y configuración acorde a los requisitos y el contexto de cada empresa. También incorpora un programa para la conversión capaz de trabajar con diferentes tipos de cambio (funcionalidad multidivisa), por lo que es capaz de diferenciar entre la moneda local (moneda contable) y la moneda de transacción (moneda de entrada). La combinación de estructuras organizativas y funciones informáticas on-line, ya sea por medio de internet o intranet, conlleva, si se desea, a un proceso de datos descentralizado bajo una arquitectura cliente/servidor en tres niveles distintos: nivel de presentación, nivel de aplicación y nivel de almacenamiento de datos; los cuales admiten múltiples configuraciones. La flexibilidad de este tipo de arquitectura le permite a las organizaciones obtener un óptimo uso de los recursos informáticos, así como la posibilidad de adaptar los procedimientos de negocio a los nuevos requerimientos que el mercado establece. Además le ofrece alta escalabilidad al sistema.

- **Openbravo ERP:**

[1] En este caso está representado por los roles que tienen los trabajadores del proyecto.
[2] Constituyen los artefactos que deben ser desarrollados por los trabajadores.
[3] Cómo y cuándo reflejan el conjunto de actividades y el flujo de las mismas, respectivamente.
[4] Sistemas, aplicaciones y productos para el proceso de datos. SAP AG es la mayor compañía de software de Europa, y la quinta en el mundo

Sistema integrado y modular de gestión empresarial, en software libre y basado en la Web, que fue creado por la compañía Openbravo S.L. Incluye una amplia gama de funcionalidades como: gestión de compras y almacenes, gestión de proyectos y servicios, gestión comercial, gestión económico-financiera, entre otras. Es un sistema fácil de ampliar y configurar, y lo suficientemente flexible. Multimoneda, multiesquema contable, multiorganización. Fue diseñado sobre la base de la arquitectura Modelo Vista Controlador que facilita el desacoplamiento de las áreas de desarrollo, permitiendo el crecimiento sostenible de la aplicación y una mayor facilidad en el mantenimiento del código; y sobre el Modelo de Desarrollo Dirigido[5] (MDD, del inglés Model Driven Development), que proporciona una mejor calidad del código al reducir drásticamente la codificación manual, al tiempo que mejora la productividad y eficiencia del desarrollo. Su arquitectura de desarrollo basada en modelos permite parametrizar el sistema sin programación adicional y añadir nuevas funcionalidades para adaptarlo a las características específicas de cada organización.

- **ADempiere ERP:**

Es una solución Open Source, es decir, una aplicación en código abierto, que ofrece las funcionalidades de Planificación de recursos empresariales, Administración de la Relación con los Clientes (CRM, por sus siglas en inglés) y Administración de la Cadena de Suministros (SCM, por sus siglas en inglés) para los procesos de negocio. Incorpora módulos para la gestión de los proveedores, inventarios de las empresas y solicitudes de compra, para gestionar las ventas, clientes y elaborar las facturas correspondientes. También incluye soporte para diversos idiomas, monedas, tipos de empresas, etcétera. Adempiere constituye una solución robusta y estable, basado totalmente en estándares abiertos para que los clientes no se aten a un solo proveedor. Es un aplicación cliente/servidor escrita enteramente en Java, que está diseñada para ser alojada en la Web, permite múltiples opciones de implementación y puede cambiar y adaptarse eficientemente según las necesidades del negocio, sin crear impactos negativos, aunque para un futuro está prevista la migración hacia una arquitectura de tres capas. A fin de crear un arquitectura sustentable y resistente a los cambios, utiliza los siguientes principios de diseño: Arquitectura MVC de Smalltalk[6] (desconectado del Modelo Vista Controlador), Desconexión Asincrónica de procesos vía mensajes, Motor de Reglas Explicito para implementar la lógica compleja y Transacciones seguras de

[5] MDD supone un modelo de diseño de software que depende de metadata almacenada en un diccionario para modelar el comportamiento de la aplicación.

[6] Smalltalk fue el primer sistema y uno de los primeros lenguajes puramente orientado a objetos.

fallas y recuperación. También tiene presente una Arquitectura de Objetos[7] (del inglés Object Architecture), en la cual cada objeto es tan independiente de los otros objetos como sea posible. Además este sistema utiliza el diccionario de datos, donde todas las reglas y parámetros son características de fácil ajuste, considerando las necesidades de los nuevos desarrollos que garantizan una funcionalidad más estable.

Los sistemas anteriormente descritos, entre otros que fueron analizados, desde el punto de vista del negocio, manejan de una forma u otra, acorde a sus particularidades la funcionalidad de la Multimoneda y la configuración del sistema para adaptarlo a otros contextos empresariales; sin embargo no existe ningún sistema en específico que permita gestionar toda la estructura organizativa de las entidades; algunos solamente gestionan la estructura interna. Por ejemplo el Openbravo brinda la posibilidad de definir la estructura de almacenes de una organización hasta el mínimo nivel (ubicación) y por su parte el SAP R/3, define las distintas estructuras de todos los departamentos y áreas funcionales de una empresa, se queda dentro de lo que se conoce como estructura interna. Ahora, valorándolos desde el punto de vista del diseño y desarrollo arquitectónico, es distinguible el hecho de que estos y otros sistemas, emplean tecnologías abiertas, lo cual garantiza un ciclo de vida de estas soluciones más largo que el de otras soluciones basadas en tecnologías propietarias. Además las soluciones abiertas implican que las aplicaciones puedan funcionar sobre múltiples sistemas operativos, múltiples gestores de base de datos y protocolos de comunicación.

1.4. Estilos arquitectónicos

La arquitectura de software es la organización fundamental de un sistema encarnada en sus componentes, las relaciones entre ellos y su entorno y los principios que orientan su diseño y evolución. (4). La misma establece los fundamentos para que los equipos de trabajo que participen en un proyecto software, colaboren en una línea común que permita lograr los objetivos y requerimientos del sistema de información.

[7] Object Architecture es comparada con Orientado a Objeto (Object-Oriented), o las arquitecturas tradicionales.

Dewayne Perry y Alexander Wolf[8] establecen el razonamiento de estilos de arquitectura como uno de los aspectos fundamentales de la arquitectura de software. Un estilo es un concepto descriptivo que define una forma de articulación u organización arquitectónica. (5). El conjunto de estilos cataloga las formas básicas viables de estructuras de software, mientras que las formas complejas surgen de la composición de los estilos fundamentales. Cada estilo arquitectónico conjuga los componentes que realizan una función requerida por el sistema, los conectores que permiten la comunicación y cooperación entre dichos componentes, las restricciones que definen como estos pueden integrarse en el sistema y modelos semánticos que permiten al diseñador entender las propiedades globales de un sistema para analizar las propiedades conocidas de sus partes constituyentes. Un software puede tener presente varios estilos arquitectónicos. Algunos de los más usados son:

1.4.1. Arquitectura orientada a servicios (SOA)

La Arquitectura orientada a servicios (en lo adelante SOA), es una forma de concebir el negocio en términos de servicios interconectados. Establece un marco de diseño para la integración de aplicaciones independientes de modo que desde la red pueda accederse a sus funcionalidades, las cuales se ofrecen como servicios (6) autónomos con interfaces bien definidas. La forma de implementarla habitualmente, aunque no es exclusivo, es mediante servicios web, tecnología con la que puede descomponer aplicaciones monolíticas en un conjunto de servicios e implementar esta funcionalidad en forma modular.

SOA tiene la capacidad de involucrar disímiles tecnologías, representa mejor la integración de las mismas y puede adaptarse eficientemente acorde a la demanda del negocio. Permite el desarrollo de aplicaciones más productivas, flexibles y seguras, más rápida y económicamente. Además, desde el punto de vista empresarial, contribuye a mejorar la toma de decisiones, la productividad y potencia las relaciones con clientes y proveedores.

[8] La arquitectura de software comenzó su expansión explosiva con los manifiestos de Dewayne Perry, de AT&T Bell Laboratorios de New Jersey y Alexander Wolf, de la Universidad de Colorado. Puede decirse que ambos fundaron la disciplina. Autores del libro “Foundations for the study of software architecture”.

1.4.2. Arquitectura basada en componentes

La arquitectura basada en componentes describe una aproximación de ingeniería de software al diseño y desarrollo de un sistema. Este estilo se enfoca en la descomposición del sistema en componentes funcionales o lógicos que proporcionen los servicios requeridos en la aplicación y expongan interfaces de comunicación bien definidas; lo cual provee un mayor nivel de abstracción que los principios de orientación por objetos. (7). En esta arquitectura la interfaz constituye el elemento básico de interconectividad, por lo que cada componente debe describir de forma completa las interfaces que ofrece, así como las interfaces que requiere para su operación. Además los componentes soportan algún régimen de introspección (8), de manera que su funcionalidad y propiedades puedan ser descubiertas y empleadas en tiempo de ejecución.

La evaluación dominante del estilo de componentes subraya su mayor versatilidad respecto del modelo de objetos, pero también su menor adaptabilidad comparado con el estilo orientado a servicios. (8). No obstante 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.

1.4.3. Modelo Vista Controlador (MVC)

El estilo conocido como Modelo Vista Controlador separa los datos de una aplicación, la interfaz de usuario y la lógica de control en tres clases distintas: el modelo, que administra el comportamiento y los datos del dominio de aplicación; la vista, que maneja la visualización de la información representada por el modelo y el controlador, que responde a eventos, generalmente acciones del usuario, e invoca cambios en la vista y en el modelo según resulte apropiado.

Tanto la vista como el controlador dependen del modelo, mientras que este no depende de las otras clases. Dicha separación permite realizar y probar el modelo, independientemente de la representación

visual. Todo esto también propicia que la interfaz de usuario pueda mostrar, simultáneamente, múltiples vistas de los mismos datos; al igual que la adaptación al cambio, puesto que los requerimientos de interfaz de usuario suelen cambiar con mayor rapidez que las reglas del negocio, agregar nuevas opciones de presentación no afecta al modelo generalmente. La separación entre la vista y el controlador puede ser secundaria en aplicaciones de clientes ricos, pero en aplicaciones web, la separación entre la vista (el browser) y el controlador (los componentes del lado del servidor que manejan los requerimientos de HTTP) está mucho más taxativamente definida. (8). A pesar de ofrecer un soporte de múltiples vistas y de ser adaptable al cambio, esta arquitectura también presenta la desventaja de costo de actualizaciones frecuentes, debido a que desacoplar el modelo de la vista no indica que los desarrolladores del modelo puedan ignorar la naturaleza de las vistas. (8)

Para el diseño de la arquitectura del sistema CEDRUX, se adoptaron los estilos de Arquitecturas basadas en componentes y el MVC. El uso de los mismos fue una decisión tomada por el departamento de Tecnología del CEIGE, teniendo en cuenta el modelo de desarrollo adoptado para el proyecto y los beneficios que aportan estos estilos. Ambos pertenecen al grupo de Estilos de Llamada y Retorno, en el cual se enfatiza la modificabilidad y la escalabilidad. Además son los más generalizados en sistemas en gran escala. Particularmente el estilo MVC fue re-implementado, por el equipo central de arquitectura, de forma tal que las clases controladoras de los diferentes componentes puedan obtener los datos mediante instanciaciones a objetos de clases del modelo del negocio, o tomándolos directamente del modelo del dominio.

1.5. Patrones de diseño

Los patrones de diseño brindan soluciones ya probadas y documentadas a problemas que están sujetos a contextos análogos en el desarrollo de software. Cada patrón permite que algunos aspectos de la estructura del sistema puedan cambiar independientemente de otros aspectos. Además posibilitan la reusabilidad, flexibilidad y el mantenimiento. Dichos patrones se clasifican según el propósito para el que han sido definidos en: Creacionales, que solucionan problemas de creación de objetos, ayudan a encapsular y abstraer dicha creación; Estructurales, que solucionan problemas de composición y/o agregación de clases y objetos y de Comportamiento, que brindan soluciones respecto a la interacción y responsabilidades entre clases y objetos, así como los algoritmos que encapsulan. (9)

En el CEIGE se seleccionaron varios patrones para ser aplicados en el diseño arquitectónico del sistema CEDRUX. Por ejemplo:

- **Composición** (del inglés Composite) es un patrón estructural. Crea objetos dentro de estructuras jerárquicas, y le permite a los clientes tratar uniformemente los objetos simples y compuestos. (10).
- **Fachada** (del inglés Facade) es un patrón estructural que proporciona una interfaz unificada de alto nivel para un subsistema, que facilita su uso. La "fachada" satisface la mayoría de los clientes sin ocultar las funciones de menor nivel a aquellos que necesiten acceder a ella. (10).
- **Mediador** (del inglés Mediator) pertenece al grupo de patrones de comportamiento. Define un objeto que encapsula cómo interactúan un conjunto de objetos. Promueve un bajo acoplamiento al evitar que los objetos se refieran unos a otros explícitamente y permite variar la interacción entre ellos de forma independiente. (10).
- **Solitario** (del inglés Singleton) es un patrón creacional. Garantiza que una clase sólo tenga una instancia y proporciona un punto de acceso global a ella. (10).

1.6. Desarrollo de software basado en componentes

Los componentes software surgen, en cierta medida de la necesidad de desarrollar sistemas mediante el ensamblaje de módulos independientes ya existentes. Analizando las definiciones de qué es un componente, por parte de Szyperski[9], en WSBC (WebSphere Business Components) de IBM[10], y del Instituto de Ingeniería de Software[11] (SEI, del inglés Software Engineering Institute), se puede afirmar que un componente, en esencia, es una unidad reutilizable que puede interoperar con otros módulos software por medio de sus interfaces, las cuales define desde donde se puede tener acceso a los servicios que este ofrece a los demás componentes. Un componente puede presentarse en forma de código fuente o código objeto; puede estar escrito en lenguaje funcional, procedural u orientado a objetos y puede ser tan simple como un botón GUI o tan complejo como un subsistema. (11).

[9] Componente de Szyperski. Definición que propone Clemens Szyperski.
[10] Componente WSBC de IBM. Definición que propone la arquitectura WSBC de IBM (*International Business Machines*).
[11] Componente SEI. Definición que adopta el SEI (Software Engineering Institute).

El desarrollo de software basado en componentes (en lo adelante DSBC) constituye una aproximación del desarrollo de software que describe, construye y emplea técnicas software para elaborar sistemas abiertos y distribuidos, mediante el ensamblaje de partes software reutilizables. (11).

Es utilizado para reducir los costos, tiempo y esfuerzos de desarrollo del software, y de esta manera incrementar el nivel de productividad de los grupos desarrolladores y minimizar los riesgos; a su vez ayuda a optimizar la fiabilidad, flexibilidad y la reutilización de la aplicación final.

La modularidad, la reusabilidad y componibilidad son características muy relevantes de la tecnología de programación basada en componentes, en las cuales coincide con la tecnología orientada a objetos de la que puede considerarse una evolución. No obstante en esta tecnología también se requiere robustez debido a que los componentes deben operar en entornos muchos más heterogéneos.

Otro aspecto a tener en cuenta en el DSBC, es el poder integrar lo mejor de las tecnologías para realizar una aplicación personalizada, a la medida de las necesidades de los clientes; lo cual le permite a los desarrolladores y/o a la empresa adquirir las tecnologías que más se adapten a sus particularidades, sin incurrir en gastos de licenciamiento o soporte y actualización de grandes soluciones.

Este enfoque de desarrollo además de que posibilita alcanzar un alto nivel de reutilización de software, también ofrece otros beneficios que no son menos importantes, como por ejemplo:

- Simplifica las pruebas. Permite que las pruebas sean ejecutadas probando cada uno de los componentes antes de probar el conjunto completo de componentes ensamblados. (12).
- Simplifica el mantenimiento del sistema. Cuando existe un débil acoplamiento entre sus componentes, el desarrollador puede actualizar y/o adicionar componentes según sea requerido, sin afectar otras partes del sistema. (12).
- Mayor calidad. Dado que un componente puede ser construido y luego optimizado continuamente por un experto u organización, la calidad de una aplicación basada en componentes mejorará con el paso del tiempo. (12).

1.7. Diseño arquitectónico de sistemas software

El diseño arquitectónico representa las estructuras de los datos y los componentes que se requieren para construir un sistema de información, así como sus propiedades y las interrelaciones que existen entre estos y otros elementos arquitectónicos del sistema.

Para un buen diseño es recomendable apoyarse en la reutilización de la experiencia ya acumulada y documentada. Antes que nada se debe identificar el tipo de sistema que se quiere construir, lo cual permite examinar la arquitectura de sistemas ya construidos, comprender los requisitos que estos enfrentan y contrastarlos con los que se exigen en el sistema a desarrollar. Hay que tener en cuenta que en cualquier tipo de aplicación que existan necesidades similares, muchos de los componentes que se utilizan en su desarrollo suelen ser los mismos y por tanto estos son altamente reutilizables para el diseño arquitectónico de otros sistemas. También deben seleccionarse adecuadamente los estilos arquitectónicos a utilizar, acorde a los requerimientos identificados en el análisis. Esto guarda una estrecha relación con la metodología de desarrollo de software que se adopte para la construcción de la aplicación. Las metodologías que gestionen de forma directa las cuestiones arquitectónicas y estructurales, podrán producir no sólo productos de mayor calidad, sino a un menor costo y en menos tiempo. (13). Esto se debe a que los riesgos arquitectónicos del proyecto son menores y están mucho más controlados, y que al poder integrar una visión orientada a componentes, las posibilidades de reutilizar software ya desarrollado son mucho mayores, con las ventajas que ello implica. (13). Otro aspecto de gran relevancia en este ámbito lo constituyen los patrones de diseño, cuyo propósito es capturar buenas prácticas que permitan mejorar la calidad del diseño de un sistema, especificando elementos que soporten roles útiles en dicho contexto, encapsulando la complejidad y haciéndolo más flexible.

1.8. Descripción de una arquitectura software.

Todo sistema tiene una arquitectura, pero no todos la tienen descrita y esto es sumamente importante, puesto que la descripción de la arquitectura software constituye la especificación de los diferentes elementos arquitectónicos que conforman un sistema, de las relaciones de integración entre los mismos y de los patrones arquitectónicos utilizados en su diseño y desarrollo.

La recomendación IEEE Std 1471-2000[12] establece una base común para la descripción de arquitecturas de software e implementa para ello tres términos básicos: arquitectura, vista y punto de vista. Un punto de vista define un patrón o plantilla para representar un conjunto de incumbencias relativo a una arquitectura, establece los convenios por las cuales una vista es creada, representada y analizada; mientras que una vista es la representación concreta de un sistema en particular desde una perspectiva unitaria (4); por lo que se regula por un punto de vista determinado. En este sentido una vista describe parte de la arquitectura de un sistema. El punto de vista determina que lenguaje se debe utilizar para describir la vista y permite la formalización de grupos de modelos, estableciendo los métodos y la notación para el desarrollo de los mismos. Una vista también se compone de modelos, aunque posee atributos adicionales. (5). Los modelos proporcionan la descripción específica o contenido de una arquitectura. (5). Cada paradigma de desarrollo exige diferente número y tipo de vistas o modelos para describir una arquitectura; no obstante hay al menos tres vistas absolutamente fundamentales en cualquier arquitectura:

- La visión estática: describe qué componentes tiene la arquitectura.
- La visión funcional: describe qué hace cada componente.
- La visión dinámica: describe cómo se comportan los componentes a lo largo del tiempo y cómo interactúan entre sí. (14)

Como ejemplos de las distintas vistas de arquitectura que existen se encuentran:

- Modelo 4+1 vistas de arquitectura, que ofrece Philippe Kruchten[13], en 1995, en el que propone una Vista lógica, que muestra el subconjunto arquitectónicamente significativo del modelo de diseño; la Vista de procesos, que describe los aspectos de concurrencia y sincronización del diseño; la Vista física o de despliegue, que describe el mapeo del software en el hardware y refleja los aspectos de distribución; Vista de desarrollo o implementación; que describe la organización estática del software en su ambiente de desarrollo (15) y una quinta vista, la de Escenarios o Vista de casos de uso**,** como también se le denomina, que se muestra y traza en cada una de las anteriores y que está formada por las necesidades funcionales que cubre el sistema. (16)

[12] Recomendación IEEE Práctica para la descripción de la Arquitectura de Software.
[13] Philippe Kruchten (1952). Ingeniero de software canadiense, conocido como Director de Desarrollo de Procesos (RUP) en Rational Software y desarrollador del modelo 4+1 Vistas de arquitectura.

- El modelo de Racionalización de Vistas, del SEI (2002), en el cual las vistas se denominan ViewTypes[14], y propone la ViewType Modular, que muestra cómo está el sistema estructurado como conjunto de unidades de implementación; ViewType Componente y Conector, que muestra cómo el sistema es estructurado en un conjunto de elementos que tienen comportamiento e interacción en tiempo de ejecución y ViewType Asignación**,** que describe la relación entre las unidades de software y los elementos del entorno, elementos que no son software. (18)
- El modelo de vistas definido por Microsoft, en el cual se establecen la Vista Conceptual, cercana a la semántica de negocios y a la percepción de los usuarios no técnicos, la Vista Lógica, que define los componentes funcionales y su relación en el interior de un sistema, en base a la cual los arquitectos construyen modelos de aplicación que representan la perspectiva lógica de la arquitectura de una aplicación y la Vista Física, que es la menos abstracta y que ilustra los componentes específicos de una implementación y sus relaciones. (8)
- El modelo de descripción arquitectónica propuesto por el marco de referencia o framework arquitectónico del Open Group (TOGAF[15]) que define cuatro vistas: Arquitectura de Negocios, que define la estrategia del negocio, gobierno, organización y los procesos de negocio principales, Arquitectura de Datos/Información, que describe la estructura lógica y física de los activos de datos y la gestión de sus recursos, Arquitectura de Aplicación, que provee un plano (blueprint en inglés) de las aplicaciones, las interacciones entres estas y sus relaciones con los procesos centrales del negocio y la Arquitectura Tecnológica, que describe las capacidades de software y hardware que se requieren para soportar en negocio, datos y aplicaciones, incluye infraestructura IT (tecnología), redes, middleware[16], comunicaciones, procedimientos y estándares. (17)

En cada proyecto, atendiendo a sus características propias y a las consideraciones de las partes interesadas del mismo, se seleccionan las vistas más adecuadas, que se ajusten a sus exigencias y a lo que requieren mostrar en cuanto a su arquitectura. Para el sistema CEDRUX se seleccionaron las

[14] Viewtypes: tipos vista.
[15] Framework de arquitectura empresarial que provee un enfoque para el diseño, planificación, implementación y gobierno de una arquitectura empresarial de información, que ha sido desarrollado por el Open Group, una organización sin fines de lucro formada por los principales actores de la industria. HP, NEC e IBM, entre otros.
[16] Componentes de software que actúan como intermediarios entre otros componentes de software, generalmente, en el marco de la interacción cliente/servidor.

siguientes vistas, acorde a la corriente arquitectónica procesual del SEI, que es la corriente adoptada por el proyecto, incluyéndole algunos elementos de otras corrientes:

- Vista de Sistema, que queda conformada por tres vistas:
  - Vista de Componentes: Encargada de las definiciones de los tipos de componentes posibles a definir en el proyecto, de la especificación de sus características, así como de la composición estructural interna de cada uno de estos componentes (vista vertical de la arquitectura). (19).
  - Vista de Datos: Encargada de todas las definiciones a nivel de datos, de la integración de los distintos modelos, de los patrones, estándares y definiciones a este nivel. (19).
  - Vista de Integración: Encargada de los procesos de integración interna (entre componentes de un mismo proyecto) y externa (entre proyectos distintos), establece las definiciones, estándares, protocolos de comunicación y reglas de intercambio de información. (19).
- Vista de Tecnología, que queda conformada por dos vistas:
  - Vista de Seguridad: Chequea e implementa todos los aspectos relacionados con el acceso a la aplicación, la modificación, lectura o eliminación de la información, etc. (19).
  - Vista de Presentación: Encargada de cómo luce el software, cuáles son los colores que lleva la aplicación, cómo son los botones, los vínculos y todos los elementos de alta significancia desde el punto de vista de la presentación. (19).
- Vista de Infraestructura: Es la encargada de la definición de la plataforma tecnológica a utilizar en la elaboración del producto, así como de la definición y disponibilidad de los distintos servicios telemáticos necesarios en la confección del producto, así como del diseño de los distintos escenarios de despliegue posibles. (19).

La definición de estas vistas también se fundamenta con los elementos estudiados de los modelos de vistas arquitectónicas formulados por Microsoft y TOGAF. El modelo propuesto por Kruchten (4+1 vistas) no se utilizó debido a que este fija sus perspectivas basándose en casos de uso, como modo representativo de las funcionalidades de un sistema y para CEDRUX, el negocio es analizado y modelado por medio de procesos. Además ni este modelo ni el del SEI, involucra el punto de vista de datos. Perspectiva que si es requerida en CEDRUX y que es tratada en las vistas de TOGAF y Microsoft, lo cual permite definir los tipos y fuentes de datos para dar soporte a los procesos de negocio de una entidad.

Microsoft, al igual que TOGAF, articula como puntos de vista el negocio, la aplicación, la información y la tecnología, lo cual posibilita la especificación de las funcionalidades de un sistema, la descripción de los servicios automatizados que soportan los procesos de negocio, de los componentes y las relaciones entre estos, el manejo y distribución de los datos que emplearía una empresa, entre otros aspectos. TOGAF, en su vista de aplicaciones también define los elementos que son manejados por la arquitectura de datos y que apoyan las funciones del negocio, especificando los elementos requeridos para la gestión y presentación de la información. Vista que a veces se denomina en otras arquitecturas, como vista de sistema. Otro aspecto valorativo en el por qué del uso de ambos modelos es el hecho de representar una perspectiva tecnológica, pues una buena arquitectura tecnológica puede brindar disponibilidad, seguridad y fiabilidad de la información. También puede ocuparse del diseño de la interfaz de usuario, de las comunicaciones. Además de que la misma es desarrollada para soportar las aplicaciones específicas para la organización, mientras que una buena arquitectura de aplicación aprovecha la arquitectura tecnológica para brindar un desempeño consistente mediante los requerimientos operacionales de un sistema, dígase fiabilidad, rendimiento, capacidad de gestión, interoperabilidad, etc. En ello radica el hecho de formular las vistas de seguridad y de presentación que conforman la vista tecnológica para CEDRUX. En si las vistas y perspectivas de Microsoft y TOGAF son modelos muy parecidos. Y los elementos manejados en los mismos son tratados en CEDRUX, de una u otra forma, puesto que este requiere tener de manera detallada la definición y descripción de aspectos como la organización que tiene el sistema, de los componentes y módulos que lo integran, las dependencias y colaboraciones entre estos, el uso y distribución de los datos, la presentación, las herramientas y lenguajes a utilizar, entre otros elementos que contribuyen al desarrollo y mantenimiento del sistema.

En un sistema complejo, las descripciones arquitectónicas pueden ser desarrolladas tanto para componentes del sistema, como para el sistema como un todo. Dichas descripciones contribuyen a las comunicaciones entre las organizaciones involucradas en el desarrollo, producción, presentación, operación y mantenimiento de un sistema, así como a los intercambios entre clientes y desarrolladores como una parte de negociaciones del contrato. También se emplean como criterios para la certificación de la conformidad de las implementaciones de la arquitectura, así como en la documentación de desarrollo y de mantenimiento, incluyendo material para repositorios y materiales de capacitación. Además sirven de

apoyo operativo y de infraestructura, gestión de configuración y reparación, revisión, análisis y evaluación del sistema en todo el ciclo de vida.

1.9. Conclusiones

Con el estudio realizado en este capítulo se han podido ampliar los conocimientos sobre los temas relacionados con el proceso de desarrollo y el diseño de la arquitectura de un sistema. También quedaron reflejadas buenas prácticas para realizar un buen diseño arquitectónico, así como la importancia del mismo. Además se logra obtener una adecuada preparación y se sentaron las bases para darle solución satisfactoriamente al problema planteado, teniendo en cuenta los estilos arquitectónicos y patrones de diseños adoptados para el proyecto.

Capítulo II Vista de arquitectura de sistema

2.1. Introducción

En este capítulo se realiza un análisis de artefactos generados en el negocio, se describe la estructura de empaquetamiento de los subsistemas, identificando de este modo los componentes involucrados, lo cual constituye el punto de partida para llevar cabo la vista de arquitectura de sistema. Por último, se especificarán los artefactos pertenecientes a las vistas de componentes y de integración, las cuales forman parte de la vista de sistema.

2.2. Análisis de artefactos generados en el negocio

Durante el modelamiento del negocio, para cada uno de los subsistemas se desarrollaron una serie de artefactos que permiten clarificar los procesos involucrados e identificar las áreas de fortalezas y debilidades, lo cual ayuda a reducir la duración y defectos de ciclos; también permiten profundizar y priorizar información, reconocer y esclarecer los conceptos y términos más importantes afines a dichos procesos y como se relacionan entre si; además muestran una descripción detallada y enfocada a las características y funcionalidades del sistema. Por tanto, estos artefactos constituyen la base del diseño y el apoyo de la implementación del sistema CEDRUX. En este documento se analizarán específicamente tres de ellos: el Mapa de procesos, los Requisitos de software y el Modelo conceptual.

Modelo conceptual. El modelo o mapa conceptual muestra los conceptos más importantes utilizados en las áreas de procesos que representan cada uno de los subsistemas, la organización jerárquica que presentan y las relaciones existentes entre ellos, visualizando así la información que se maneja en un subsistema, en su conjunto.

Mapa de procesos de negocio. El mapa de procesos visualiza como se relacionan los procesos de negocio involucrados en los subsistemas, especificando las entradas y salidas que tiene cada proceso.

Requisitos de software. El propósito fundamental del levantamiento de requisitos es guiar el desarrollo hacia el sistema correcto, lo cual se logra mediante una descripción de los requisitos del sistema lo suficientemente buena como para poder establecer un acuerdo entre los clientes y los desarrolladores sobre qué debe y qué no debe hacer el sistema.

2.2.1. Estructura y Composición

El subsistema Estructura y Composición permite definir el entorno estructural organizativo en el cual se van a ubicar las entidades (que es la estructura superior o externa) y la estructura interna dentro de cada una de las mismas; además de definir los cargos y puestos de trabajos por los cuales estará compuesta cada área de una unidad. También se puede especificar el nivel jerárquico que existe entre las entidades que se estén gestionando, es decir, a quien se subordina las mismas.

Dichas estructuras, los niveles jerárquicos, así como los cargos civiles y/o militares que pueden contener las entidades, están relacionados en el modelo conceptual, conjuntamente a otros términos del dominio no menos importantes.

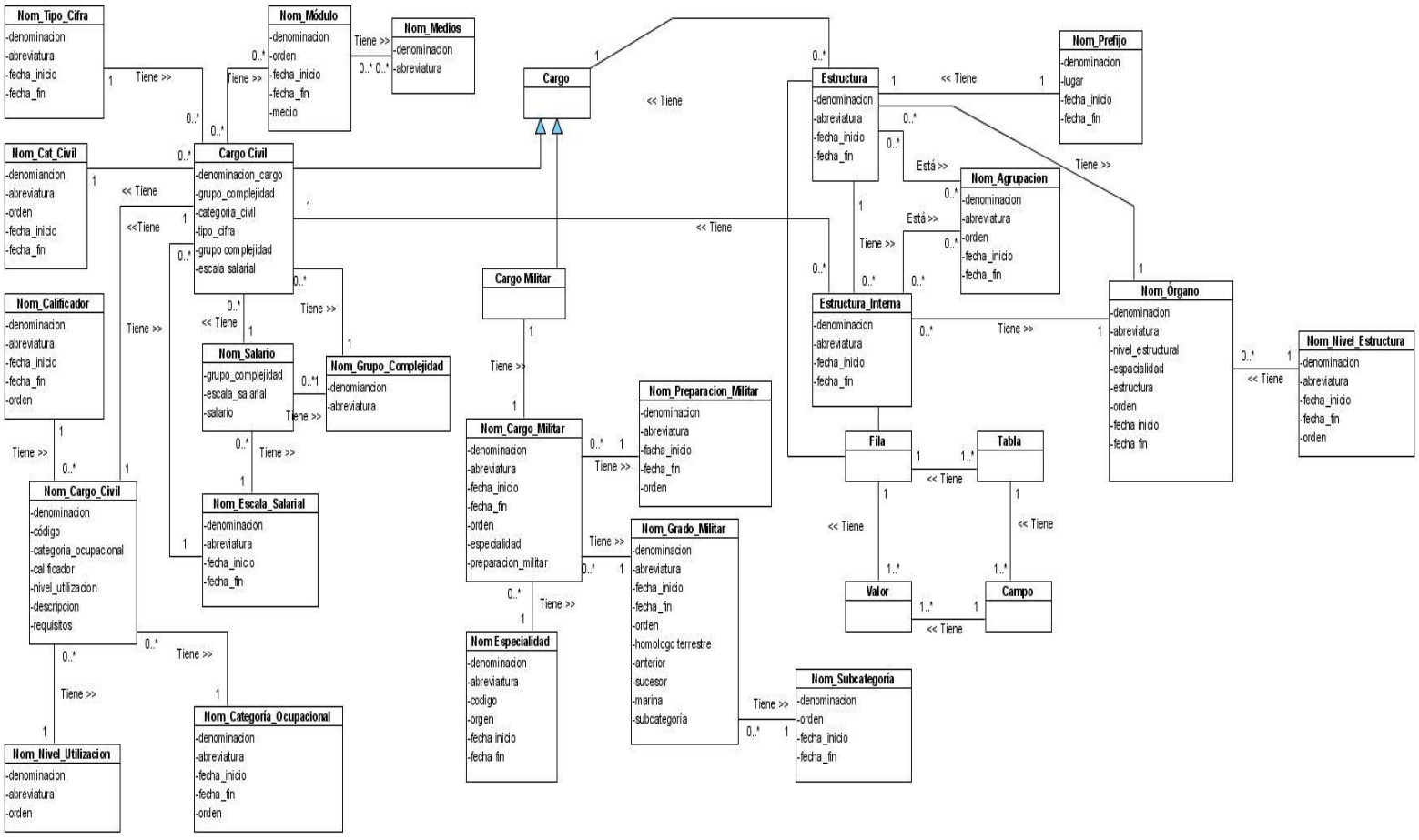

Figura 1 Modelo conceptual del subsistema Estructura y Composición

Cada estructura tiene un prefijo que identifica el lugar donde se implante el software, este prefijo representa la especificación de utilidad de la entidad en que se instale el sistema. Ambas estructuras, se definen en una agrupación determinada y tienen órganos específicos, por ejemplo Subdirección, Grupo empresarial, etc., que presentan un nivel estructural determinado. Los cargos civiles están asociados a grupos de complejidad del trabajo, los cuales influyen en la asignación de los salarios correspondientes a cada cargo, teniendo en cuenta además, las escalas salariales que se apliquen. Los cargos militares pueden tener un grado militar determinado, al cual se encuentra asociada la subcategoría salarial que le será aplicada a los oficiales con dichos cargos; a estos también se les asigna la correspondiente preparación militar, que deben recibir estos oficiales. En el modelo también se muestran los módulos, que constituyen conjuntos de medios que pueden ser asignados a un cargo o área de una entidad.

Este subsistema consta de seis procesos, definidos en el modelamiento del negocio, que responden a la gestión (se refiere a crear, modificar y eliminar) de las estructuras, tanto externas como internas, correspondientes a una o varias entidades:

- ✓ Crear estructura: proceso en el cual se crean las estructuras requeridas (Nivel 1, Agrupaciones, Entidades y Unidades). Para ello se necesita un documento que contiene las especificaciones de la estructura a crear, los datos y requisitos que debe cumplir. Luego de creada se conforma un documento con las características y funcionalidades de la misma.
- ✓ Modificar estructura: proceso en que se modifica una estructura ya efectiva, para lo cual se emplea el documento existente de la situación de la estructura. Luego de modificada se conforma un documento con todos los cambios realizados.
- ✓ Eliminar estructura: proceso que permite eliminar estructuras ya efectivas. Para realizar el proceso se necesita un documento legal que explique dicha eliminación, luego se conforma un documento con la información sobre las razones de eliminación de la estructura.
- ✓ Crear estructura interna: proceso que permite crear estructuras internas. Para ello se necesita un documento que contiene las especificaciones de la estructura a crear, los datos y requisitos que debe cumplir. Luego de creada se conforma un documento con las características y funcionalidades de la misma.

- ✓ Modificar estructura interna: proceso en que se modifica una estructura interna ya efectiva, para lo cual se emplea el documento existente de la situación de la estructura. Luego de modificada se conforma un documento con todos los cambios realizados.
- ✓ Eliminar estructura interna: proceso que permite eliminar estructuras internas ya efectivas. Para realizar el proceso se necesita un documento legal que explique dicha eliminación, luego se conforma un documento con la información sobre las razones de eliminación de la estructura.

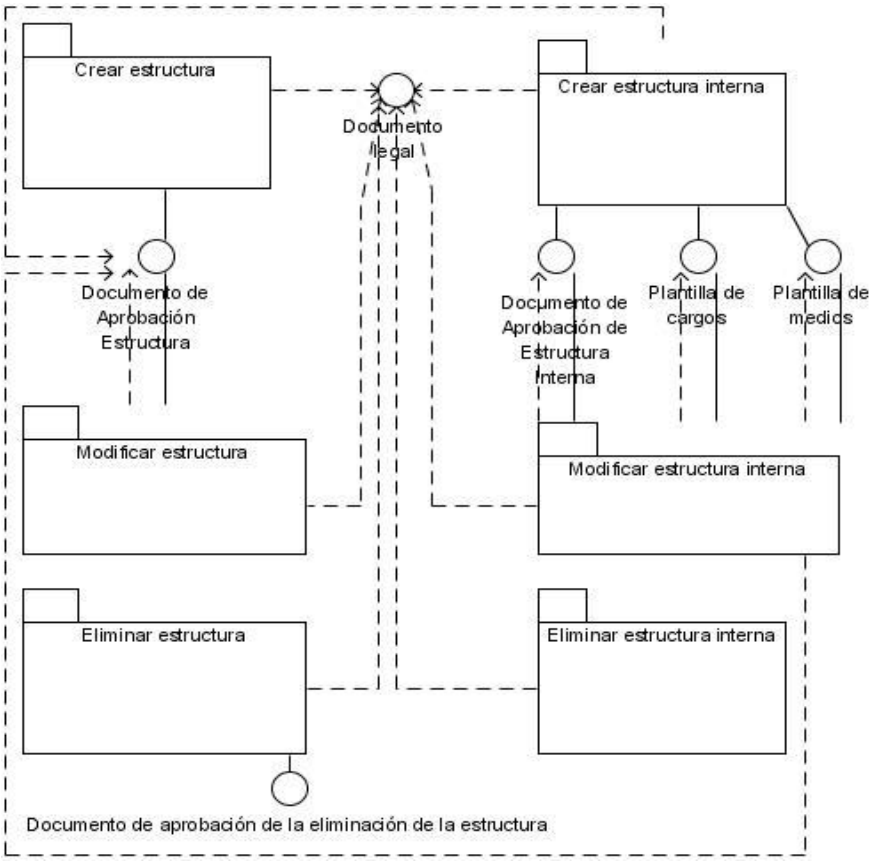

Figura 2 Mapa de procesos del subsistema Estructura y Composición

Las entradas son los documentos necesarios para llevar a cabo los procesos y las salidas constituyen la constancia de lo que se realizó y las condiciones en que queda el proceso luego de ser ejecutado. La estructura externa se corresponde a Ministerios, Institutos, Grupos, Uniones, Entidades y Unidades, mientras que la interna son las Áreas y Subáreas. Esto se cumple en el caso de Cuba, pero pueden definirse otras formas de organización que sean apropiadas al lugar donde se implante el sistema.

Para efectuar la implementación de todos estos procesos se delimitaron una serie de requerimientos que establecen que debe hacer exactamente el subsistema funcionalmente. Estos están agrupados en cuatros paquetes: Definir Estructuras, Gestionar Nomencladores, Gestionar Estructuras y Gestionar Reportes, que fueron definidos con el propósito de lograr una mejor organización y comprensión de dichos requisitos[17] en el diseño de esta solución (Figura 2). Los mismos especifican cómo proceden la gestión de las estructuras, la gestión de la información que persistirá en los nomencladores, de los niveles estructurales y la generación de una serie de reportes que pueden ser de interés para el usuario, por ejemplo la Plantilla de cargos de una o varias unidades, la Relación de registros de las entidades por agrupación, la Relación de la localización de las entidades por nivel estructural, entre otros.

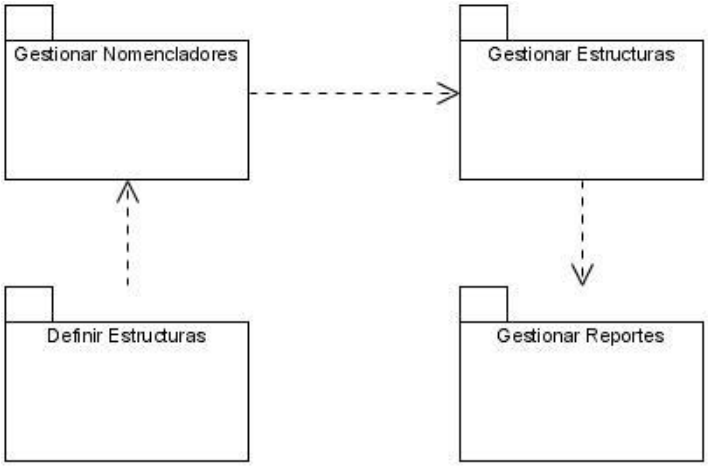

Figura 3 Diagrama de paquetes del subsistema Estructura y Composición

2.2.2. Configuración

El subsistema de Configuración permite gestionar toda la configuración que se establece en una entidad; brinda facilidades de configuración y adaptación que guiarán a las entidades en el manejo de sus procesos contables, financieros, logísticos, entre otros procesos y funciones administrativas, permitiendo de este modo, adaptar el sistema a sus peculiaridades y entorno de trabajo.

Los términos y atributos tratados en el dominio de Configuración se relacionan en el mapa conceptual definido durante el análisis de esta solución (Figura 4).

[17] El término requisitos se refiere también al término requerimientos.

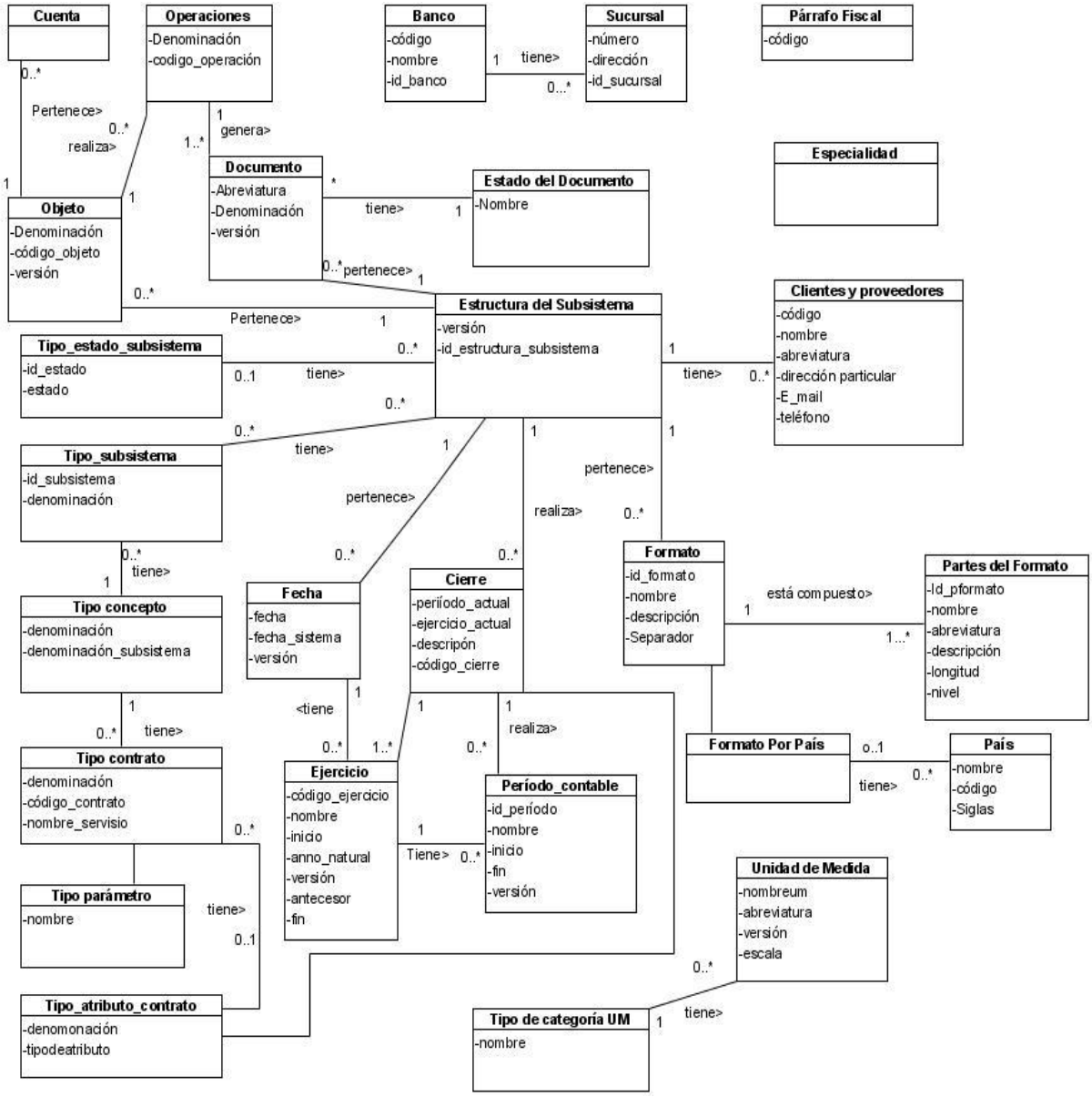

Figura 4 Modelo conceptual del subsistema Configuración

Cada subsistema dentro de una entidad determinada, presenta un estado (Activado, En explotación, Detenido, etc.). Este tiene asociados conceptos por los cuales se realizan determinados contratos que poseen atributos específicos y reciben parámetros en los servicios que se emplean en la realización de los mismos. Todos los subsistemas disponen de varios objetos, con los cuales se realizan diferentes operaciones; cada operación genera un documento especificando los detalles de la misma y se afectan

una o más cuentas pertenecientes a dichos objetos, por ello es posible llevar el control de las cuentas que son afectadas por los objetos que contiene una entidad. Los subsistemas, además, pueden presentar fechas reales y/o fechas contables[18], debido a que si ocurre un atraso en la ejecución de cualquier proceso o procedimiento, las operaciones vinculadas a los mismos pueden realizarse en fechas anteriores a la real. Estos realizan sus cierres o cambios de períodos contables, permitiendo cambiar, modificar los ejercicios y períodos que estos presentan, con el propósito de que siempre se encuentren en el ejercicio y período actual, por lo que cada cierre va asociado al ejercicio contable, que constituye el período comprendido entre dos balances anuales sucesivos, y cada ejercicio contiene uno o mas períodos contables, que son los espacios de tiempo en que deben rendirse y registrarse los resultados de una entidad. También queda señalada en este diagrama la relación de los bancos y las sucursales que se encuentran vinculados con las entidades, así como los clientes y proveedores que trabajan con las mismas.

Para este subsistema no se modelaron los procesos de negocio; el análisis de la solución se realizó partiendo de la definición de los conceptos que se manejan en la misma y del levantamiento y la validación de los requisitos del sistema, por tal razón no se muestra un mapa de procesos. Estos requerimientos están sujetos a la gestión de la mayoría de los componentes configurables que integran el sistema, a la definición de la configuración inicial de las entidades, a la gestión de los ejercicios y períodos contables, de los objetos, párrafos fiscales, a la gestión de los diferentes documentos que se emplean en la mayoría de las operaciones, al manejo de las cuentas, entre otras funciones que son substanciales para el trabajo del resto de los subsistemas.

2.2.3. Multimoneda.

El subsistema Multimoneda se ocupa de las operaciones que se realizan referentes a temas monetarios y al uso de la dualidad monetaria. Permite que las entidades definan y empleen diversas monedas, según lo requieran, en la ejecución de transacciones u operaciones económicas y su registro. Igualmente posibilita la gestión de las tasas de cambios y la reevaluación de cuentas, entre otras funcionalidades, que son esenciales para llevar el control de la gestión económica y del estado financiero en cualquier entidad.

[18] Real: fecha real del subsistema (Ej. 05 enero 2010). Fecha contable: es cuando se hacen operaciones en fechas anteriores a la real.

Cada entidad define las monedas que utilizará en sus operaciones, seleccionándolas de un nomenclador que contiene todas las monedas reconocidas nacional e internacionalmente, y especifica los tipos de billetes y monedas en que se puede desglosar físicamente una moneda. Las tasas de conversión o cambio, son contratadas por los clientes o proveedores, estas rigen la reevaluación de cuentas y se aplican a la moneda, en relación con la moneda contable y/o alternativa que se defina, de las monedas ya establecidas, para llevar a cabo el registro contable de las operaciones, sobre la cual se emite el valor de la mayoría de las transacciones que se ejecuten en la entidad. Todos estos conceptos, conjuntamente a sus atributos, se encuentran reflejados en el siguiente modelo:

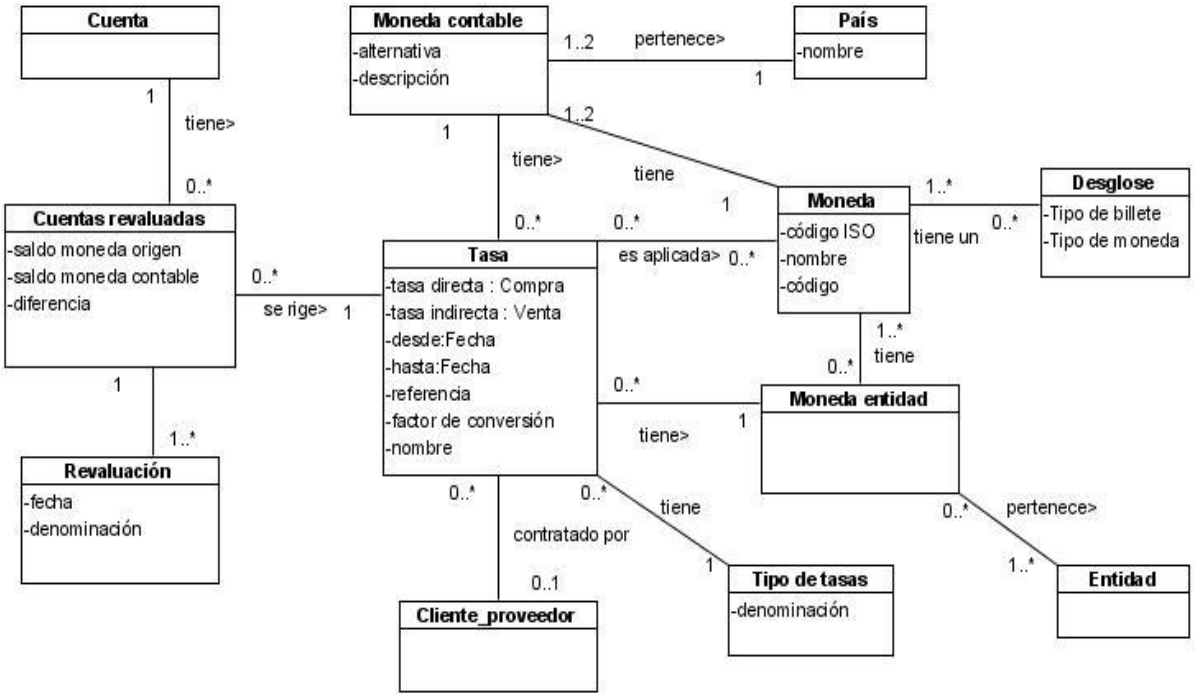

Figura 5 Modelo conceptual del subsistema Multimoneda

En este caso ocurre lo mismo que el caso de Configuración, el hecho de que tampoco se modelaron los procesos de negocio, por lo cual tampoco se muestra un mapa de procesos; el análisis de esta solución también se llevó a cabo a partir del levantamiento de requisitos, en los cuales se tratan la mayoría de los términos y conceptos anteriormente descritos. Concretamente, las funcionalidades o requerimientos generales de Multimoneda suplen la gestión de todas las monedas que existen en el mundo, la gestión de las monedas que emplearán las entidades, la gestión del desglose de monedas que estas requieran, la

gestión de la moneda contable y/o alternativa, de las tasas de cambio a aplicar sobre las monedas y el cálculo y reversión de cuentas pertenecientes a estas entidades.

2.3. Breve descripción de la estructura de empaquetamiento de un componente

En estos subsistemas cada uno de sus componentes, está empaquetado independiente del resto de los componentes del sistema, acorde a las decisiones de arquitectura de sistema tomadas por el proyecto (Figura 6). Cada componente tiene implementado un modelo, un controlador y una vista (models, controllers y views en inglés, en ese mismo orden), compuesto por un conjunto de clases y paquetes, encargados de gestionar el acceso a datos, los eventos del sistema, atendiendo así las solicitudes que provienen de la vista y del modelo y de definir las interfaces de usuario para modelar visualmente la funcionalidad del componente, respectivamente. Dentro del paquete Modelo, se localizan los paquetes Negocios (business en inglés), que comprende las clases del negocio y Dominio (domain en inglés), que presenta las clases entidad, en las cuales se gestiona la persistencia de los datos en la base de datos. También contiene una interfaz para establecer la comunicación con los componentes conque se interrelacione y ofrecerles una serie de funcionalidades que se agrupan como servicios dentro de un mismo paquete, además de un paquete denominado Validaciones (validators en inglés), que engloba las reglas del negocio.

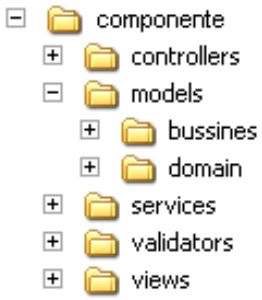

Figura 6 Estructura de un componente.

2.4. Vista de Arquitectura de sistema para las soluciones Estructura y Composición, Configuración y Multimoneda

Al diseñar la Arquitectura de sistema se define la organización principal de un sistema basado en los sistemas, subsistemas, módulos, componentes y las colaboraciones entre estos, las restricciones y las

configuraciones a asumir en función de los elementos y términos del negocio que los mismos abstraen. Dentro de esta arquitectura deben quedar bien definidos los patrones de diseño y los estilos arquitectónicos que se deben aplicar en el diseño del sistema, la estructura de empaquetamiento de los componentes abstraídos, se debe diseñar la integración entre los diferentes elementos arquitectónicos identificados en esta área y con otros elementos vinculados a la Arquitectura tecnológica. Es por ello que se considera una de las disciplinas más complejas de la Arquitectura de software.

Para realizar una vista arquitectónica de sistema, el equipo de arquitectura propone el desarrollo de un documento que constituye la Línea base de la arquitectura de cada subsistema. Este documento, como lo indica su nombre, delimita las responsabilidades arquitectónicas de un subsistema; permite establecer la comunicación y dependencias entre los componentes y módulos que lo integran, así como precisar los objetivos, la criticidad y complejidad de los componentes con vistas a su reutilización y prioridad en la implementación. A este se vinculan cuatro artefactos que describen gráfica y/o matricialmente todos estos aspectos: el Mapa de componentes, la Especificación de componentes, la Matriz de complejidad y criticidad y la Matriz de integración. Dichos artefactos se encuentran organizados dentro del expediente arquitectónico de sistema, definido también por el equipo de arquitectura. El expediente para los subsistemas se denomina Línea base de subsistema y guarda por cada uno de los subsistemas su línea base de arquitectura, conformada por los artefactos de sistema que se mencionan con anterioridad.

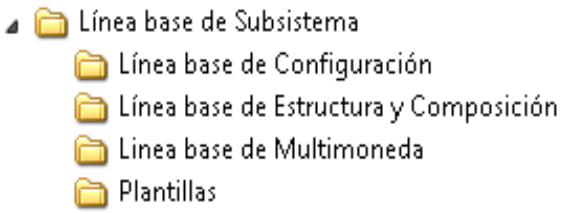

Figura 7 Expediente de arquitectura de sistema para los subsistemas que integra CEDRUX.

2.4.1. Mapa de componentes

Este contiene todos los componentes definidos en un subsistema, así como las dependencias que existen entre los mismos. De este modo permite obtener una vista general del subsistema, partiendo de las dependencias e integraciones de los componentes, y permite evaluar visualmente por cada componente su complejidad.

Los subsistemas Multimoneda y Estructura y Composición presentan, cada uno, un único componente que recibe el mismo nombre del subsistema al que pertenece (componente Multimoneda y componente Estructura y Composición). Por tanto estos componentes no tienen relaciones de dependencia dentro de dichos subsistemas y por ende no fue necesario realizar un mapa de componentes para los mismos. Por su parte, Configuración contiene catorce componentes, que se relacionan a continuación. (Figura 8). Como puede observarse en el modelo, las relaciones entre los componentes se muestran mediante las entradas y salidas que tienen cada uno de ellos, las cuales representan los servicios que estos ofrecen y consumen, por medio de las interfaces que los mismos tienen definidas.

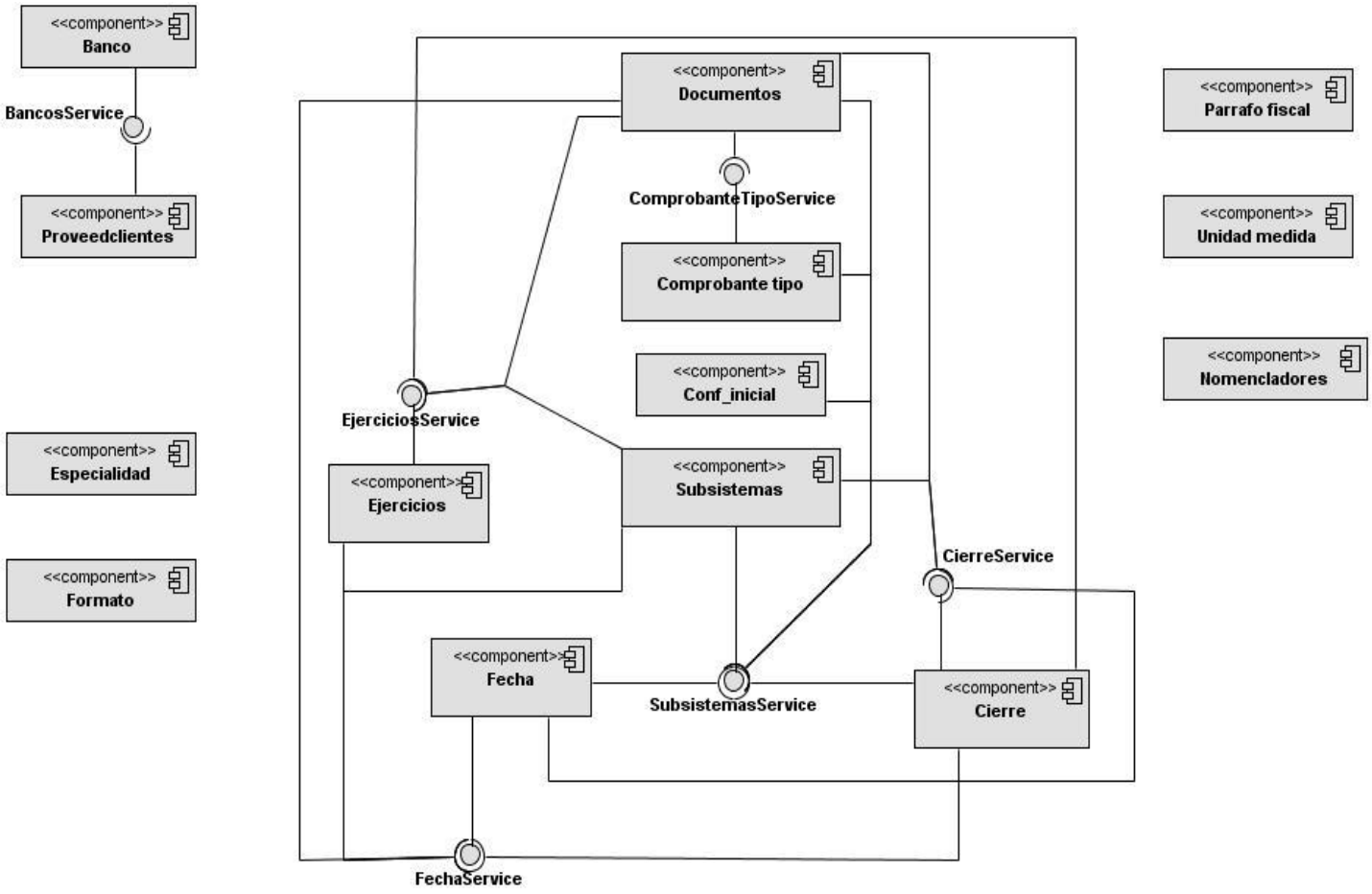

Figura 8 Diagrama de componentes del subsistema Configuración

Por ejemplo, el componente Proveedores y clientes utiliza servicios provenientes de Banco; Ejercicios utiliza servicios que brinda Fecha, a su vez este último emplea servicios del propio componente Ejercicios, de Cierre y de Subsistemas. Por su parte Comprobante tipo sólo emplea los servicios de Subsistemas. Sin embargo, los componentes Nomencladores, Especialidad, Párrafo fiscal, Formato y Unidad medida, no consumen ningún servicio del resto de los componentes.

Realizando una valoración visual, de cómo se encuentran integrados los mismos, se puede determinar que el componente Subsistemas presenta un número mayor de dependencias comparado con el resto de los componentes. Como puede observarse, sus servicios los utilizan otros cinco componentes, para ejecutar algunas de sus funciones. Mientras que del componente Banco, sólo depende un único componente, por ello su dependencia es menor. Lo mismo ocurre para el caso del componente Comprobante tipo. Por consiguiente, puede afirmarse que el componente Subsistemas, resulta ser el más crítico desde el punto de vista de la integración. Al cual le seguirían Fecha, Ejercicios, Cierre, Banco y Proveedclientes.

2.4.2. Análisis de complejidad y criticidad

La matriz de complejidad y criticidad contiene todos los componentes definidos en el subsistema, por cada componente los requisitos funcionales que implementa, la complejidad de dichos requisitos, la cantidad de servicios que brinda y la categoría o tipo de componente. También muestra la dependencia entre componentes, a partir de la cual, conjuntamente con la prioridad de requisitos se determinan la complejidad de los componentes y la criticidad de los mismos. Este análisis se desarrolla con el fin de evaluar la complejidad de cada componente y la relevancia que estos presentan desde el punto de vista de la integración. Ambos criterios son muy importantes en el momento de precisar una prioridad de implementación y a la hora de aceptar o no un cambio determinado.

El valor de complejidad de un componente se calcula mediante la cantidad de requisitos que este engloba más cuatro veces los valores de complejidad de los mismos, más la cantidad de servicios que ofrece el componente, duplicada; mientras que el valor de criticidad de integración depende de la cantidad de componentes que utilicen sus servicios, que dependen de él, sumado a los valores de complejidad que estos presentan. Las fórmulas quedarían de la siguiente forma:

complejidad = cantidad de requisitos + (complejidad de los requisitos*4) + (cantidad de servicios*2)

criticidad = dependencia + ∑complejidad de los componentes dependientes

Con este análisis se obtiene como resultado que el componente de Estructura y Composición, tiene un valor de complejidad de doscientos cincuenta y siete; el mismo no presenta dependencias por ser el único componente de ese módulo, por tanto su criticidad es cero. Lo mismo ocurre con el componente de Multimoneda, respecto a las dependencias, mientras que su valor de complejidad es ciento ocho.

Dentro del módulo de Configuración, el componente de menor complejidad es Nomencladores, con un valor de dieciocho, mientras que el más complejo es Comprobante tipo, con valor de ciento veinte y uno. Solamente seis de sus componentes presentan dependencias, siendo el componente Subsistemas, el que tiene mayor número de componentes dependientes. Por consiguiente, haciendo una comparación entre todos los componentes, sujeta a sus valores de criticidad, también resulta ser Subsistemas, el componente de mayor criticidad de integración. El segundo más crítico es el componente Ejercicios, luego siguen Fecha, Cierre, Banco y por último, Comprobante tipo, en ese mismo orden. El resto presentan criticidad cero, debido a que ningún componente consume sus servicios. Este resultado corrobora la visión obtenida mediante el mapa de componentes correspondiente a este módulo, donde ya se hubo determinado que el componente Subsistemas era el de mayor dependencia, y a su vez define concretamente cuan críticos se encuentran el resto de los componentes que conforman el módulo, desde el punto de vista de la integración.

2.4.3. Especificación de los componentes

En este artefacto se describen todos los componentes definidos en un subsistema, basándose en el objetivo y las características de los mismos. Relaciona y detalla los servicios que ofrece cada componente, precisando los atributos de entrada y de salida de dichos servicios. Dicha especificación se lleva a cabo con el propósito de tener registrada la formalización de los componentes y la relación de los servicios que proveen.

El componente Estructura y Composición se encarga de gestionar las estructuras externas (dígase Nivel 1, entidades, agrupaciones, unidades), las estructuras internas (áreas) y los cargos que componen dichas

áreas dentro de cada unidad establecida. El mismo ofrece cuarenta servicios que son importantes para realizar algunas funcionalidades en otros subsistemas de la aplicación como Capital Humano, Finanzas, Logística, Contabilidad, Configuración, entre otros.

Los componentes de Configuración ofrecen en total ciento ochenta y un servicios, que son utilizados entre estos mismos componentes y por otros subsistemas.

- Banco: Permite gestionar los bancos y las sucursales asociadas a estos, así como especificar los datos de los mismos.
- Cierre: Se encarga de gestionar el cierre de los subsistemas por estructura, permite cambiar el ejercicio y período contables a cada subsistema, de modo que se encuentre siempre en el ejercicio y período actual.
- Comprobante tipo: Se ocupa de configurar cómo se contabilizan operaciones y generar comprobante de operaciones.
- Conf_inicial: Se encarga de la configuración de la entidad a establecer.
- Documentos: Permite gestionar y cambiar el estado de los documentos manejados por el sistema.
- Ejercicios: Permite definir los ejercicios contables y los períodos que estos comprenden. Dicho ejercicio es el año económico entre dos balances anuales sucesivos, y el período contable no es más que el espacio de tiempo en el que deben rendirse y registrarse todos los resultados de una entidad.
- Especialidad: Se encarga de gestionar las especialidades militares en una entidad.
- Fecha: Componente que permite a los subsistemas cambiar la fecha del sistema.
- Formato: Permite especificar los formatos que tendrán los nomencladores. La entidad Formato representa la forma que se le dará a un formato fijado, y la entidad Parteformato constituye los fragmentos que conforman el código formato.
- Nomencladores: Permite gestionar los nomencladores que tendrá la aplicación, que es donde persiste la información perenne del sistema.
- Párrafo fiscal: Se ocupa de gestionar los párrafos fiscales.
- Proveedores y clientes: Componente que posibilita gestionar (adicionar, modificar, consultar,) los clientes y proveedores y precisar los datos de los mismos (nombre, código, etc.).

- Subsistemas: Se encarga de gestionar el estado (activo, en explotación, con problemas, detenido y desactivado) de los subsistemas por entidades y de definir el identificador del subsistema, por el cual se van a comunicar con él el resto de los subsistemas.
- Unidad medida: Permite gestionar las unidades de medidas que se aplicarán a los productos.

Por su parte, el componente Multimoneda, permite definir las monedas conque se trabajará en la entidad, especificar la moneda contable y/o la alternativa, además de las tasas de cambio que se aplican a cada moneda y las reevaluaciones de cuentas. Este componente contiene alrededor de treinta servicios, que se encuentran detallados en el documento de especificación correspondiente y que también son utilizados por otros subsistemas.

2.4.4. Integración de componentes. Matriz de integración

La integración puede definirse como el proceso de establecer las relaciones de colaboración y comunicación entre los distintos componentes que se encuentran definidos en un sistema. La misma constituye uno de los procesos más difíciles dentro de un proyecto software, ya sea por la heterogeneidad de plataformas y lenguajes de programación, la diversidad y complejidad de cada sistema individual, la dificultad de comprensión de los requisitos para la solución integrada resultante, entre otros motivos. Pero no por ello deja de ser un proceso muy importante y necesario, puesto que permite establecer la colaboración entre todos los componentes del sistema, o entre este y otros sistemas diferentes, garantizándole al mismo una mayor funcionalidad, eficiencia e interoperabilidad.

En el sistema CEDRUX, la integración entre los componentes se establece mediante los servicios que estos ofrecen y consumen entre ellos, utilizando el patrón Inversión de Control (del inglés Inversion of Control, IoC) como medio de especificación de estos servicios; cuyo uso permite lograr un menor acoplamiento entre los componentes de la aplicación y fomentar así la reutilización de los mismos. En el fichero del IoC, los servicios se definen a través de una sentencia en lenguaje XML[19], puntualizando en la misma el nombre del servicio, la clase específica, dentro del código fuente, en que este se encuentra

[19] Lenguaje de marcado extensible (Extensible markup language, en inglés). Es un lenguaje utilizado para estructurar la información en cualquier documento que contenga texto (por ejemplo: archivos de configuración de un programa en particular o una base de datos).

implementado, los parámetros que recibe, en caso de que los requiera en su ejecución, y el valor o valores que retorna como resultado.

Para describir cómo queda la interacción o las relaciones entre los distintos componentes de un subsistema, se emplea la Matriz de Integración, que se encuentra definida dentro de la vista de integración, establecida por el proyecto. Este artefacto contiene todos los componentes definidos en el subsistema, de forma matricial, y en las intercepciones se especifican los servicios que consume el componente en la vertical del componente en la horizontal. Como se menciona anteriormente, dicha matriz se realiza con el propósito de tener registradas las relaciones existentes entre los componentes por medio de sus servicios y conocer mediante cuáles servicios específicamente se establecen dichas relaciones.

En el subsistema Configuración, como se muestra en el mapa de componentes expuesto anteriormente, el componente Cierre se integra con los componentes Ejercicios, Fecha y Subsistemas. A continuación se listan los servicios por los cuales se relacionan.

De Ejercicios consume los servicios:

- PrimerPeriodo: devuelve el primer período contable perteneciente al ejercicio especificado por su ID.
- PrimerPeriodoObjeto: devuelve el primer período objeto perteneciente al ejercicio especificado por su ID.
- ObtenerPeriodo: devuelve los datos del período especificado por su ID.
- EjercicioSiguiente: devuelve el ejercicio posterior al ejercicio especificado por su ID.
- PeriodoSiguiente: devuelve el período posterior al período especificado por su ID.
- PeriodoAnterior: devuelve el período que antecede al período especificado por su ID.
- UltimoPeriodo: devuelve el último período contable del ejercicio referido por su ID.

De Fecha consume el servicio ActualizarFechaSubsistema, que permite cambiar la fecha que tiene un subsistema determinado, en una entidad determinada, ambos referidos por sus identificadores.

De Subsistemas consume los servicios:

- BuscarSubsistemaPorId: devuelve los datos del subsistema especificado por su ID.
- BuscarInstancia: devuelve los datos de la estructura del subsistema especificado por su ID y de la entidad especificada por su ID.
- BuscarSubsistemasEstructura: busca los subsistemas por estructura, en un entidad especificada por su ID.

2. El componente Comprobante tipo es dependiente de Subsistemas, del cual consume los siguientes servicios:

- BuscarSubsistemaPorId: devuelve los datos del subsistema especificado por su ID.
- BuscarSubsistemaEntidad: devuelve los datos de la relación de un subsistema y una entidad determinados, especificada por el identificador de dicha relación.
- BuscarSubsistemasPorEntidad: devuelve todos los subsistemas que pertenecen a la entidad especificada por su ID.
- EncuentraIdestructurasubsistByIdestructura: busca el subsistema especificado por su ID, que ya se encuentra asociado a la entidad también especificada por su ID.

3. El componente Documentos es dependiente de los componentes Cierre, Comprobante tipo, Ejercicios, Fecha y Subsistemas. De Cierre consume los servicios:

- PeriodoActualSubsistema: devuelve el período actual en el que se encuentra el subsistema especificado por su ID, perteneciente a la entidad especificada por su ID.
- EjercicioActualSubsistema: devuelve el ejercicio actual, en el que se encuentra el subsistema especificado por su ID, perteneciente a la entidad especificada por su ID.
- BuscarCierre: devuelve 1 ó 0, si existe o no respectivamente, un cambio de período en el subsistema especificado por su ID.
- AdicionarCierre: permite adicionar los cierres de subsistemas a la base de datos.
- EliminarCierre: permite eliminar los cierres de subsistemas.

De Comprobante tipo consume el servicio ObtenerNomdocs, que devuelve todos los documentos nomenclados en la base de datos.

De Ejercicios consume los servicios:

- PrimerEjercicio: devuelve el primer ejercicio de la entidad especificada por su ID.
- PrimerPeriodo: devuelve el primer período perteneciente al ejercicio especificado por su ID.

De Fecha consume el servicio FechaCinicial, que permite establecer las fechas iniciales.

De Subsistemas consume los servicios EscribirBienvenidaCinicial y CierreCinicial.

4. El componente Ejercicios hace dependencia de Fecha, consumiendo los servicios:

- ProximaFecha: devuelve la fecha que está próxima a la fecha indicada, en correspondencia con la cantidad de días especificados.
- RestaFechas: devuelve la cantidad de días que restan de la fecha fin con respecto a la fecha de inicio, indicadas como parámetros.
- FechaCinicial: servicio para establecer las fechas iniciales.

5. El componente Fecha depende de Cierre, Ejercicios y Subsistemas. De Cierre consume el servicio PeriodoActualSubsistema, que devuelve el período actual en el que se encuentra el subsistema especificado por su ID, perteneciente a la entidad especificada por su ID. De Ejercicios consume los servicios:

- CierreActualSubsistema: devuelve el ejercicio y el período actual en que se encuentra el subsistema especificado por su ID, en la entidad especificada por su ID, además de la descripción del cierre.
- CerrarPeriodo: permite cerrar un período ya establecido, del subsistema especificado por su ID, perteneciente a la entidad especificada por su ID.
- AperturarPeriodo: permite abrir un nuevo período para subsistema especificado por su ID, perteneciente a la entidad especificada por su ID.

De Subsistemas consume los servicios:

- BuscarInstancia: devuelve el subsistema y el estado de este, indicado por su ID, perteneciente a la entidad también indicada por su ID.
- BuscarEstSubSistPorId: busca el subsistema que está asociado a una estructura determinada, especificado por el identificador de dicha asociación.
- BuscarSubsistemasPorEntidad: devuelve todos los subsistemas que pertenecen a la entidad especificada por su ID.
- CierreCinicial:

7. El componente Proveedclientes consume los siguientes servicios de Banco:

- ObtenerBancos: devuelve todos los bancos que están en la base de datos.
- ObtenerSucursales: devuelve todas las sucursales que pertenecen al banco especificado por su ID.
- ObtenerBancoPorId: devuelve los datos del banco especificado por su ID.
- ObtenerSucursalPorId: devuelve los datos de la sucursal especificada por su ID.

8. Por último, el componente Subsistemas hace dependencia de los componentes Cierre, Ejercicios y Fecha.

De Fecha consume el servicio FechaCinicial, que permite establecer las fechas iniciales.

De Cierre consume los servicios:

- PeriodoActualSubsistema: devuelve el período actual en el que se encuentra el subsistema especificado por su ID, perteneciente a la entidad especificada por su ID.
- EjercicioActualSubsistema: devuelve el ejercicio actual, en el que se encuentra el subsistema especificado por su ID, perteneciente a la entidad especificada por su ID.
- BuscarCierre: devuelve 1 ó 0, si existe o no respectivamente, un cambio de período en el subsistema especificado por su ID.
- AdicionarCierre: permite adicionar los cierres de subsistemas a la base de datos.
- EliminarCierre: permite eliminar los cierres de subsistemas.

De Ejercicios consume los servicios:

- PrimerEjercicio: devuelve el primer ejercicio de la entidad especificada por su ID.
- PrimerPeriodo: devuelve el primer período perteneciente al ejercicio especificado por su ID.
- PeriodoSiguiente: devuelve los datos del período posterior al período especificado por su ID.

Para los subsistemas Estructura y Composición y Multimoneda, no fue necesario realizar una matriz de integración debido a que presentan un único componente, lo cual refleja que esos componentes no tienen dependencias dentro del propio subsistema al cual pertenecen y por tanto no existe otro componente dentro de estos subsistemas, que utilice sus servicios.

2.5. Conclusiones

Con el desarrollo de la vista de sistema se logra establecer y detallar los lazos de colaboración entre los distintos componentes, dentro de cada uno de los subsistemas en cuestión. También se obtiene una descripción de los mismos, respecto a sus funcionalidades generales y características. Además se determina el nivel de complejidad y de criticidad que presentan estos componentes desde el punto de vista de la integración; factores que son utilizados por el equipo de desarrollo del proyecto, para puntualizar la prioridad que estos presentan durante el desarrollo de los subsistemas a los cuales pertenecen, determinar la reusabilidad de los mismos y para efectuar un cambio determinado, tanto en estas soluciones, como en el sistema como un todo.

Capítulo III Vista de arquitectura de datos

3.1. Introducción

En este capítulo se realiza un análisis del modelo de datos de cada subsistema, para luego describir el diseño estructural de este modelo, también se define la integración con otros subsistemas a nivel de datos. Todos estos elementos que quedan señalados y detallados en los artefactos que serán explicados a continuación; los cuales forman parte de la vista de arquitectura de datos de los subsistemas, que constituye la tercera y última vista, que conforma la vista de sistema.

3.2. Vista de arquitectura de datos de las soluciones Estructura y Composición, Configuración y Multimoneda

La vista de arquitectura de datos constituye una de las primeras vistas que se debe definir dentro de un proyecto, debido a que especifica los principales tipos y fuentes de datos necesarios para apoyar los procesos de una entidad, cómo deben ser manejados y la forma en que estarán distribuidos, de modo tal que sean entendibles, estables, consistentes y completos. Además establece las pautas, la nomenclatura, los estándares, herramientas y otros elementos no menos importantes, que se deben seguir y aplicar en el diseño y desarrollo de una base de datos y que contribuyen a mejorar el rendimiento, el mantenimiento y la calidad de la misma y al modelado y control del flujo de datos del sistema.

Para el sistema CEDRUX, la vista de datos tiene definido un expediente de datos, que comprende una serie de artefactos que describen el uso y la distribución de todos los datos que se manejan en el sistema, así como las relaciones que existen entre todos los subsistemas por medio de la integración de los datos que estos utilizan. En este documento se analizan y se describen específicamente cuatro de ellos: el Modelo de datos, el Diccionario de datos, la Matriz de trazabilidad de los datos y la Prueba de concepto de diseño.

Este expediente se encuentra organizado del siguiente modo. (Figura 9). En el paquete DAT-Subsistemas_Lineas se encuentran organizadas por paquetes todas las líneas del proyecto, por ejemplo, L_Arquitectura, L_Auditoría L_EstructuraComp, etc. Dentro de estas carpetas se encuentran el o los módulos que integran la línea y dentro de cada módulo están posicionados los artefactos mencionados anteriormente, organizados en paquetes. Por ejemplo en la línea de Estructura y Composición se

encuentra el módulo que la integra (es la carpeta denominada Mod_EstructuraComp) y dentro de este se encuentran los artefactos de datos, correspondientes a dicho esquema. En la línea de Arquitectura se encuentran los artefactos correspondientes al módulo de datos maestros y al resto de los módulos que conforman esta línea.

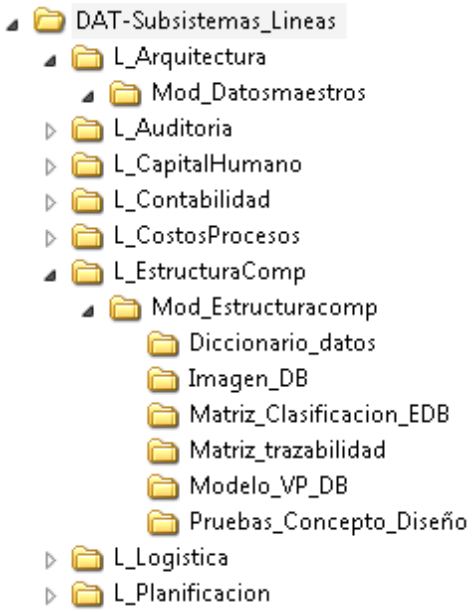

Figura 9 Expediente de datos para los subsistemas.

3.2.1. Modelo de datos

El modelo de datos se realiza con el objetivo de describir las estructuras de datos físicas del sistema; esto se refiere a los objetos que contiene la base de datos y las relaciones que existen entre ellos, además de detallar las operaciones de manipulación y recuperación de información.

3.2.1.1. Estandarización aplicada en el desarrollo de los modelos de datos

Primeramente, antes de proceder a la descripción de los modelos de datos, es importante aclarar algunos de las pautas o estándares que han sido aplicados durante el diseño y desarrollo de los esquemas de datos, que se corresponden a los subsistemas en cuestión, los cuales proveen mayor organización, claridad y entendimiento de la base de datos por parte del resto del equipo de desarrollo de la misma. Acorde a la estandarización aprobada por el proyecto, para las soluciones Estructura y Composición,

Configuración y Multimoneda, se definieron los esquemas mod_estructuracomp y mod_datosmaestros, respectivamente; ambos nombres escritos en minúsculas y haciendo uso del prefijo mod_, que hace referencia al término módulo. Los módulos Configuración y Multimoneda están asociados a un mismo esquema debido a que ambos tratan aspectos vinculados a la configuración del sistema.

Las tablas y campos correspondientes a estos esquemas se crearon, con los nombres en minúsculas y utilizando los siguientes prefijos ya predefinidos: dat_, utilizado en las tablas que guardan la mayor cantidad de características de una entidad; nom_, para las tablas nomencladores; conf_, utilizado en las tablas que guardan parámetros de configuración del sistema; his_, para tablas que guardan datos por largos períodos de tiempo y que son consultados esporádicamente y el prefijo id, para los campos que son identificadores; por ejemplo: dat_estructura, nom_moneda, conf_entidades, his_tasa, idfecha. Los identificadores representan las llaves primarias de las tablas y reciben en su nombre, luego del prefijo id, el nombre de la tabla a la cual pertenecen, por ejemplo: idfecha (llave primaria de la tabla dat_fecha). Para denominar las funciones (se refiere a procedimientos almacenados), funciones de triggers y triggers, se emplearon los prefijos f_, ft_ y t_ respectivamente. Se debe aclarar que estos no son los únicos prefijos delimitados dentro de las políticas de trabajo del proyecto, pero sólo se ha hecho referencia a los utilizados en los esquemas asociados a las soluciones en cuestión. Los tipos de datos establecidos son: numeric, para los identificadores y valores enteros; date, para las fechas; boolean, para valores condicionales; y varchar, para valores que representan cadenas.

Por otra parte, para algunos esquemas, como es el caso de los esquemas asociados a los subsistemas tratados en este trabajo, puede ocurrir que dos o más clientes accedan a un mismo campo, para realizar una determinada acción al mismo tiempo. Esto es lo que se conoce como concurrencia y para el tratamiento de la misma se aplica lo siguiente: es obvio que las solicitudes a un mismo recurso se realizan por orden de llegada, por tanto se efectúa la primera solicitud y se le notifica al cliente, mientras que al cliente de la segunda solicitud se le notifica, de la forma que esté establecida, que la acción no puede ejecutarse. Para llevar este control se decidió incluir un campo denominado “versión”, en las tablas que estén sujetas a esta situación, el cual permitirá conocer las veces que la tupla accedida ha sido modificada.

Otro aspecto importante dentro de los estándares establecidos para el diseño y desarrollo de la base de datos, es el manejo de los datos jerárquicos. Estos constituyen un conjunto de datos donde cada elemento tiene un padre, a excepción del nodo raíz, que no lo tiene, y que puede tener varios o ningún hijo. Esta jerarquía se maneja mediante modelos de conjuntos anidados, donde los padres engloban a sus hijos y se representa en las tablas, con la inclusión de los campos izquierdo y derecho (Lft y Rgt[20], respectivamente), para mostrar el anidado de los nodos. Por tanto una estructura árbol queda modelada con los atributos: Id, que guarda el identificador de la estructura, el idpadre, que guarda la estructura padre, el ordenizq y ordender, que guardan los valores de los hijos izquierdo y derecho de la estructura. Uno de los esquemas en donde se aprecia ejemplos de esto, es el de Estructura y Composición (mod_estructuracomp) que será descrito a continuación. Para obtener más información sobre esto puede verse el documento Decisiones de Datos, elaborado por el proyecto.

3.2.1.2. Subsistema Estructura y Composición

El modelo de datos correspondiente al subsistema Estructura y Composición comprende cuarenta y tres tablas, de las cuales treinta son nomencladores, el resto constituyen las tablas que almacenan las características propias de una entidad. A continuación se describirán algunas de las más importantes.

En la siguiente imagen se muestran las tablas dat_estructura, dat_estructuraop, nom_organo, nom_agrupacion, nom_nivelestr, dat_agrupacionest y dat_agrupacionestop y cómo se relacionan entre sí (Figura 10). La tabla dat_estructura almacena los datos de la estructura externa de la entidad que se esté gestionando. Contiene como atributos el identificador (en lo adelante Id) de la estructura (idestructura), 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 Id del órgano asociado a la misma (idorgano), el Id del nivel estructural que presenta (idnivelestr), el Id del prefijo asociado (idprefijo), el cual representa la especificación de utilidad de dicha entidad, las fechas de inicio y de fin del uso de la tabla (fechaini y fechafin, respectivamente), contiene el campo versión, para el tratamiento de la concurrencia, el identificador del padre de la estructura (idpadre) y los campos lft y rgt, que guardan el hijo izquierdo y el hijo derecho de la estructura. Estos tres últimos atributos están presentes debido a que la tabla maneja

[20] Se refieren a los términos left y right en inglés. Se utilizan de este modo debido a que son palabras reservadas de MySQL.

datos jerárquicos. Lo cual se explicó dentro de la estandarización aplicada en el desarrollo de la base de datos.

La tabla dat_estructuraop almacena los datos referentes a la estructura interna u organización del personal, que presenta una entidad determinada. 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 (idestructuraop) y que se incluye como un atributo más, el identificador de la estructura superior o externa, a la cual se asocia la estructura interna gestionada. Como se menciona anteriormente, tanto la estructura superior como la estructura interna se definen en agrupaciones determinadas, las cuales se encuentran establecidas en el nomenclador nom_agrupacion. Sus atributos constituyen el identificador de la agrupación (idagrupacion), que se corresponde con la llave primaria del nomenclador, el nombre de la agrupación (denagrupacion), la abreviatura (abrevagrupacion) que responde a dicho nombre, el orden que tiene la agrupación y las fechas de inicio y fin del uso del nomenclador (fechaini y fechafin, respectivamente).

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 Id de la agrupación y el Id de la estructura, en el caso de dat_agrupaciones, para el caso de dat_agrupacionestop, es el Id de la estructura interna. El nomenclador nom_organo especifica los distintos órganos que forman parte de las estructuras de las entidades, 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 Id del órgano (idorgano), que también es la llave primaria, el Id del nivel estructural al que pertenece, su nombre (denorgano) y la abreviatura que responde a este (abrevorgano), el orden que tiene el nomenclador y las fechas de inicio y fin del uso del mismo (fechaini y fechafin, respectivamente). 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 entidades, por ejemplo nivel 1, nivel 2, nivel 4, etc. Dentro de sus atributos se encuentran la denominación del nivel (denivelestr) y la abreviatura que recibe la misma (abrevnivelestr), el orden del nomenclador y las fechas de inicio y fin del uso del mismo (fechaini y fechafin, respectivamente).

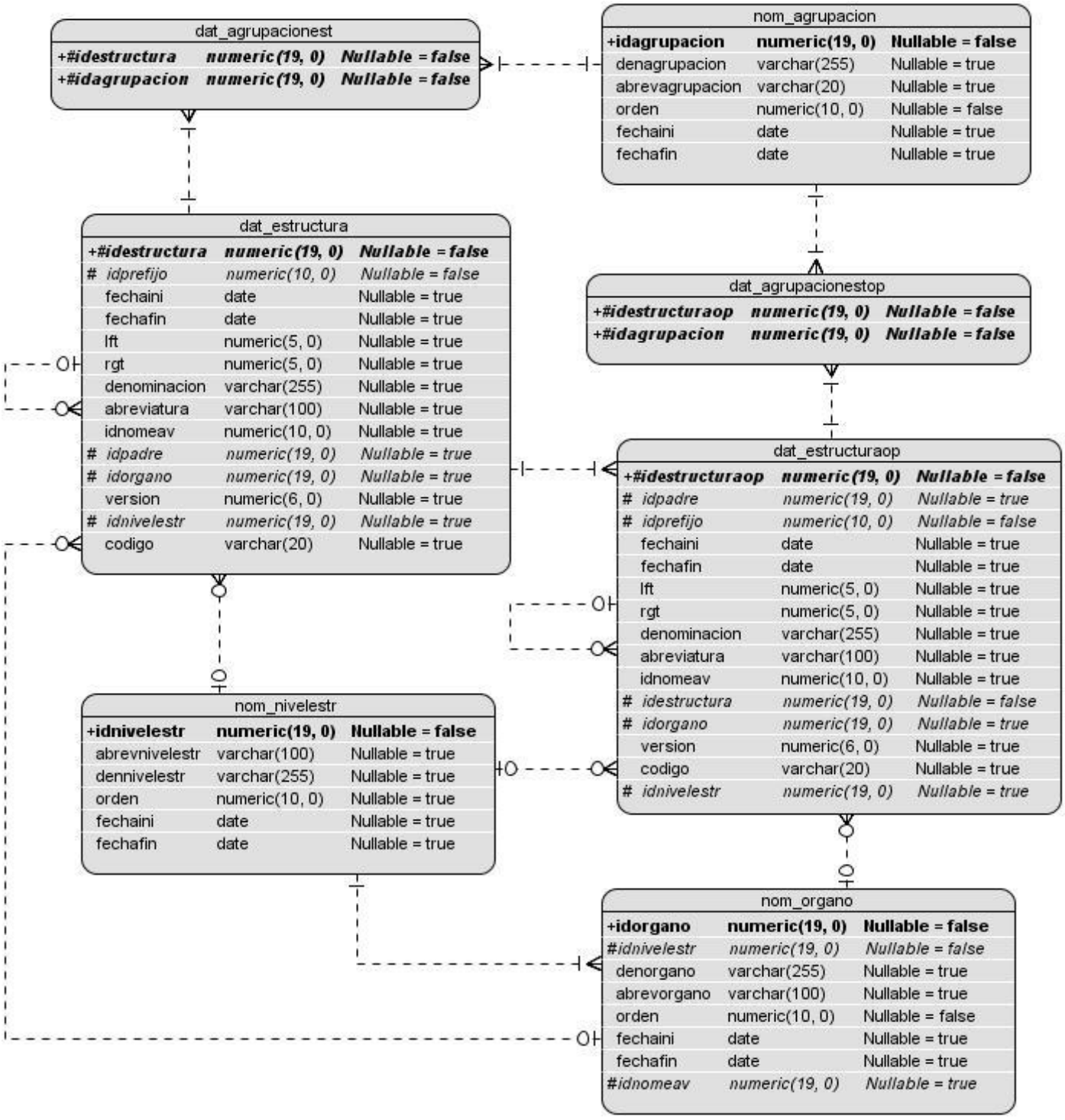

Figura 10 Fragmento del Modelo de datos del subsistema Estructura y Composición

3.2.1.3. Subsistemas Configuración y Multimoneda

Los subsistemas Configuración y Multimoneda están representados en un mismo modelo de datos, se reitera que esto ocurre debido a que toda la información que manejan ambos subsistemas, se almacena dentro de un mismo esquema denominado mod_datosmaestros. El modelo relaciona cuarenta y siete tablas, de las cuales catorce son nomencladores y dos son historiales. A continuación se describen algunas de las tablas más importantes.

La siguiente imagen relaciona las tablas dat_estructurasubsist, dat_ejerciciocontable, dat_periodocontable, dat_cierre, nom_moneda, dat_moneda, dat_dezglozemoneda y dat_nommoneda_entidad. (Figura 11). La tabla dat_ estructurasubsist guarda la relación entre las tablas dat_estructura, del esquema estructura y composición, con nom_subsistema, propia del esquema datos maestros y se utiliza para controlar los subsistemas que contiene una entidad y en que estado se encuentran. Por ello comprende entre sus campos, además del identificador de la estructura (idestructurasubsist), que es la llave primaria de la tabla, el Id del subsistema asociado a la estructura (idsubsistema), el Id del estado que este presenta (idestado), también contiene el campo “versión” para el tratamiento de la concurrencia, el campo “aperturado”, que guarda si el subsistema está o no aperturado y un campo “contabiliza”, que guarda si el subsistema está o no contabilizado. Cada subsistema dentro de una entidad realiza sus cierres o cambios de períodos contables. Estos cambios se almacenan en la tabla dat_cierre, de ahí surge su relación. A su vez dat_cierre permite conocer el ejercicio y el período actual contable en que se encuentra un subsistema. Contiene como atributos el Id de la estructura (idestructurasubsist), el Id del subsistema que realiza el cierre (idsubsistema), los Id del período y del ejercicio actual del subsistema en cuestión (idperiodoactual e idejercicioactual, respectivamente), la descripción del cierre y un último atributo denominado “cierrediario”, que guarda si ocurre o no el cierre.

La tabla dat_ejerciciocontable se emplea para guardar los datos de los ejercicios contables pertenecientes a una entidad. Comprende como atributos el Id del ejercicio (idejercicio), que es la llave primaria de la tabla, el nombre del ejercicio, el orden que presenta el mismo con respecto a otros ejercicios (antecesor), registra si el año en que este se desarrolla es un año natural o no (annonatural[21]), las fechas de inicio y fin

[21] Se refiere a una fecha de orden natural. Ej.: 2009, 2010.

del ejercicio (inicio y fin, respectivamente) y guarda además, el atributo “versión”. Esta tabla se relaciona con la tabla dat_periodocontable, debido a que cada ejercicio engloba uno o más períodos. Esta tabla almacena los datos de los períodos contables pertenecientes a una entidad y contiene, además del Id del período (idperiodo), los mismos campos que la tabla anterior, con excepción de los campos “annonatural” y “antecesor”, y con la inclusión del Id del ejercicio (idejercicio) al cual pertenece el período en cuestión.

Por su parte, el nomenclador nom_moneda almacena los datos propios de las monedas que existen sin importar su nacionalidad. Especifica de cada moneda el Id, que es la llave primaria del nomenclador (idmoneda), el nombre y el código ISO de la misma, así como el código que se le asigna. Cada entidad u organización selecciona de este nomenclador las monedas específicas a utilizar. Estos datos se guardan en la tabla dat_nommoneda_entidad, que contiene en sus atributos el Id que presenta la moneda seleccionada en el nomenclador nom_moneda (idmoneda) y que por ende establece la relación entre estas tablas, el Id que se le asigna como moneda propia de esa entidad (idnommonedaentidad), que es la llave primaria y el campo “versión”. Con esta tabla se manejan los elementos del nomenclador de monedas, de un modo más viable, puesto que sólo se definen las de interés para cada entidad. La tabla dat_dezglozemoneda permite asociar el desglose de monedas que aplicará una entidad para las monedas que utiliza, por ejemplo monedas de 20 céntimos, billetes de 1, billetes de 3, etc. De este modo guarda la relación entre el nomenclador de monedas y el del desglose de monedas. Por lo cual contiene el Id del desglose seleccionado (iddezglozemonedas), el Id de la moneda a la cual se aplica el desglose (idmoneda) y el Id que se le asigna a ese desglose como propio de la entidad (iddezglozemoneda). Por último se muestra la tabla dat_moneda, que almacena los datos de la moneda que se definan como contables y/o alternativas, sobre las cuales se expresa el valor de la mayoría de las transacciones que se realicen en una organización. Tiene como atributos el Id de la moneda seleccionada del nomenclador (idmoneda), el Id que se le asigna como moneda contable (idmonedacontable), la nacionalidad de la misma (idpais), una descripción, guarda si esa moneda es alternativa o no (monaltern) y el campo “versión”, para tratar la concurrencia.

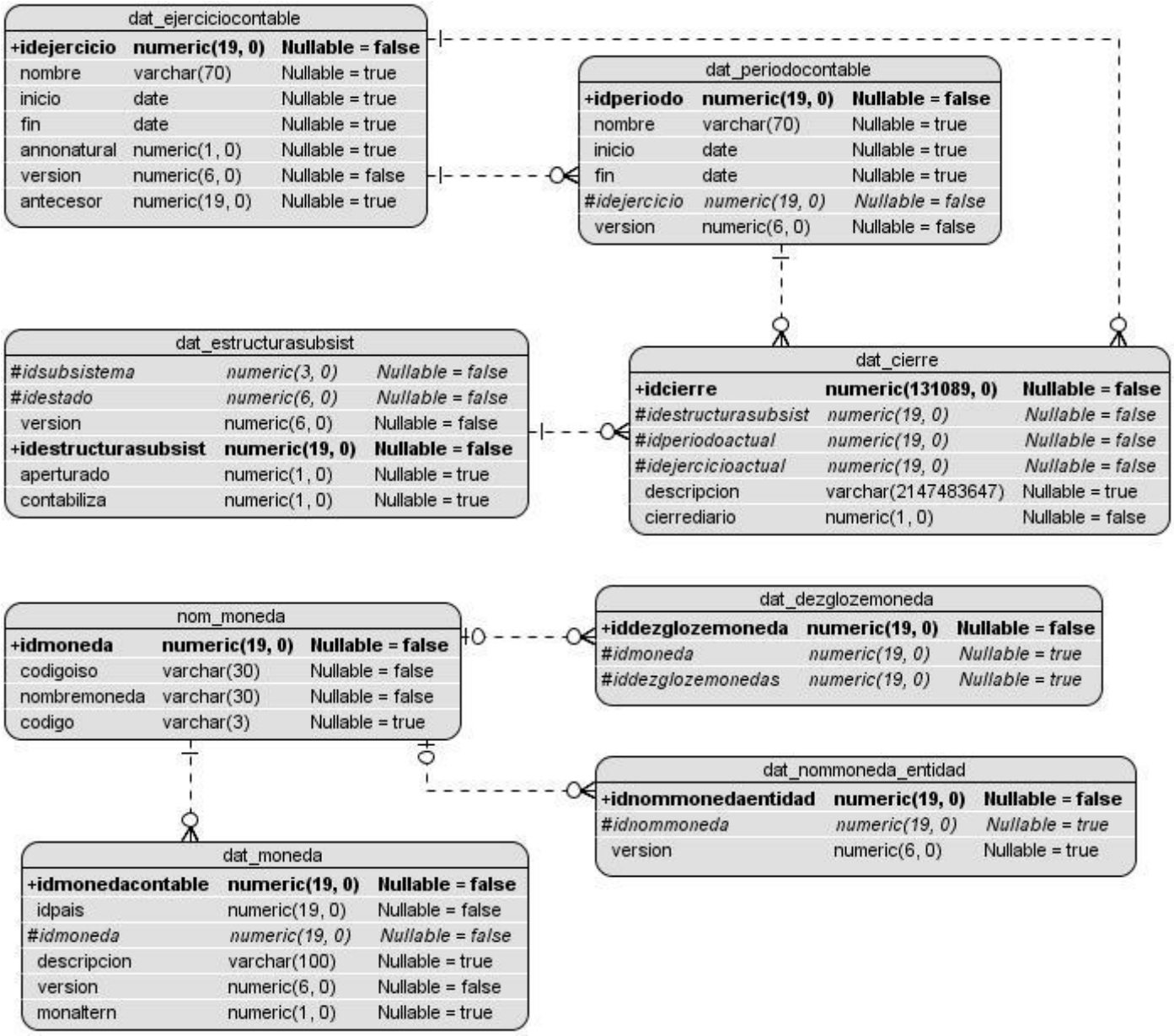

Figura 11 Fragmento del Modelo de datos del esquema datos maestros, asociado a Configuración y Multimoneda

3.2.2. Diccionario de datos

El diccionario de datos contiene una descripción detallada de las tablas, atributos y relaciones del subsistema, así como de los procedimientos almacenados y las funciones disparadoras, que responden a

las distintas operaciones que se ejecutan sobre la información en la base de datos. Cada diccionario contiene primeramente una breve descripción del subsistema y seguidamente, la descripción de sus tablas y funciones. A continuación se muestra un ejemplo con la tabla dat_cierre, del esquema de datos maestros, correspondiente a los subsistemas de Configuración y Multimoneda:

1. dat_cierre

Tabla 1 Descripción de la tabla dat_cierre

Nombre	Valor
Datos Modelo	Physical
Documentación	Tabla que contiene los cambios de períodos contables de los subsistemas, permitiendo conocer el ejercicio contable y el período contable en que se encuentran actualmente dichos subsistemas, en una entidad.
Esquema	mod_datosmaestros
Primary Key Constraint Name	dat_cierre_pkey

1.1 Columnas

Tabla 2 Descripción de los campos de la tabla dat_cierre

Nombre	Date type	Limitaciones	Anulable	Documentación
idcierre	Numeric (131089)	PK	No	Llave primaria de la tabla y guarda el identificador de los cierres.
idestructurasubsist	Numeric (19)	FK (dat_estructurasubsist.idestructurasubsist)	No	Llave foránea proveniente de la tabla dat_estructurasubsist e identifica la estructura y el subsistema de la misma, al cual se asocia el periodo referenciado.
idperiodoactual	Numeric (19)	FK (dat_periodocontable.idperiodo)	No	Llave foránea proveniente de la tabla dat_periodocontable y guarda el periodo actual.
idejercicioactual	Numeric (19)	FK (dat_ejerciciocontable.idejercicio)	No	Llave foránea proveniente de la tabla dat_ejerciciocontable y guarda el ejercicio actual.
descripcion	Varchar		Sí	Guarda una descripción del cierre.

	(2147483647)			
cierrediario	Numeric (1)		Sí	Guarda si ocurre el cierre o no.

1.1.1 Relaciones

Tabla 3 Descripción de las relaciones de la tabla dat_cierre

dat_cierre_fk : Relación	
Desde	dat_estructurasubsist
Documentación	Relación de uno a mucho entre las tablas dat_estructurasubsist y dat_cierre, que guarda el identificador idestructurasubsist, como llave primaria en la tabla dat_estructurasubsist y como llave foránea en la tabla dat_cierre. Permite asociarle a la estructura referenciada y al subsistema de la misma, el período y el ejercicio contable actual en que se encuentra dicho subsistema.
Identifying	false
On Delete	No action
On Update	No action
To Multiplicity	0..*
From Multiplicity	1
Sync To Association	0
Datos Modelo	2

Tabla 4 Descripción de las relaciones de la tabla dat_cierre

fkdat_cierre816948 : Relación	
Desde	dat_periodocontable
Documentación	Relación de uno a mucho entre las tablas dat_periodocontable y dat_cierre, que guarda el identificador idperiodo, como llave primaria en la tabla dat_periodocontable y el identificador idperiodoactual, como llave foránea en la tabla dat_cierre. Permite asociarle a un subsistema determinado el período contable en que se encuentra dicho subsistema.
Identifying	false
On Delete	No action

On Update	No action
To Multiplicity	0..*
From Multiplicity	1
Sync To Association	0
Datos Modelo	2

Tabla 5 Descripción de las relaciones de la tabla dat_cierre

fkdat_cierre147677 : Relación	
Desde	dat_ejerciciocontable
Documentación	Relación de uno a mucho entre las tablas dat_ejerciciocontable y dat_cierre, que guarda el identificador idejercicio, como llave primaria en la tabla dat_ejerciciocontable y el identificador idejercicioactual, como llave foránea en la tabla dat_cierre. Permite asociarle al subsistema referenciado el ejercicio contable en que se encuentra dicho subsistema.
Identifying	false
On Delete	No action
On Update	No action
To Multiplicity	0..*
From Multiplicity	1
Sync To Association	0
Datos Modelo	2

3.2.3. Matriz de trazabilidad.

La matriz de trazabilidad de los datos permite establecer las relaciones que existen entre las tablas de los diferentes subsistemas. En este artefacto se muestran los distintos esquemas, que representan a las líneas del proyecto, especificando en las intercepciones, la tabla de origen y la tabla de destino entre las que se establece la relación y mediante que variable o campo específico, el valor que tiene por defecto dicha variable, la cardinalidad que presenta la relación (uno a uno, uno a mucho, mucho a mucho) y el estado de la integración (hecho, no hecho, en espera de información).

3.2.4. Prueba de concepto de diseño

Este artefacto contiene los distintos escenarios de prueba y las funciones de un módulo determinado, además de los resultados proyectados por este, respecto al tiempo total de ejecución de una función, el costo de la misma y las filas buscadas o retornadas por esta. Estas pruebas se realizan con el fin de validar el diseño de la base de datos y el negocio implementado en la misma.

Específicamente en estos subsistemas se realizaron estas pruebas sobre los requisitos que responden a la generación de varios reportes que engloban una serie de informaciones que pueden ser de interés para los usuarios. A continuación se muestra como ejemplo la prueba realizada a uno de los reportes implementados en el componente Estructura y Composición.

1- Escenario de Prueba del diseño para generar el reporte Cantidad de Entidades y Unidades por Agrupaciones de un Nivel 1.

El sistema requiere el nombre y el código del nivel para generar un reporte que tiene como objetivo mostrar de un nivel 1: un número consecutivo para numerar las agrupaciones del nivel 1, el código ONE de la entidad que representa al nivel 1, la denominación de las agrupaciones pertenecientes a ese nivel 1, la abreviatura de cada agrupación, la clasificación, la categoría, la cantidad de entidades subordinadas a cada agrupación y la cantidad de unidades subordinadas a cada agrupación. Para ello se utilizan las tablas dat_estructura, que guarda la información referente a la estructura externa de una entidad determinada, nom_organo, nomenclador que guarda los órganos, que forman parte de las estructuras externas de las entidades, acorde al nivel estructural al que pertenecen, nom_filaestruc, que permite definir las estructuras que son creadas para las entidades y nom_valorestruc que guarda la relación de mucho a mucho entre las tablas nom_filaestruc y nom_campoestruc y el valor que tiene dicho campo. La tabla nom_campoestruc guarda el nombre de los campos que se van a generar dinámicamente en la tabla nomenclador, por ejemplo: localidad, categoría. Para realizar la prueba se insertaron alrededor de 100 mil tuplas y se realizó la consulta pertinente a la función que responde a este requisito.

Consulta: **EXPLAIN ANALYZE SELECT * from** mod_estructuracomp.cant_entida_unidades_agrupacion (nombre_nivel varchar, codigo_nivel varchar)

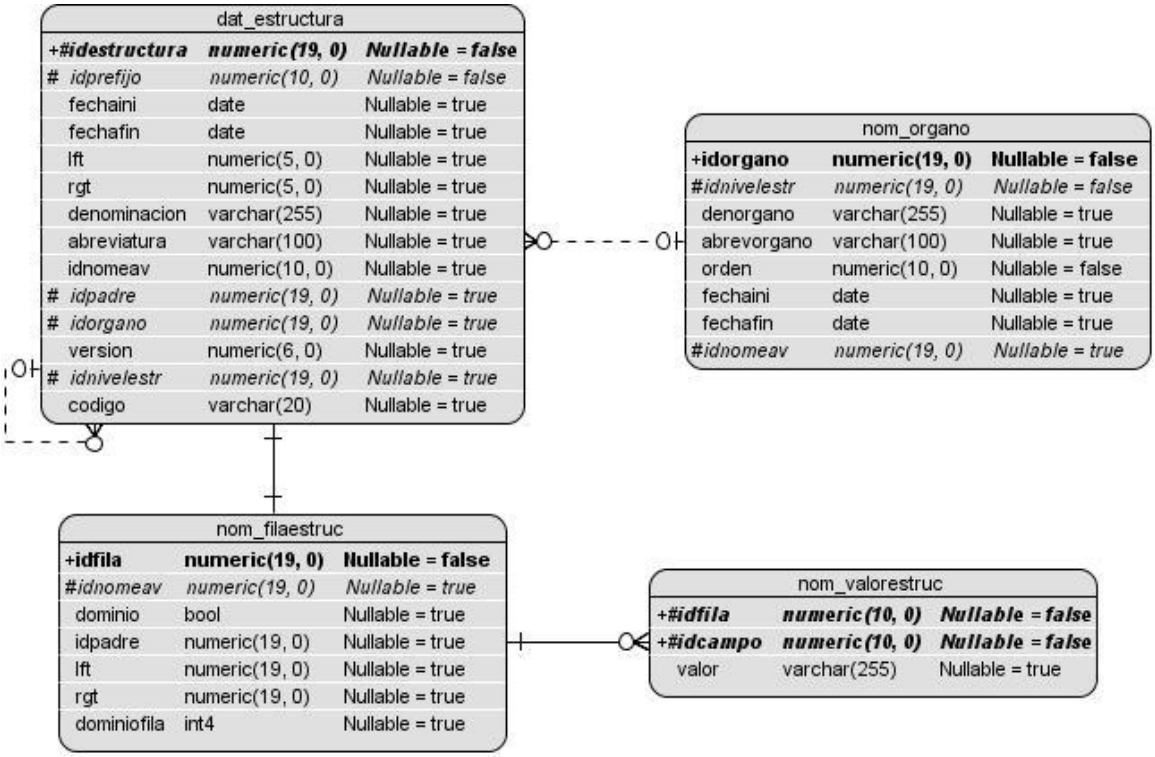

Figura 12 Escenario de prueba del requisito evaluado

Resultados de la prueba: (cost=0.00..260.00 rows=1000 width=3052) (actual time=2.836..2.840 rows=4 loops=1)

Total runtime: 2.924 ms (Este dato refleja el tiempo total de ejecución de una consulta).

Tuplas recuperadas: 2

Aquí queda reflejado el costo (cost) que presenta la función respecto al tiempo real que tarda el gestor de base de datos en ejecutar esta transacción (actual time). Analizando dicho resultado puede afirmase que la base de datos tiene un tiempo de respuesta bajo al ejecutar dicha función, puesto que el resultado de la consulta retorna dos tuplas en un tiempo de 2.924 ms.

3.3. Conclusiones

Con el desarrollo de la vista de datos se obtiene la descripción detallada de los distintos modelos de datos correspondientes a los subsistemas en cuestión, díganse la descripción de las tablas, sus atributos, las relaciones entre estas, entre otros elementos. Por consiguiente se logra especificar la integración, a nivel de datos, entre estos y el resto de los subsistemas que conforman el sistema CEDRUX. Además fue posible verificar que en los esquemas de datos correspondientes a dichos subsistemas, se emplearon correctamente los estándares establecidos por el proyecto para el diseño y desarrollo de la base de datos.

Conclusiones Generales

Con la realización del presente trabajo se concluye, primeramente que el marco teórico de la investigación permitió profundizar los conocimientos relacionados con la descripción de la arquitectura de software, para aplicarlos adecuadamente en el marco conceptual de las soluciones propuestas en esta investigación, además de obtener los fundamentos y conocimientos requeridos para verificar y validar que en el desarrollo de las mismas se sigan los lineamientos y/o estándares definidos por los grandes exponentes del tema.

Con el desarrollo de la vista de arquitectura de sistema se logró detallar la organización que presentan los subsistemas en cuestión, basada en los componentes que los integran, sus funcionalidades y en las relaciones de comunicación y colaboración existentes entre estos. También se determinó la importancia que presentan dichos componentes desde el punto de vista de la integración y su complejidad de implementación, lo cual permite evaluar hasta que punto son reutilizables estas soluciones, evaluar la interoperabilidad y usabilidad de las mismas y especificar cuáles componentes deben ser priorizados en el desarrollo.

El desarrollo de la vista de arquitectura de datos permitió obtener la descripción de los modelos de datos correspondientes a los módulos descritos en este trabajo, lo suficientemente explícita. Lo cual permitió especificar las relaciones que existen entre estos y el resto de los subsistemas que integran el sistema CEDRUX a nivel de datos y realizar una validación del diseño de los mismos y de la implementación de los elementos del negocio manejados y representados en sus esquemas, dentro de la base de datos.

Por consiguiente estas vistas contribuyen a obtener un mayor entendimiento de estas soluciones y mejoran las relaciones colaborativas dentro del equipo de trabajo implicado propiamente en el desarrollo y evolución de las mismas, como en todo el personal implicado en el desarrollo del proyecto. También permiten comprobar la escalabilidad, robustez y el rendimiento de dichas soluciones, organizar todo el proceso de desarrollo y hacer evolucionar el sistema como un todo.

Recomendaciones

- ⇨ Utilizar adecuadamente las vistas de sistema y de datos desarrolladas como guía de apoyo para brindar soporte a los subsistemas en cuestión, como material de referencia para el repositorio y para la capacitación de cualquier personal que se integre al proyecto.
- ⇨ Realizar las pruebas de concepto de diseño al resto de los requisitos implementados por cada uno de los subsistemas en cuestión.

Referencias Bibliográficas

1. **IEEE.** *Standard 610 12-1990 Glossary of Software Engineering Terminology.* New York : s.n., 1990. ISBN 1-55937-067-X.

2. **CEIGE.** *Modelo de Desarrollo orientado a componentes del proyecto ERP Cuba.* 2008.

3. —. *Ciclo de vida del proyecto.* 2009.

4. **IEEE.** *Recommended Practice for Architectural Description of Software-Intensive Systems.* New York : s.n., 2000. ISBN 0-7381-2519-9.

5. **Reynoso, Carlos Billy.** *Introducción a la Arquitectura de Software.* Universidad de Buenos Aires : s.n., 2004.

6. **Microsoft Corporation.** *La Arquitectura Orientada a Servicios (SOA) de Microsoft aplicada al mundo real.* 2006.

7. **Peláez, Juan.** Geeks.ms: ARQUITECTURA BASADA EN COMPONENTES. *Geeks.ms.* [En línea] 2009. http://geeks.ms/blogs/jkpelaez/archive/2009/04/18/arquitectura-basada-en-componentes.aspx.

8. **Carlos Reynoso, Nicolás Kicillof.** *Estilos y Patrones en la Estrategia de Arquitectura de Microsoft.* Universidad de Buenos Aires : s.n., 2004.

9. **Lago, Ramiro.** proactiva-calidad: Patrones de diseño software. *proactiva-calidad.* [En línea] 2007. http://www.proactiva-calidad.com/java/patrones/index.html.

10. **Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides.** *Design Patterns - Elements of Reusable Object-Oriented Software.* 1995.

11. **Martínez, Luis F. Iribarne.** *Modelo de Mediacion para el Desarrollo de Software basado en Componentes COTS.* Universidad de Almería : s.n., 2003.

12. **Mendez, Oscar.** Desarrollo de Software basado en Componentes. *msdn.microsoft.com.* [En línea] http://msdn.microsoft.com/es-es/library/bb972268.aspx

13. **Toledano, Moisés Daniel Díaz.** Cómo desarrollar una arquitectura software: los lenguajes de patrones. *www.moisesdaniel.com.* [En línea] http://www.moisesdaniel.com/es/wri/ComoDesArqSoft.htm.

14. ingsoftiush.blogspot: Ingenieria del Software IUSH: Diseño arquitectónico. *ingsoftiush.blogspot.* [En línea] 2010. http://ingsoftiush.blogspot.com/2010/03/diseno-arquitectonico.html.

15. **Kruchten, Philippe.** *Planos Arquitectónicos: El Modelo de "4+1" Vistas de la Arquitectura de Software.*

16. LA ARQUITECTURA DE SOFTWARE EL MODELO 4+1. *sites.google.com.* [En línea] http://sites.google.com/site/jgarzas/4mas1.

17. **Gómez, Omar.** Documentando la arquitectura de software. *osgg.net.* [En línea] http://osgg.net/omarsite/resources/papers/doc_arch.pdf.

18. Arquitectura Empresarial con TOGAF. *www.mug.org.ar.* [En línea] www.mug.org.ar/Descargas/Jornadas/Downloads_GetFile.aspx?id=3389

19. **CEIGE. Osmar Leyet Fernández.** *DOCUMENTO DE DESCRIPCION DE LA ARQUITECTURA DE SOFTWARE.* 2010.

Bibliografía

1. **IEEE.** *Standard 610 12-1990 Glossary of Software Engineering Terminology.* New York : s.n., 1990. ISBN 1-55937-067-X.

2. **CEIGE.** *Modelo de Desarrollo orientado a componentes del proyecto ERP Cuba.* 2008.

3. —. *Ciclo de vida del proyecto.* 2009.

4. **IEEE.** *Recommended Practice for Architectural Description of Software-Intensive Systems.* New York : s.n., 2000. ISBN 0-7381-2519-9.

5. **Reynoso, Carlos Billy.** *Introducción a la Arquitectura de Software.* Universidad de Buenos Aires : s.n., 2004.

6. **Microsoft Corporation.** *La Arquitectura Orientada a Servicios (SOA) de Microsoft aplicada al mundo real.* 2006.

7. **Carlos Reynoso, Nicolás Kicillof.** *Estilos y Patrones en la Estrategia de Arquitectura de Microsoft.* Universidad de Buenos Aires : s.n., 2004.

8. **Peláez, Juan.** Geeks.ms: ARQUITECTURA BASADA EN COMPONENTES. *Geeks.ms.* [En línea] 2009. http://geeks.ms/blogs/jkpelaez/archive/2009/04/18/arquitectura-basada-en-componentes.aspx.

9. **Martínez, Luis F. Iribarne.** *Modelo de Mediacion para el Desarrollo de Software basado en Componentes COTS.* Universidad de Almería : s.n., 2003.

10. **Toledano, Moisés Daniel Díaz.** Cómo desarrollar una arquitectura software: los lenguajes de patrones. *www.moisesdaniel.com.* [En línea] http://www.moisesdaniel.com/es/wri/ComoDesArqSoft.htm.

11. ingsoftiush.blogspot: Ingenieria del Software IUSH: Diseño arquitectónico. *ingsoftiush.blogspot.* [En línea] 2010. http://ingsoftiush.blogspot.com/2010/03/diseno-arquitectonico.html.

12. **Kruchten, Philippe.** *Planos Arquitectónicos: El Modelo de "4+1" Vistas de la Arquitectura de Software.*

13. LA ARQUITECTURA DE SOFTWARE EL MODELO 4+1. *sites.google.com.* [En línea] http://sites.google.com/site/jgarzas/4mas1.

14. **Computación, Departamento de Sistemas Informáticos y.** *Proceso de desarrollo de software.* Universidad Politécnica de Valencia : s.n.

15. Qualitrain - Metodologías Ágiles de Desarrollo de Software. *www.qualitrain.com.* [En línea] http://www.qualitrain.com.mx/Metodologias-Agiles-de-Desarrollo-de-Software-Segunda-Parte.html.

16. **Patricio Letelier, Mª Carmen Penadés.** *Métodologías ágiles para el desarrollo de software: eXtreme Programming (XP).* Universidad Politécnica de Valencia : s.n.

17. **Ivar Jacobson, Grady Booch, James Rumbaugh.** *Proceso Unificado de Desarrollo de Software.* ISBN: 84-7829-036-2.

18. Manual de Usuario de Openbravo. *docs.huihoo.com.* [En línea] http://docs.huihoo.com/openbravo/Openbravo-Manual-de-Usuario-v1.1.pdf.

19. www.openbravo.com. [En línea] http://www.openbravo.com/es/product/erp/ke.

20. ERP OPENBRAVO. Software libre para la gestión empresarial. *www.consoltic.com.* [En línea] http://www.consoltic.com/productos/consoltic_erp_openbravo/.

21. Adempiere, solución empresarial de código abierto. *www.aplicacionesempresariales.com.* [En línea] http://www.aplicacionesempresariales.com/adempiere-solucion-empresarial-de-codigo-abierto.html.

22. **OPENBIZ.** Vista General ADempiere Business Solution. *www.openbiz.com.* [En línea] http://www.openbiz.com.ar/ADempiere%20Business%20Solution%20-%20Vista%20General.pdf.

23. SAP R/3 DE PROYECTOS INFORMATICOS A PROYECTOS EMPRESARIALES. *www.abap.es.* [En línea] http://www.abap.es/centro_FAQ_cap1.htm.

24. El Sistema R/3 de SAP AG. *catarina.udlap.mx.* [En línea] http://catarina.udlap.mx/u_dl_a/tales/documentos/lis/hernandez_c_ej/capitulo1.pdf.

25. RUP vs. XP. *www.usmp.edu.pe.* [En línea] http://www.usmp.edu.pe/publicaciones/boletin/fia/info49/articulos/RUP%20vs.%20XP.pdf.

26. Consultoría en Metodologías de Desarrollo de Software - RUP y las mejores prácticas para el desarrollo de software. *www.histaintl.com.* [En línea] http://www.histaintl.com/servicios/consulting/rup.php.

27. Tutorial IT blog: ¿Que es RUP? *www.conexionit.com.* [En línea] http://www.conexionit.com/blog/metodologias/que-es-rup.html.

28. ADempiere ADempiere [El ERP/CRM Open Source]. *www.adempiere.com.* [En línea] http://www.adempiere.com/images/a/ac/3circles_letter_sp.pdf.

29. **Pressman, Roger S.** *Ingenieria del Software un enfoque practico.* Quinta Edición.

30. **b2b Solutions Group.** Arquitectura Orientada a Servicios. *www.b2bsg.com.* [En línea] http://www.b2bsg.com/pdf/ArquitecturaOrientadaaServicios.pdf.

31. Arquitectura Orientada a Servicios (SOA). *www.tcpsi.com.* [En línea] http://www.tcpsi.com/download/Hoja_SOA.pdf.

32. Tutorial de Java - Arquitectura MVC. *sunsite.dcc.uchile.cl.* [En línea] http://sunsite.dcc.uchile.cl/java/docs/JavaTut/Apendice/arq_mvc.html.

33. **Andrés Vignaga, Daniel Perovich.** *Enfoque Metodológico para el Desarrollo Basado en Componentes.* Facultad de Ingeniería, Universidad de la República : s.n.

34. **Lidia Fuentes, José M. Troya, Antonio Vallecillo.** *Desarrollo de Software Basado en Componentes.* Dept. Lenguajes y Ciencias de la Computací´on. Universidad de M´alaga. : s.n.

35. **Maribel Ariza Rojas, Juan Carlos Molina García.** *INTRODUCCIÓN Y PRINCIPIOS BÁSICOS DEL DESARROLLO DE SOFTWARE BASADO EN COMPONENTES .* 2004.

36. **Vázquez, José María Morales.** *Arquitectura de Sistemas: Un enfoque Evolutivo.*

37. Arquitectura basada en componentes. *www.monografias.com.* [En línea] http://www.monografias.com/trabajos14/aplicacion-distrib/a.

38. Usabilidad y arquitectura del software. *www.desarrolloweb.com.* [En línea] http://www.desarrolloweb.com/articulos/1622.php.

39. Arquitectura basada en componentes. *www.scribd.com.* [En línea] http://www.scribd.com/doc/14704374/Arquitectura-Basada-en-Componentes.

40. Diseño del sistema. Generalidades. Fundamentos del diseño de software. *epcc.unex.es.* [En línea]

41. **Unidad de Compatibilización Integración y Desarrollo De Software para la Defensa.** *Proceso de Desarrollo y Gestión de Proyectos de Software.* 2009.

42. **CEIGE.** *MODELO DE PRODUCCIÓN PARA EL FLUJO DE ARQUITECTURA DE SISTEMA DEL PROYECTO ERP-CUBA.* 2009.

43. —. *DOCUMENTO LÍNEA BASE DE PROYECTO.*

44. **Cristiá, Maximiliano.** *Introducción a la Arquitectura de Software.* 2007.

45. *Integración e Intercambio (The Arquitecture Journal).* **Anna Liu, Ian Gorton.** 5,

46. Patrón Inversión de Control (IoC). *glnconsultora.com.* [En línea] http://glnconsultora.com/blog/?p=3.

47. **Gómez, Omar.** Documentando la arquitectura de software. *osgg.net.* [En línea] http://osgg.net/omarsite/resources/papers/doc_arch.pdf.

48. **Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides.** *Design Patterns - Elements of Reusable Object-Oriented Software.* 1995.

49. Arquitectura Empresarial con TOGAF. *www.mug.org.ar.* [En línea] www.mug.org.ar/Descargas/Jornadas/Downloads_GetFile.aspx?id=3389

48. **Gómez, Omar.** Documentando la arquitectura de software. *osgg.net.* [En línea] http://osgg.net/omarsite/resources/papers/doc_arch.pdf.

Anexos

Glosario de términos

Capacidad de gestión (o administración): define que tan fácil es gestionar la aplicación, usualmente a través de una instrumentación suficiente y adecuada que se expone en un sistema de monitoreo para efectos mejoramiento del rendimiento e identificación de errores.

Componente IBM: Una implementación que (a) realiza un conjunto de funciones relacionadas, (b) puede ser independientemente desarrollado, entregado e instalado, (c) tiene un conjunto de interfaces para los servicios proporcionados y otro para los servicios requeridos, (d) permite tener acceso a los datos y al comportamiento sólo a través de sus interfaces, (e) opcionalmente admite una configuración controlada. (Arquitectura WSBC (WebSphere Business Components) de IBM (*International Business Machines*))

Componente SEI: Un componente software es un fragmento de un sistema software que puede ser ensamblado con otros fragmentos para formar piezas más grandes o aplicaciones completas. (Instituto de Ingeniería del Software)

Componente Szyperski: Un componente es una unidad binaria de composición de aplicaciones software, que posee un conjunto de interfaces y un conjunto de requisitos y que ha de poder ser desarrollado, adquirido, incorporado al sistema y compuesto con otros componentes de forma independiente, en tiempo y espacio. (Clemens Szyperski)

Componibilidad: Está relacionada con la noción de extensibilidad que hace alusión a la capacidad de añadir funciones a un sistema sin necesidad de emplear una cirugía radical, sin causar un gran impacto en el sistema como un todo.

Disponibilidad: Define la proporción del tiempo en que el sistema es funcional y trabaja. Puede ser medido como un porcentaje del tiempo total en que el sistema estuvo caído en un período predefinido. La disponibilidad puede verse afectada por errores del sistema, problemas de infraestructura, ataques o carga del sistema.

Eficiencia: Es el logro de los objetivos y metas con el mínimo de los recursos y tiempo. Es el resultado del mejor aprovechamiento de los recursos utilizados para la realización de las actividades que se prevén a fin del cumplimiento de una meta o acción determinadas.

Escalabilidad: Es la habilidad de un sistema, una red o un proceso, que indica su habilidad para extender el margen de operaciones sin perder calidad, o bien manejar el crecimiento continuo de trabajo de manera fluida, o bien para estar preparado para hacerse más grande sin perder calidad en los servicios ofrecidos.

Fiabilidad: Es la capacidad del software de mantener su nivel de prestación bajo condiciones determinadas durante un período de tiempo establecido.

Flexibilidad: Es la habilidad del sistema para adaptarse a ambientes y situaciones variables y para soportar cambios en políticas de negocios y reglas de negocio. Un sistema flexible es uno que es fácil de reconfigurar o que se adapta en respuesta a los diferentes requerimientos de usuarios y del sistema.

Funcionalidad: Coherencia entre las necesidades detectadas y los resultados que se obtienen con el uso del material. Es lo que un producto puede hacer. Probar la funcionalidad significa asegurar que el producto funciona tal como estaba especificado.

Interoperabilidad: Es la habilidad de que diversos componentes de un sistema y/o de diferentes sistemas funcionen correctamente al intercambiar información, comúnmente por medio de servicios. Un sistema interoperable hace fácil intercambiar y usar información interna y externamente.

Introspección: Forma de aislar el componente del mundo exterior, centrándonos solamente en lo qué hace y cómo lo hace.

Modificabilidad: Capacidad que tiene el sistema de ser modificable, de cambiar, transformarse.

Modularidad: Característica de un sistema formado por la unión de varias partes que interactúan entre sí, denominadas módulos y que trabajan para alcanzar un objetivo común, realizando cada una de ellas una tarea necesaria para la consecución de dicho objetivo.

Rendimiento: Es un indicador de la capacidad de respuesta del sistema para ejecutar una acción dentro de un intervalo de tiempo dado. Puede ser medida en términos de latencia o de respuesta. Latencia es el tiempo que tarda en responder a un evento, respuesta es el número de eventos que tiene lugar en una cantidad dada de tiempo.

Reusabilidad: Capacidad de un componente y un subsistema para ser usado por otras aplicaciones en otros escenarios. Esta minimiza la duplicación de componentes así como el tiempo de implementación.

Reutilización: Ver Reusabilidad.

Taxativamente: Que limita, circunscribe y reduce un caso a determinadas circunstancias. Que no admite discusión.

Usabilidad: Define que tan bien la aplicación cumple con los requerimientos de los usuarios y los consumidores al ser intuitiva, fácil de localizar y globalizar, y capaz de proveer acceso correcto para usuarios con discapacidad así como una experiencia general buena para el usuario.