Universidad de las Ciencias Informáticas

Facultad 3

Título: Arquitectura del Sistema Nacional Público para el Seguimiento de Inversiones y Sectores (SINAPSIS).

Trabajo de Diploma para optar por el título de

Ingeniero en Ciencias Informáticas

Autores: George Pérez Concepción
José Sevila Hidalgo

Tutor: Ing. Daynier Ruiz Rodríguez.

Junio 2009

DECLARACIÓN DE AUTORÍA

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

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

George Pérez Concepción	**José Sevila Hidalgo**	**Daynier Ruiz Rodríguez**
_____	_____	_____
Firma del Autor	Firma del Autor	Firma del Tutor

DEDICATORIA

A mis padres, espero estén orgullosos.

A mi hermano, al cual espero ver realizar lo mismo algún día.

A mi familia toda, en donde quiera que estén, en especial a mis tías Isa y Mercy y a mi prima Betty, las quiero mucho.

A Camilo, Rubén y al Rega, para animarlos a que continúen.

Pepe.

A mis padres que han estado pendientes de mí en cada momento desde que nací, los quiero mucho y espero haber sido capaz de estar a la altura de todo el cariño que ellos me han dado.

En especial a mi abuela Lala, mi abuelo Lorenzo y mi tío chito que me gustaría mucho que pudieran disfrutar todos estos momentos, estoy seguro que sentirían mucho orgullo.

A mi familia, mi tía Mirta y mis abuelos Wilfredo y Mirta, los quiero mucho.

George.

AGRADECIMIENTOS

A mis padres y mi hermano, los que más quiero, por estar pendientes de mí en todo momento.

A mi novia Yurita, por quererme tanto, apoyarme y ayudarme durante todo este tiempo.

A todos mis maestros que desde la enseñanza primaria tuvieron fe en mí y me trataron como a un hijo, especialmente a mi abuela Mercedes, por enseñarme el camino del sacrificio y la virtud en el estudio.

Al profesor Luiver Duran por darme aliento e incentivarme a investigar.

A mis amigos del pre, en especial a Camilo, por compartir la distancia, y la alegría de nuestros reencuentros.

A José M. Borroto, Ricardo Sosa, Tomás Martínez y George Pérez, por formar el equipo tan fuerte y especial que hacemos.

A todos: muchas gracias.

Pepe.

A mis padres Jorge Alejandro y Virginia Inocencia, que son las dos personas que mas quiero en la vida, les agradezco el haberme llevado por el camino correcto y hacer de mi un hombre de bien.

A Mirta la tía que mas quiero y a mi tío Aldo, gracias a los dos.

A mi hermano Reimer que siempre me ha apoyado.

A mi pareja, Yanelis gracias por estar a mi lado en cada momento.

A mi amigo Daynier, gracias por confiar en mí.

A mis amigos Pepe, Ricardo, Borroto y tomas, siempre recordare los momentos que pasamos juntos.

A mis profesores pero en especial a Israel Ramos Sospedra, gracias Isra, por tu ayuda en mi formación.

George.

RESUMEN

Este trabajo presenta una propuesta arquitectónica para la automatización del Sistema Nacional Público para el Seguimiento de Inversiones y Sectores. Para la descripción de la propuesta se realizó un estudio del arte sobre las principales metodologías, herramientas y tecnologías de software con el objetivo de resolver un grupo de requisitos y necesidades.

El estudio está enmarcado en elementos básicos relativos a la arquitectura de software pudiendo, a partir de ellos, definir una propuesta arquitectónica que contribuya a garantizar el mantenimiento, eficiencia, seguridad, portabilidad y estabilidad del sistema.

Asimismo la proposición se documenta a partir de la generación del artefacto fundamental definido para el rol de arquitecto en la metodología RUP, teniendo en cuenta los puntos establecidos para el documento según la plantilla que se destina para tal efecto. En vistas a la validación de la propuesta se empleó el método de evaluación de arquitecturas Active Reviews for Intermediate Designs, que permite determinar el cumplimento de la arquitectura con los atributos de calidad establecidos a partir de escenarios.

PALABRAS CLAVES

SINAPSIS, Arquitectura de Software.

TABLA DE CONTENIDOS

ÍNDICE DE FIGURAS

INTRODUCCIÓN

En la República Bolivariana de Venezuela se lleva a cabo el presupuesto por proyectos por disposición de su Presidente, el Comandante Hugo Rafael Chávez Frías. A través del presupuesto el gobierno proporciona bienes y servicios para atender las necesidades de la comunidad; cancela servicios que permiten su funcionamiento, paga a proveedores y contratistas, resuelve problemas de pasivos laborales y contractuales con sus trabajadores.

Para llevar a cabo la gestión del presupuesto por proyectos, el Ministerio del Poder Popular para la Planificación y Desarrollo, conjuntamente con el Ministerio de Economía y Finanzas y la Vicepresidencia de la República, decidieron implantar un sistema informático que facilitara el registro, seguimiento y control de todos los proyectos que formarían parte del presupuesto anual de la nación, denominado Nueva Etapa.

La aplicación Nueva Etapa se diseñó para registrar la información concerniente a los datos generales de los proyectos así como la información presupuestaria para su control, seguimiento y evaluaciones posteriores; esto con la finalidad de proveer una herramienta informática para el reporte de planes, acciones y avances de los proyectos y/o medidas para obtener reportes actualizados del estado de las acciones y evaluar los posibles financiamientos especiales a ser aprobados por parte de la comisión presidencial.

El sistema Nueva Etapa no fue desarrollado utilizando una metodología de desarrollo de software, y no cuenta con la documentación necesaria que facilite la comprensión y optimización de dicho sistema. La arquitectura aplicada a este sistema no permite una fácil y rápida modificación de las funcionalidades del mismo. Los usuarios constantemente llaman al equipo de soporte para realizar alguna modificación en el sistema - modificación que debiera estar como una funcionalidad básica dentro del mismo. El sistema no fue pensado para que los usuarios ingresen los datos referentes a los proyectos y acciones centralizadas en todo momento, sino que cuentan con períodos de tiempos que se establecen de forma manual sobre el sistema; estos períodos de tiempo son muy cortos por lo que la información registrada en el mismo no es del todo confiable ya que los usuarios se ven presionados a cumplir con los plazos de tiempo establecidos. La información referente a la planificación de los proyectos puede ser modificada libremente sin tener en cuenta fechas o períodos de modificación alguno, lo que trae como consecuencia que la planificación histórica de un proyecto pueda ser modificada, lo que brinda reportes falsos del seguimiento que se le ha dado a los proyectos, afectando grandemente a la planificación de presupuestos.

Venezuela es un país revolucionario en constante transformación, debido a esto, las necesidades de la administración pública son cada vez más exigentes. El año siguiente a la implantación del sistema Nueva Etapa fue necesaria la realización de nuevos cambios referentes al manejo de la información de los proyectos.

Estos cambios realizados incorporaron un grupo de funcionalidades para la gestión de proyectos de algunos de los sectores de Venezuela que no fueron incluidos en la versión inicial. Los nuevos cambios realizados al sistema incluyeron modificación en la forma en que se ingresaban los datos de las fichas de los proyectos, donde se hizo notar la ausencia de una metodología y una correcta arquitectura de software: se duplicó innecesariamente tablas en la base de datos, aumentaron los errores provocados por consultas SQL no optimizadas, no se dio tratamiento a los nuevos tipos de errores provocados, los usuarios se ven agobiados por la desorganización de las interfaces del sistema, entre muchos otros efectos. A raíz de estos problemas se ha visto comprometida la consistencia de la información manejada por dicho sistema y ciertos requerimientos para poder llevar un control estricto de los recursos y los usuarios que interactúan con el mismo.

Dada la necesidad que representa un sistema de este tipo para la administración pública de Venezuela, los principales organismos rectores de dicho país deciden cambiar el actual software, dándole la tarea a Petróleos de Venezuela S.A. (PDVSA) de ser el organismo encargado de la búsqueda de un nuevo sistema para sustituir a Nueva Etapa. La tarea fue dada al departamento de Automatización, Informática y Telecomunicaciones de PDVSA (AIT PDVSA). Este último solicita los servicios de ALBET para el desarrollo de un sistema que sustituya al actual Nueva Etapa y que sea capaz de cubrir todas las necesidades y problemas que afronta el sector público para la gestión de los proyectos en Venezuela.

Para darle solución a esta problemática se decidió desarrollar el Sistema Nacional Público para el Seguimiento de Inversiones y Sectores (SINAPSIS). Un sistema informático que sustituiría completamente el anterior. Este nuevo sistema estará constituido por siete módulos: Registro y aprobación de proyectos, en el cual se recoge todo lo referente a la introducción de los datos y el proceso de aprobación de los fichas técnicas de los proyectos; Acciones centralizadas, en el cual se registra y aprueba lo referente a las acciones centralizadas de los organismos; Seguimiento y control, en este módulo es donde se lleva el control de la ejecución de todos los proyectos y acciones centralizadas; Migración de datos, este se encarga de migrar todos los datos existentes y necesarios desde el sistema Nueva Etapa hacia SINAPSIS; Reportes, el cual se encargaría de la realización de los reportes pertinentes en el sistema; Administración, en este módulo se maneja todo

lo referente a los usuarios, roles y permisos que serán asignados para interactuar con el sistema así como todo lo referente a la seguridad; y, por último, el módulo de Configuración, en el cual se administrarán todos los nomencladores que existan en la aplicación.

Para la selección de la plataforma bajo la cual se desarrollaría SINAPSIS se tuvo en cuenta el decreto 3.390 de la Gaceta Oficial No 38.095 de fecha 28/12/2004 establece que la Administración Pública Nacional de Venezuela empleará prioritariamente Software Libre con Estándares Abiertos en sus sistemas, proyectos y servicios informáticos. De ahí que la nueva solución fuese implementada utilizando Software Libre con Estándares Abiertos.

AIT PDVSA solicitó que la nueva solución fuese desarrollada utilizando el lenguaje de programación Java, a raíz de que algunos de los sistemas que emplea la Administración Pública para el manejo de presupuestos de la nación, como el Sistema Integrado de Gestión y Control de las Finanzas Públicas (SIGECOF) y el Sistema Presupuestario (SIPRE), están desarrollados también bajo esta tecnología. Mediante el SIGECOF se lleva el control de los gastos que tienen cada una de las entidades centralizadas de Venezuela; y mediante el SIGPRE se asignan los presupuestos anuales por concepto de gastos para cada una de las entidades.

El sistema SINAPSIS se estima sea un software de un gran impacto en la gestión presupuestaria de Venezuela. Para la construcción de este sistema se hace necesario establecer el punto de partida del desarrollo así como especificar que estructura deberá tener el mismo para sus posteriores etapas de implementación y despliegue. Es necesario especificar cuáles herramientas serán utilizadas y que estas complementen de una forma u otra la nueva aplicación.

Una arquitectura bien elaborada propicia robustez al sistema, además que sienta las bases para un buen desarrollo. También, ayuda en gran medida a minimizar el impacto que puede ocasionar algún cambio en las etapas posteriores al diseño e implementación del sistema, logrando que el software sea liberado en el menor tiempo posible, con la mayor calidad y que los clientes queden satisfechos con la solución hecha.

Por lo antes expuesto surge el siguiente **problema**: ¿qué arquitectura de software definir para el desarrollo de SINAPSIS, de forma que contribuya a garantizar el cumplimiento de los requerimientos del cliente?

El **objeto de estudio** lo constituye el proceso de desarrollo de software.

El **campo de acción** queda enmarcado en la arquitectura de software de SINAPSIS.

El **objetivo general** de este trabajo es definir una arquitectura de software dentro de la plataforma empresarial JEE para el desarrollo de SINAPSIS, de forma que contribuya a garantizar el cumplimiento de los requerimientos del cliente.

Para dar cumplimiento a este objetivo se plantean los siguientes **objetivos específicos:**

- Elaborar el marco teórico de la investigación.
- Definir una arquitectura de software bajo la plataforma JEE.
- Evaluar la arquitectura de software propuesta.

La **hipótesis** en la que está basada esta investigación es la siguiente: si se define una correcta arquitectura de software bajo la plataforma JEE para el desarrollo de SINAPSIS, entonces se contribuirá a garantizar el cumplimiento de los requerimientos del cliente.

La estrategia de investigación a seguir es la exploratoria, pues se realizará un estudio acerca de las diferentes metodologías para el desarrollo de software, así como los principales elementos dentro de la plataforma Java, y otras herramientas a seleccionar, para la confección de una arquitectura de software. Esto con el fin de adquirir los conocimientos necesarios para darle solución de la mejor forma posible al problema planteado.

Los métodos científicos de investigación a utilizar son el teórico y el empírico. Dentro de los métodos teóricos se utilizará el histórico, al realizar un estudio del estado del arte sobre todo lo relacionado con las arquitecturas de software. El hipotético deductivo es otro método a poner en práctica, pues a partir del problema se plantearon objetivos específicos para darles solución a lo largo de la investigación. Para la descripción de la arquitectura del sistema es necesaria la elaboración de diagramas, figuras y otros artefactos importantes, por lo que se hará uso del método de modelación, pues mediante este se pueden crear abstracciones con el propósito de explicar la realidad. Dentro de los métodos empíricos otro a seguir es el de la observación, pues se va a definir dentro de la Ingeniería de Software la parte que específicamente se tratará durante la investigación.

A partir del estudio se espera obtener el modelo correspondiente a la descripción de la arquitectura de software para SINAPSIS, de forma que constituya la línea base de la arquitectura y siente las bases para del ciclo de desarrollo del software funcional.

La presente tesis está estructurada de la siguiente forma:

En el **Capítulo 1** se realiza un estudio sobre algunas de las metodologías para el desarrollo de software más usadas en la actualidad. Se hace un análisis sobre los tipos de arquitecturas para la selección de cuál llevará el sistema. Se estudian además los principales elementos a tener en cuenta a la hora de elaborar una arquitectura: frameworks, sistemas gestores de base de datos, entornos de ejecución y herramientas de desarrollo.

En el **Capítulo 2** se muestra el modelo de arquitectura propuesto para la construcción del sistema, derivada de la actividad del arquitecto de software. Se expone como estará constituido el sistema, cuáles serán los frameworks a utilizar, el sistema gestor de base de datos y sobre cual entorno de ejecución estará la aplicación. Además se presenta como estará estructurado el espacio de trabajo del equipo de desarrollo basándose en patrones y convenciones de códigos. Por último, se brindarán además otros elementos como los referentes a la seguridad del sistema y la comunicación con sistemas legados.

En el **Capítulo 3** se analizan los resultados obtenidos con la propuesta de arquitectura. Se evalúa la arquitectura del sistema mediante el uso de un método de evaluación.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

A medida que el tamaño y la complejidad de los sistemas de software se incrementan, los problemas de diseño van más allá de los algoritmos y las estructuras de datos: el diseño y especificación de la estructura global del sistema emerge como un nuevo tipo de problema. Esta estructura global incluye la organización total y la estructura de control del sistema; protocolos de comunicación, sincronización, y acceso a datos; asignación de las funcionalidades a elementos del diseño; distribución física; composición de elementos de diseño; escalabilidad y rendimiento; y selección entre las alternativas de diseño. De todo lo anteriormente mencionado se encarga la arquitectura de software, y el diseño de la misma se ha hecho prácticamente imprescindible dentro del proceso de desarrollo de un software.

En este capítulo se hace un análisis de los principales aspectos a tener en cuenta a la hora de elaborar la arquitectura de un sistema de software, primeramente se hace un estudio de las metodologías de desarrollo, que es en donde se traza la organización del equipo de trabajo que tomará parte en la elaboración del software; luego se verán aspectos que deben ser analizados a la hora de elaborar la arquitectura para un software como son su definición, tipos de arquitectura y la selección de cuales tecnologías y/o herramientas utilizar.

1.1. Metodologías de desarrollo

Todo desarrollo de software es riesgoso y difícil de controlar, pero si no se aplica una metodología por medio, lo que se obtiene es clientes insatisfechos con el resultado y desarrolladores aún más insatisfechos. Sin embargo, muchas veces no se toma en cuenta el utilizar una metodología adecuada, sobre todo cuando se trata de proyectos pequeños de dos o tres meses. Cuando los proyectos que se van a desarrollar son de mayor envergadura, sí toma sentido basarse en una metodología de desarrollo, y se comienza a buscar la que más se adecúe al proceso que se quiera establecer. La mayoría de las veces no se encuentra la más adecuada y se termina por adaptar una metodología existente o hacer una propia, lo que para nada es incorrecto, siempre y cuando se cumpla con el objetivo.

Las metodologías de desarrollo de software se han categorizado en dos grandes grupos: las ágiles y las pesadas. Entre las metodologías ágiles están XP (Extreme Programming), FDD (Feature Driven Development), DSDM (Dynamic Systems Development Method), AUP (Agile Unified Process), Scrum, Crystal, Adaptive Software Development y otras. Así mismo se puede mencionar RUP (Rational Unified Process) como metodología pesada. (1) (2) (3)

A continuación se abordarán características fundamentales de dos de las metodologías más usadas actualmente; por una parte, XP como principal exponente de las metodologías ágiles, y por el otro, RUP como una de las más generalizadas y usadas entre las metodologías pesadas.

XP (Extreme Programming)

XP es una de las metodologías ágiles para el desarrollo de software más exitosas de la actualidad. Se utiliza en proyectos con pequeños equipos de desarrollo y con corto plazo de entrega. Se basa en la retroalimentación entre el cliente y el equipo de desarrollo, buena comunicación entre todos los participantes y simplicidad en las soluciones implementadas. Consiste en una programación rápida, cuya particularidad es que tiene como miembro del equipo al usuario final. Es adecuada para proyectos con requisitos imprecisos y muy cambiantes, y donde existe un alto riesgo técnico. (4)

En XP se sigue la idea de la programación en pares, dado las ventajas que ofrece respecto a la creación del código, pues se pueden evitar errores y malos diseños al controlar cada línea de código y decisión de diseño instantáneamente. La interacción entre ambos desarrolladores puede generar discusiones que lleven a mejores estructuras y algoritmos, aumentando la calidad del software. (5) (6)

El ciclo de vida ideal de XP está compuesto por seis fases: Exploración, Planificación de la entrega, Iteraciones, Producción, Mantenimiento y Muerte del proyecto. (1) (7) (8)

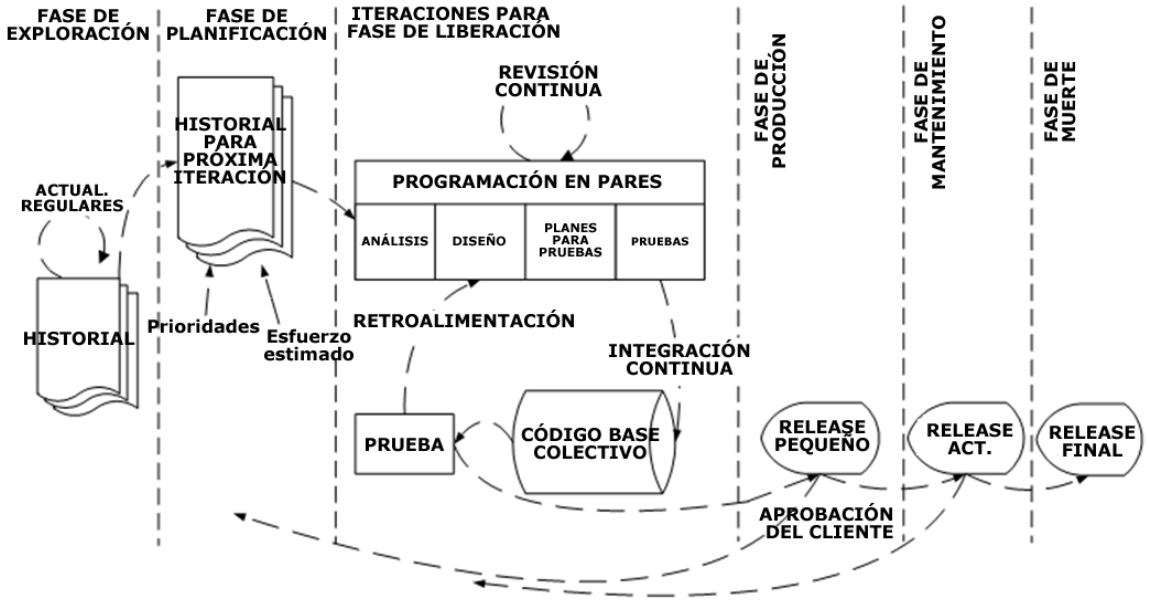

Figura 1.1: Ciclo de vida del proceso XP (1).

RUP (Rational Unified Process)

RUP es una metodología para el desarrollo de software que es el resultado de varios años de trabajo y uso práctico, en el que se han unificado varias técnicas partiendo del Lenguaje Unificado de Modelado (UML). Al igual que cualquier notación, el Proceso Unificado actúa como un modelo que puede adaptarse a cualquier tipo de proyecto y empresa, ya sean grandes o pequeñas. Es un marco de trabajo genérico que puede especializarse para una gran variedad de sistemas de software, para diferentes áreas de aplicación, diferentes tipos de organizaciones, diferentes niveles de aptitud y diferentes tamaños de proyectos. (2)

RUP define como sus principales elementos: (2) (10)

- Trabajadores (“quién”): define habilidades y responsabilidades (rol) de un individuo o grupo de individuos, sistema automatizado o máquina, que trabajan en conjunto como un equipo. Ellos realizan las actividades y son propietarios de elementos.
- Actividades (“cómo”): es una tarea que tiene un propósito claro, es realizada por un trabajador y manipula elementos.
- Artefactos (”qué”): productos tangibles del proyecto que son creados, modificados y usados por los trabajadores al realizar actividades.
- Flujo de actividades (“cuándo”): secuencia de actividades realizadas por trabajadores y que produce un resultado de valor observable.

RUP divide el proceso de desarrollo en una serie de ciclos que constituyen la vida de un sistema. El ciclo de vida de RUP se caracteriza por ser: Dirigido por casos de uso, Centrado en la arquitectura, e Iterativo e incremental. (2) (10) (11)

Cada ciclo está compuesto por cuatro fases, y dentro de cada una, el trabajo se puede descomponer en iteraciones. Cada fase finaliza con un hito y cada uno de estos se determina por la disponibilidad de un conjunto de artefactos. Los hitos tienen muchos objetivos, entre ellos se encuentran: la toma de decisiones por parte de los directivos antes de que el trabajo pueda continuar en la siguiente fase, y además, permiten controlar el progreso del trabajo a la dirección del proyecto y a los mismos desarrolladores. (2) (11)

La Figura 1.2 muestra una representación gráfica de los flujos de trabajo y las fases de RUP, mostrando la dinámica expresada en iteraciones y puntos de control. Las fases definidas son: Inicio, Elaboración, Construcción y Transición. (2) (10) (11) (12)

Para cada una de las fases se debe trabajar en 9 disciplinas o flujos de trabajo. Los 6 primeros son conocidos como flujos de ingeniería y los tres últimos como flujos de apoyo. Estos flujos de trabajos son: Modelamiento del negocio, Requerimientos, Análisis y diseño, Implementación,

Prueba, Despliegue, Administración de configuración y cambios, Administración del proyecto y Ambiente. (10) (12)

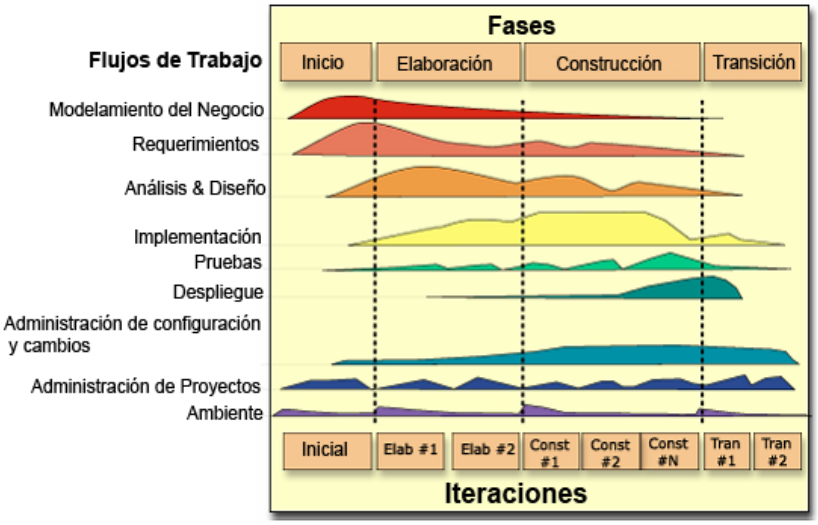

Figura 1.2: RUP en Dos Dimensiones (2).

1.1.1. Selección de la metodología de desarrollo de software

Luego de realizar un estudio de las metodologías más utilizadas, se puede llegar a la conclusión de que RUP es la más indicada para guiar el desarrollo del sistema que se desea implementar. RUP constituye uno de los estándares internacionales que más aceptación ha tenido. Además varias herramientas CASE soportan dicha metodología, permitiendo el trabajo en equipo y con la capacidad de generar código en distintos lenguajes de programación a partir de un diseño UML.

A parte de esto, RUP es una metodología que se encarga de: (10)

- Asegurar la producción de un software de alta calidad que reúna las necesidades de los usuarios finales dentro de un plan y un presupuesto predecible.
- Proveer un enfoque disciplinado para asignar tareas y responsabilidades dentro del desarrollo del sistema.
- Proveer un camino metódico, sistemático para desarrollar, diseñar y validar una arquitectura.
- Reducir en gran medida el riesgo que representa la construcción de sistemas complejos, porque evoluciona de forma incremental partiendo de sistemas más pequeños.

1.2. Arquitectura de software

La definición de arquitectura de software varía de un autor a otro; entre estas definiciones se puede encontrar la especificada por la IEEE 1471 – 2000:

La Arquitectura de Software es la organización fundamental de un sistema encarnada en sus componentes, las relaciones entre ellos y el ambiente y los principios que orientan su diseño y evolución.

Pressman, en su libro, hace una definición de la arquitectura:

La arquitectura de software de un sistema de programa o computación es la estructura de las estructuras del sistema, la cual comprende los componentes del software, las propiedades de esos componentes visibles externamente, y las relaciones entre ellos. (13) (15) La arquitectura no es el software operacional. Más bien, es la representación que capacita al ingeniero del software para:

- Analizar la efectividad del diseño para la consecución de los requisitos fijados.
- Considerar las alternativas arquitectónicas en una etapa en la cual hacer cambios en el diseño es relativamente fácil.
- Reducir los riesgos asociados a la construcción del software.

Otra definición no menos acertada está la propuesta por RUP: (12)

La arquitectura de software abarca las decisiones significativas sobre la organización de un sistema de software, la selección de los elementos estructurales y de sus interfaces por los cuales el sistema es compuesto, junto con su comportamiento según lo especificado en las colaboraciones entre estos elementos, la composición de estos elementos estructurales y del comportamiento en subsistemas progresivamente más grandes y el estilo arquitectónico que dirigen esta organización, los elementos y sus interfaces, sus colaboraciones y su composición. La arquitectura de software no tiene que ver solamente con la estructura y el comportamiento del sistema, sino que también se refiere a la usabilidad, funcionalidad, rendimiento, capacidad de recuperación, la reutilización, la comprensión, limitaciones tecnología y económicas, y la estética.

Pero, ¿qué importancia tiene la arquitectura dentro del proceso de desarrollo de software?

La arquitectura de software es unos de los pilares más importantes en el desarrollo de un sistema. Esta representa las decisiones de diseño más tempranas, es lo más difícil de tener correcto: se necesita tiempo y experiencia para poder llegar a establecer una arquitectura que cumpla con todos los requerimientos, tanto funcionales como no funcionales, y que además sea escalable y sea propensa a la reutilización; es lo más difícil de cambiar: una vez que está establecido y en desarrollo un modelo de arquitectura para un sistema se tornan complejas las decisiones sobre el cambio de un elemento o componente de la arquitectura, ya que, generalmente,

los componentes tienen dependencias entre sí; la arquitectura de software representa el vehículo de comunicación entre todos los interesado, los que van desde los clientes, stakeholders, desarrolladores, probadores. La arquitectura es el primer artefacto que se obtiene del diseño y el principal responsable del rendimiento, modificabilidad, confiabilidad y seguridad del sistema; es el elemento esencial para la reutilización sistemática, es quien define el grado de lo transferible, abstracción y de reutilización de un sistema. (16)

Todo sistema de software tiene una arquitectura. Esto es fácilmente comprobable, pues estos se componen de componentes y relaciones entre estos. Una buena arquitectura sienta las bases para un sistema exitoso. Una mala arquitectura generalmente se convierte en resultados no favorables, teniendo luego que invertir tiempo y recursos innecesarios e imprevistos para continuar con el desarrollo del sistema.

1.2.1. Arquitectura según la metodología seleccionada

Anteriormente se vio la definición que propone RUP para arquitectura de software; además de esto, RUP especifica la forma en que se debe describir la arquitectura, que es mediante vistas o modelos que encierran la información necesaria para establecer la línea base a partir de la cual se desarrollará el sistema.

La arquitectura se representa a través de 4 + 1 vistas, dicho de modo que se entienda que 4 de estas vistas están regidas por una rectora. La rectora se considera la vista de casos de uso, y las restantes son la vista lógica, la vista de procesos, la vista de despliegue y la vista de implementación. (16)

Vista de casos de uso: Esta vista representa un subconjunto del artefacto Modelo de casos de uso y lista los casos de usos o escenarios del modelo de casos de uso más significativos, con las funcionalidades centrales del sistema.

Vista lógica: Esta vista representa un subconjunto del artefacto Modelo de diseño, la cual representa los elementos de diseño más importantes para la arquitectura del sistema. Este describe las clases más importantes, su organización en paquetes y subsistemas.

Vista de procesos: Esta vista suministra una base para la comprensión de la organización de los procesos de un sistema, ilustrados en el mapeo de las clases y subsistemas en procesos e hilos. Sólo suele usarse cuando el sistema presenta procesos concurrentes o hilos.

Vista de despliegue: Esta vista suministra una base para la comprensión de la distribución física de un sistema a través de nodos. Suele utilizarse cuando el sistema está distribuido. Existe una traza directa del modelo de implementación, puesto que cada componente físico debe estar almacenado

en un nodo, esto incluye también la asignación de tareas provenientes de la vista de procesos en los nodos.

Vista de implementación: Esta vista describe la descomposición del software en capas y subsistemas de implementación. También provee una vista de la trazabilidad de los elementos de diseño de la vista lógica para la implementación.

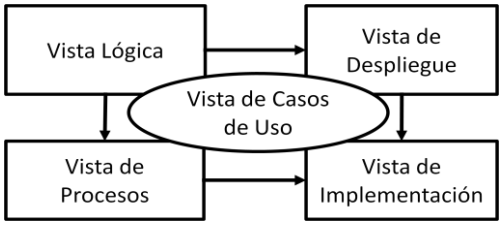

Figura 1.3 Las "4+1" vistas de la arquitectura. (21)

1.2.2. Tipos de aplicaciones de acuerdo a la arquitectura

La elaboración de un software generalmente no requiere la elaboración de una nueva arquitectura. Lo más común es utilizar una arquitectura conocida teniendo en cuenta las ventajas que esta tiene de acuerdo al tipo de aplicación que se quiere desarrollar y, además, valorando las desventajas y las formas de disminuirlas. Entre las más comunes están: (17)

Aplicaciones de arquitectura Monolítica

El software se estructura en grupos funcionales muy acoplados, involucrando los aspectos referidos a la presentación, procesamiento y almacenamiento de la información. En este conjunto están considerados las distintas aplicaciones para escritorio: sistemas operativos, ofimática, juegos monousuario, etc.

Figura 1.4 Arquitectura Monolítica.

Desventajas de la arquitectura monolítica

Todas las tareas se realizan en una sola terminal, con la cual el usuario establece una comunicación simple donde no hay distribución de tareas entre sí, y en donde el cliente y el servidor no pueden cambiar de roles. En este tipo de arquitectura no hay concurrencia de usuarios. La

escalabilidad e integración de estos sistemas están muy comprometidas debido al alto acoplamiento entre las capas lógicas de la aplicación.

Aplicaciones de arquitectura Cliente–Servidor

Son aquellas donde el software reparte su carga de cómputo en dos partes independientes, una parte denominada servidor y otra parte denominada clientes, pero sin establecer una delimitación clara de funciones. El servidor alberga toda la información estática, y todos los clientes se conectan al servidor para consultar y modificar dicha información. De este modo la información es única, está siempre compartida para todos y siempre todos los clientes ven la misma información. Además, de esta manera la concurrencia de accesos al servidor central se puede controlar totalmente y no existe ningún riesgo de solapar ni perder información en el servidor. Esta sería la mejor manera y la más sencilla y eficiente de plantear un sistema distribuido.

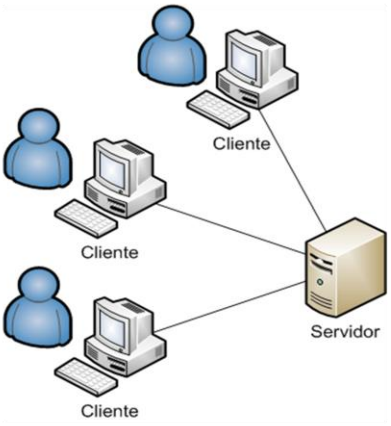

Figura 1.5 Arquitectura Cliente/Servidor.

Desventajas de la arquitectura Cliente-Servidor

La no clara especificación de los roles entre las partes cliente y servidor dificulta la escalabilidad de los sistemas. La facilidad y coste de mantenimiento dependen mucho de la forma en que estén distribuidas las funcionalidades entre los distintos componentes de la aplicación. Este tipo de arquitectura genera un alto intercambio de datos a través de la red, lo que puede, cuando una cantidad considerable de clientes envían peticiones simultáneas al servidor, ocasionar congestiones de tráfico. La centralización de la información o funcionalidades en un solo servidor puede ser un factor negativo: si el servidor no se encuentra disponible en un momento determinado las peticiones de los clientes no pueden ser satisfechas dejando el sistema inutilizable.

Aplicaciones de arquitectura Peer to Peer

En este tipo de arquitectura el software se encuentra distribuido por equipos conectados a un grupo de trabajo que comparten datos sin un servidor central establecido. Estos programas utilizan la metodología de grupos de trabajo. Los equipos conectados a un grupo de trabajo comparten la información entre ellos y todos los equipos del grupo contienen la misma información, pero localmente, en cada uno de los equipos.

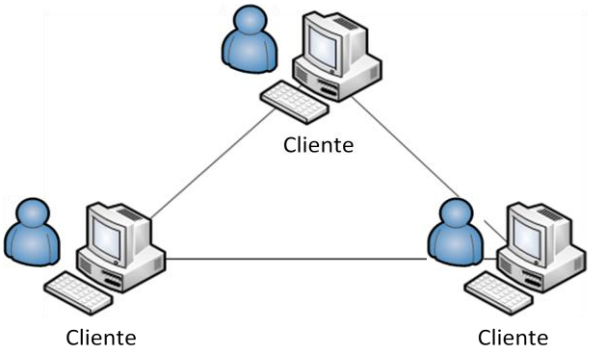

Figura 1.6 Arquitectura Peer to Peer.

Desventajas de la arquitectura Peer to Peer

El principal problema de este método es la actualización de dicha información en cada equipo. El sistema no asegura que la información contenida en cada equipo sea la última versión del grupo de trabajo. Esto es por la metodología que se utiliza en esta técnica de distribución. Cuando uno de los equipos pertenecientes al grupo de trabajo realiza un cambio en la información, este cambio se transmite a todos los usuarios del grupo que estén conectados en ese momento, y estos usuarios ya tienen en su equipo local dicho cambio. Si el equipo que ha hecho el cambio se desconecta y a todos los que les ha pasado los cambios también, y entra un equipo que no ha visto estos cambios aun, no tiene a nadie que se los pase, y por lo tanto, no sabe que existen en ese momento.

Aplicaciones de arquitectura de 3 Capas Físicas o Niveles

Es una generalización de la arquitectura cliente-servidor donde la carga se divide en tres partes con un reparto claro de funciones: una capa para la presentación, otra para el cálculo y procesamiento de la información, y otra para el almacenamiento de datos. Una capa solamente tiene relación con la siguiente. De esta forma:

- La capa de presentación es aquella con la que interactúa el usuario y generalmente brinda interfaces de entrada y salida de información.

- La capa de negocios es aquella que se encarga de los cálculos y/o procesamiento de la información; involucrando estas tareas las labores de validación, clasificación, operaciones matemáticas, etc.
- La capa de datos es aquella que se encarga del envío/obtención de datos de las fuentes de datos; las que pueden ser por ejemplo: bases de datos, archivos planos, otros sistemas a través de interfaces CORBA, RMI, etc. Es aquella que se encarga del control de la persistencia de la información.

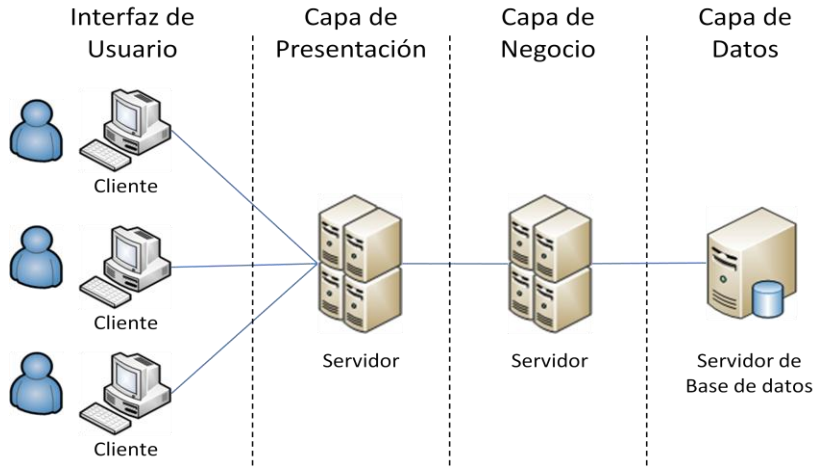

Figura 1.7 Arquitectura de 3 Capas más utilizada.

Desventajas de la arquitectura de 3 Capas

La principal desventaja que tiene este tipo de arquitectura es respecto al rendimiento, ya que al estar los recursos distribuidos en varias partes, la comunicación dentro de los procesos es más lenta respecto a una aplicación en donde la comunicación sea dentro de una misma terminal.

Aplicaciones de arquitectura de "N" Capas Físicas o Niveles

Generalización de la arquitectura cliente-servidor donde la carga se divide en un conjunto de "N" partes con un reparto claro de funciones: unas capas serán designadas para la presentación, otras para el cálculo y procesamiento de la información y otras para el almacenamiento; pero se mantiene el estándar de una capa determinada solamente tiene relación con la siguiente.

El denominado "Web 2.0" es considerado una puesta en práctica de esta arquitectura.

Aplicaciones de arquitectura Orientada a Servicios (SOA)

Las aplicaciones están formadas por servicios de aplicación débilmente acoplados y altamente interoperables. Para comunicarse entre sí, estos servicios se basan en una definición formal

independiente de la plataforma subyacente y del lenguaje de programación. SOA es una arquitectura para la construcción de aplicaciones de negocio como un conjunto de componentes débilmente acoplados orquestados para entregar un nivel de servicios bien definidos que vinculan los procesos de negocio. (18)

A continuación se mencionan las principales características de SOA que ayudan a esclarecer el concepto de “orientación a servicios”: (18)

- SOA es para la construcción de aplicaciones empresariales.
- SOA es una arquitectura basada en componentes.
- Los componentes de SOA son débilmente acoplados.
- Los componentes de SOA son orquestados a través de procesos de negocio para ofrecer un nivel de servicio bien definido.

1.3. La plataforma empresarial de Java

JEE es una plataforma independiente, centrada en entorno Java para desarrollar, construir y desplegar aplicaciones empresariales basadas en la Web. Incluye muchos de los componentes de JSE. Simplifica el desarrollo de aplicaciones y disminuye la necesidad de programar y de entrenar al programador mediante la creación estandarizada, componentes modulares reutilizables y brindando el nivel para manejar muchos aspectos de la programación automáticamente. (19)

Los desarrolladores de aplicaciones empresariales necesitan JEE porque escribir aplicaciones empresariales distribuidas no es fácil, y se necesita una solución de alta productividad que les permite centrarse sólo en la escritura de su lógica de negocio y con una gama completa de servicios de clase empresarial en la cual apoyarse, como objetos de transacción distribuida, middleware orientado a mensajes, y de nombres y servicios de directorio. (19)

JEE provee un conjunto de librerías, interfaces, frameworks que brindan una infraestructura para el desarrollo de arquitecturas empresariales, abarcando cada uno de los elementos como las transacciones, la mensajería, las conexiones a base de datos, interfaces de usuario, manejo de recursos, seguridad, entre otros. Además JEE es una plataforma con experiencia y amplio soporte en el mundo. Dentro de las principales empresas en el mundo que desarrollan para la plataforma se encuentran Sun, IBM, Oracle, Apache, BEA, JBOSS, entre otras; las cuales aportan gran cantidad de componentes y herramientas que apoyan y facilitan el trabajo de los desarrolladores en los proyectos. (20)

1.3.1. APIs JEE

La plataforma JEE define un grupo de extensiones estándares de Java que son la base para el desarrollo de arquitecturas empresariales, las APIs. A continuación se exponen algunas de ellas:

Enterprise JavaBeans (EJB)

Los Enterprise JavaBeans proporcionan un modelo de componentes distribuido estándar para el lado del servidor. Un componente de EJB, o Enterprise Bean, es un cuerpo de código que contiene los campos y métodos para implementar módulos de lógica de negocio. El hecho de estar basado en componentes permite que estos sean flexibles y sobre todo reutilizables. Un EJB es ejecutado dentro de un contenedor EJB, y se benefician de los servicios que aporta este último como son seguridad, comunicación, manejo de transacciones, entre otros. Se puede pensar en un Enterprise Bean como un elemento que se puede utilizar solo o con otro Enterprise Bean para la ejecución de la lógica de negocio en el servidor Java. (24)

Java Servlets

La tecnología Java Servlet permite definir las clases específicas de servlet. Una clase servlet extiende las capacidades de los servidores que albergan las aplicaciones que se acceden por medio de una solicitud de modelos de programación de petición/respuesta. Aunque los servlets pueden responder a cualquier tipo de solicitud, son comúnmente utilizados para ampliar las aplicaciones alojadas en servidores web. (24)

Los servlets se ejecutan dentro de un contenedor web y proporcionan la comunicación entre el cliente y el servidor web. Pueden comunicarse entre sí, por tanto es posible la reasignación dinámica de procesos. En grandes aplicaciones el desarrollo con servlets puede ser agotador por la gran cantidad de tipos de solicitudes que se pueden generar en ellas y por tanto las complejas reglas de navegación que se generan. (20)

Java Server Pages (JSP)

Permiten poner fragmentos de código servlet directamente en un documento basado en texto. Una página JSP es un documento basado en texto que contiene dos tipos de texto: los datos (que puede expresarse en cualquier formato basado en texto, tales como HTML, WML y XML) y elementos JSP que determinan la forma en que la página construye el contenido dinámico. (24)

Simplifica la creación de páginas web dinámicas proporcionando un sistema de composición más conveniente que los servlets. Las páginas JSP se ejecutan en un servidor web y combinan anotación

estática, como HTML y XML, con las librerías de etiquetas y scriptlets. JSP ha sido un método sólido y de uso muy extendido para la generación de contenido web. Los servlets son adecuados para decidir cómo manejar solicitudes de cliente e invocar otros objetos del lado del servidor pero no son tan adecuados para generar contenidos. (20)

Java Server Pages Standard Library (JSTL)

La Librería Estándar de JSP encapsula funcionalidades básicas comunes a muchas aplicaciones JSP. En lugar de mezclar las etiquetas de los proveedores en sus numerosas aplicaciones JSP, se emplea un único estándar de etiquetas. Esta estandarización permite desplegar las aplicaciones en cualquier contenedor JSP que soporte JSTL y hace más probable que la implementación de las etiquetas sea de forma optimizada. (24)

Java Data Base Connection (JDBC)

El API JDBC permite la invocación de comandos SQL desde métodos de lenguaje de programación Java. Se puede utilizar las API JDBC en un Enterprise Bean cuando hay acceso a la base de datos mediante sesiones de Enterprise Bean. También se puede usar el API JDBC desde un servlet o una página JSP para acceder a la base de datos directamente sin pasar por una capa de lógica de negocios. (24)

Web Services

Son componentes de aplicación accesibles a través de protocolos Web estándar. Son unidades de lógica de aplicación. Proporciona servicios y datos a clientes remotos y otras aplicaciones. Los clientes y las aplicaciones remotas acceden a servicios Web a través de protocolos de Internet omnipresentes. Utilizan XML para el transporte de datos y SOAP para hacer uso de servicios. Debido al uso de XML y SOAP, el acceso al servicio es independiente de la implementación. Así, un Web Service es como una arquitectura de componentes para la Web. (20)

1.3.2. Patrones de JEE

Los patrones de la plataforma empresarial de Java describen típicos problemas encontrados por desarrolladores de aplicaciones empresariales y proveen soluciones para los mismos. Estos patrones, a diferencia de los clásicos, están enfocados a los API JEE.

Los patrones de JEE se agrupan en tres grupos: (20)

- Patrones de presentación: todo lo necesario para presentar los datos de la aplicación y los elementos de la interfaz de usuario está dentro del grupo de presentaciones de la aplicación. Las tecnologías fundamentales en uso son JSP y los Java Servlet.
- Patrones de negocio: el grupo de negocio es donde tiene lugar todo el proceso de negocio. Las principales tecnologías JEE de este grupo son los EJB.
- Patrones de integración: el grupo de integración proporciona conexiones al grupo de recursos. El grupo de recursos incluye elementos como las colas de mensajes, bases de datos y sistemas heredados. Las tecnologías JEE más destacadas son JMS y JDBC.

A continuación se describen los más utilizados comúnmente.

Patrones de presentación

Front Controller (25)

Contexto: Los mecanismos de manejo de peticiones de la capa de presentación deben controlar y coordinar el procesamiento de todos los usuarios a través de varias peticiones. Dichos mecanismos de control se pueden manejar de una forma centralizada o descentralizada.

Problema: El sistema requiere un punto de acceso centralizado para que el manejo de peticiones de la capa de presentación soporte la integración de los servicios del sistema, recuperación de contenidos, control de vistas, y navegación. El control distribuido es más difícil de mantener, ya que los cambios se tienen que realizar en numerosos lugares.

Solución: Usar un controlador como el punto inicial de contacto para manejar las peticiones. El controlador proporciona un punto de entrada centralizado que controla y maneja las peticiones Web. Centralizando los puntos de decisión y control, el controlador también ayuda a reducir la cantidad de código Java, llamadas a scriptlets, embebidos en la página Java Server Pages (JSP).

Consecuencias: Centralización del control, mejora la manejabilidad de la seguridad y mejora la reusabilidad.

View Helper (25)

Contexto: El sistema crea el contenido de la presentación, lo que requiere el procesamiento de datos de negocio dinámicos.

Problema: Los cambios en la capa de presentación ocurren muy frecuentemente y son difíciles de desarrollar y mantener cuando la lógica de acceso a los datos de negocio y la lógica del formateo de la presentación se mezclan. Esto hace el sistema menos flexible, menos reutilizable, y generalmente menos adaptable a los cambios.

Solución: Una vista contiene código de formateo, delegando sus responsabilidades de procesamiento en sus clases de ayuda, implementadas como JavaBeans o etiquetas personalizadas. Las clases de ayuda o *helpers* también almacenan el modelo de datos intermedio de la vista y sirven como adaptadores de datos de negocio.

Consecuencias: Mejora el particionamiento de la aplicación, la reutilización y el mantenimiento, mejora la separación de roles.

Composite View (25)

*Contexto***:** Las páginas Web sofisticadas presentan contenido de varias fuentes de datos, utilizando varias subvistas que completan una sola página. Además, varios individuos con diferentes habilidades contribuyen al desarrollo y mantenimiento de esas páginas Web.

Problema: En lugar de proporcionar un mecanismo para combinar modularmente, las páginas se construyen embebiendo código de formateo directamente dentro de cada vista. La modificación de la distribución de múltiples vistas es difícil y propensa a errores, debido a la duplicación de código.

Solución: Utilizar vistas compuestas que se componen de varias subvistas atómicas. Cada componente de la plantilla se podría incluir dinámicamente dentro del total y la distribución de la página se maneja independientemente del contenido.

Consecuencias: Mejora la modularidad, la reutilización, la flexibilidad, el mantenimiento y la manejabilidad, impacto en el rendimiento.

Dispatcher View (25)

Contexto: El sistema controla el flujo de ejecución y accede al proceso de presentación, que es el responsable de generar el contenido dinámico.

Problema: No hay un componente centralizado para manejar el control de acceso, la recuperación de contenido, o el manejo de la vista, y hay código de control duplicado esparcido por varias vistas. Además, la lógica de negocio y la de formateo de la presentación están mezcladas en esas vistas, haciendo que el sistema sea menos flexible, menos reutilizable y generalmente menos resistente a los cambios. Mezclar lógica de negocio con el procesamiento de la vista también reduce la modularidad y proporciona una pobre separación de roles entre los equipos de producción Web y de desarrollo de software.

Solución: Combinar un controlador y un dispatcher con vistas y helpers (ver Front Controller y View Helper) para manejar peticiones de clientes y preparar una presentación dinámica como respuesta. Los controladores delegan la recuperación de contenido en los helpers, que manejan el relleno del modelo intermedio para la vista. Un dispatcher es el responsable del control de la vista y la navegación, y puede encapsularse dentro de un controlador o de un componente separado.

Consecuencias: Centraliza el control y mejora la modularidad, la reutilización, el particionamiento de la aplicación y la separación de roles.

Patrones de negocio

Busisness Delegate (26)

Contexto: Un sistema multicapa distribuido requiere invocación remota de métodos para enviar y recibir datos entre las capas. Los clientes están expuestos a la complejidad de tratar con componentes distribuidos.

Problema: Los componentes de la capa de presentación interactúan directamente con servicios de negocio. Esta interacción directa expone los detalles de la implementación del API del servicio de negocio a la capa de presentación. Como resultado, los componentes de la capa de presentación son vulnerables a los cambios en la implementación de los servicios de negocio: cuando cambia la implementación del servicio de negocio, la implementación del código expuesto en la capa de presentación también debe cambiar.

Solución: Utilizar un Business Delegate para reducir el acoplamiento entre los clientes de la capa de presentación y los servicios de negocio. Este oculta los detalles de la implementación del servicio de negocio, como los detalles de búsqueda y acceso de la arquitectura EJB.

Consecuencias: Reduce el acoplamiento, mejora la manejabilidad, traduce las excepciones del servicio de negocio, implementa recuperación de fallos y sincronización de threads, expone un interface simple y uniforme a la capa de negocio, impacto en el rendimiento y presenta una capa adicional.

Service Locator (26)

Contexto: La búsqueda y creación de servicios implican interfaces complejos y operaciones de red.

Problema: Los clientes JEE interactúan con componentes de servicio, como componentes Enterprise JavaBeans y Java Message Service (JMS), que proporcionan servicios de negocio y capacidades de persistencia. Para interactuar con estos componentes, los clientes deben o localizar el componente de servicio o crear un nuevo componente. Localizar un objeto servicio administrado JNDI es una tarea común para todos los clientes que necesiten acceder al objeto de servicio, por lo que el código JNDI aparece varias veces en esos clientes.

Solución: Utilizar un objeto Service Locator para abstraer toda la utilización JNDI y para ocultar las complejidades de la creación del contexto inicial, de búsqueda de objetos home EJB y de volver a crear objetos EJB. Varios clientes pueden reutilizar el objeto Service Locator para reducir la

complejidad del código, proporcionando un punto de control, y mejorando el rendimiento proporcionando facilidades de caché.

Consecuencias: Abstrae la complejidad, proporciona a los clientes un acceso uniforme a los servicios, facilita la adicción de nuevos componentes de negocio, mejora el rendimiento de la red y del cliente mediante el caché.

Session Facade (26)

Contexto: Los Enterprise Beans encapsulan lógica y datos de negocio y exponen sus interfaces, y con ellos la complejidad de los servicios distribuidos, a la capa de cliente.

Problema: Acoplamiento fuerte, que provoca la dependencia directa de los clientes a la implementación de los objetos de negocio, los clientes de la aplicación necesitan acceder a los objetos de negocio para cumplir sus responsabilidades y los requerimientos del usuario. El cliente se vuelve muy susceptible a los cambios en la capa de objetos de negocio. Se produce una degradación del rendimiento de la red porque el gran volumen de llamadas a métodos remotos incrementa la cantidad de interacciones sobre la capa de red. Como los objetos de negocio se exponen directamente a los clientes, no hay estrategia unificada para acceder a ellos.

Solución: Usar un bean de sesión como una fachada para encapsular la complejidad de las interacciones entre los objetos de negocio participantes en un flujo de trabajo. El Session Facade maneja los objetos de negocio y proporciona un servicio de acceso uniforme a los clientes.

Consecuencias: Introduce la capa controladora para la capa de negocio, expone un interface uniforme, reduce el acoplamiento, incrementa la manejabilidad, mejora el rendimiento, reduce los métodos específicos, proporciona acceso genérico, centraliza el control de seguridad y el de las transacciones, expone menos interfaces remotos a los clientes.

Transfer Object (26)

Contexto: Las aplicaciones cliente necesitan intercambiar datos con Enterprise Beans.

Problema: Toda llamada a método hecha al objeto de servicio de negocio potencialmente es una llamada remota. Así, en una aplicación de Enterprise JavaBean (EJB) dichas llamadas remotas usan la capa de red sin importar la proximidad del cliente al bean, creando una sobrecarga en la red. Según se incrementa la utilización de estos métodos remotos, el rendimiento de la aplicación se puede degradar significativamente.

Solución: Utilizar un Transfer Object para encapsular los datos de negocio. Se utiliza una única llamada a un método para enviar y recuperar el Transfer Object. Cuando el cliente solicita los datos de negocio al Enterprise Bean, éste puede construir el Transfer Object, rellenarlo con sus valores de atributos y pasarlo por valor al cliente.

Consecuencias: Simplifica el Enterprise Bean y el interface remoto, transfiere más datos en menos llamadas remotas, reduce el tráfico de red, reduce la duplicación de código; podría introducir Transfer Objects obsoletos, incrementar la complejidad debido a la sincronización y el control de versión, accesos y transacciones concurrentes.

Patrones de integración

Data Access Object (26)

Contexto: El acceso a los datos varía dependiendo de la fuente de los mismos. El acceso al almacenamiento persistente, como una base de datos, varía en gran medida dependiendo del tipo de almacenamiento (bases de datos relacionales, bases de datos orientadas a objetos, ficheros planos, etc.) y de la implementación del vendedor.

Problema: Muchas aplicaciones de la plataforma JEE necesitan utilizar datos persistentes en algún momento. Para muchas de ellas, este almacenamiento persistente se implementa utilizando diferentes mecanismos, y hay marcadas diferencias en los APIs utilizados para acceder a esos diferentes mecanismos de almacenamiento. Otras aplicaciones podrían necesitar acceder a datos que residen en distintos sistemas. Existe una dependencia directa entre el código de la aplicación y el código de acceso a datos, que hace difícil y tedioso migrar la aplicación de un tipo de fuente de datos a otro.

Solución: Utilizar un Data Access Object (DAO) para abstraer y encapsular todos los accesos a la fuente de datos. El DAO maneja la conexión con la fuente de datos para obtener y almacenar datos.

Consecuencias: Permite la transparencia, una migración más fácil, reduce la complejidad del código de los objetos de negocio, centraliza todos los accesos a datos en un capa independiente, no es útil para persistencia manejada por el contenedor, añade una capa extra y necesita diseñar un árbol de clases.

Service Activator (26)

Contexto: Los Enterprise Beans y otros servicios de negocio necesitan una forma de activarse asíncronamente.

Problema: Las llamadas a métodos, así como las búsquedas y las llamadas a métodos remotos, son síncronos. El cliente tiene que esperar hasta que esos métodos retornan. Un servicio de negocio como un bean de sesión o de entidad sólo proporciona procesamiento síncrono y esto representa un reto para implementar procesamiento asíncrono.

Solución: Utilizar un Service Activator para recibir peticiones y mensajes asíncronos de los clientes. Cuando se recibe un mensaje, el Service Activator localiza e invoca a los métodos de los componentes de negocio necesarios para cumplir la petición de forma asíncrona.

Consecuencias: Integra JMS en implementaciones Pre-EJB 2.0, proporciona procesamiento asíncrono para cualquier Enterprise Bean.

La dificultad en la aplicación de patrones radica en que estos no son triviales de implementar en aplicaciones empresariales, los desarrolladores necesitan conocer metodologías para una correcta implementación de los mismos. Una aplicación compleja puede demandar el uso de cada uno de estos patrones, por lo que llevaría grandes esfuerzos por parte de los desarrolladores en su implementación, y por consiguiente, consumo de tiempo. Además, un error en su interpretación o implementación puede desencadenar un mal diseño de la arquitectura. En vez de aplicarlos directamente se recomienda la utilización de frameworks de desarrollo. (20)

1.4. Frameworks de desarrollo

Los frameworks de desarrollo son soluciones que implementan patrones y ayudan a desarrollar funciones dentro de las aplicaciones a un nivel superior. Abstraen de complejas implementaciones y hacen el desarrollo más rápido, ya que proveen soluciones a problemas comunes a la hora de lidiar con la tecnología, son como conjunto de librerías que se usan en las aplicaciones.

Los frameworks son elementos a tener en cuenta a la hora de modelar una arquitectura pues hacen que se cubra en gran medida los objetivos con los que debe cumplir la misma. Entre los frameworks más populares de JEE se encuentran:

- Para la capa de presentación: Struts, Spring, JavaServer Faces (JSF), Tapestry, Cocoon, DWR. (27)
- Para la capa de negocio: Spring.
- Para la capa de persistencia: Hibernate, iBATIS, Entity Enterprise Java Bean, TopLink. (28)

Existen otros frameworks como Acegi, que se ocupa de la seguridad; Log4j, para la generación y manejo de logs; JFreeChart, para la generación de gráficas; JasperReports, utilizado para la generación de reportes. A continuación se abordarán características fundamentales de los frameworks más utilizados actualmente: Struts, JSF, Spring, Hibernate y Acegi.

Struts

Struts se encarga de normalizar el desarrollo de la capa Vista dentro de la arquitectura MVC. También proporciona mecanismos para trabajar con la capa Controller; pero nunca Struts se mezcla con la capa del Modelo. Esta característica de Struts permite separar la lógica de presentación de la lógica del negocio. Este framework tiene integrado una serie de patrones de diseño como son: Front Controller, View Helper, Dispatcher View, Service to Worker y Composite View. (20)

Struts tiene un vocabulario específico para definir su funcionamiento, entre los elementos más importantes se encuentran: (20) (27)

- Actions: Posible acción a invocar. Son objetos que heredan de la clase Action donde se escribe que es lo que se hará.
- ActionMaping: Mapea las URLS a acciones. Es decir, se le da un nombre a cada clase acción de manera que puedan ser invocadas desde el cliente como un string.
- ActionServlet: Es el servlet controlador.
- ActionForm: Encapsulan los parámetros de las peticiones de los clientes presentándolos como datos de un formulario. Representan los datos de entrada de la acción a realizar. Un formulario se puede compartir entre varias peticiones de manera que se pueda ir llenando a partes antes de invocar a la acción.

La inteligencia del controlador se define en un archivo XML llamado struts-config.xml. En este archivo se guardan los mapeos, las acciones y los formularios existentes con los que trabajará el framework. Entre las opciones que brinda Struts vale destacar: (20) (27)

- Validación de entradas: Provee mecanismos de validación de las entradas ingresadas, redefiniendo el método validate() de los ActionForms o a través del struts-validator.
- Internacionalización: Brinda soporte para la internacionalización extendiendo la funcionalidad que ya provee Java. Permite manejar dinámicamente el idioma de una aplicación utilizando archivos de texto .properties conteniendo conjuntos de datos con el formato clave=valor y referenciando estas claves en los archivos de la vista.
- Independencia del motor de visualización: Es independiente del motor de visualización aunque generalmente se elija jsp para mostrar las vistas. Las vistas pueden ser procesadas por motores de plantillas como Velocity, transformadores de estilos de documentos XSLT u otros frameworks de presentación como JSF. Por lo tanto no está ligado a un motor de visualización particular, sino que puede convivir con varios de estos, incluso utilizándolos en simultáneo.

Este framework se ha venido utilizando por más de 8 años; posee amplia gama de documentación; permite crear sitios internacionales de manera rápida y efectiva; está liberado bajo licencia Open Source de Apache (27). Entre sus contras está que puede tornarse lento el desarrollo de aplicaciones grandes, pues no consta con herramientas que agilicen el desarrollo de las interfaces; las clases que manejan los formularios están atadas a este framework pues heredan de las clases Action y Action Form (20).

JavaServer Faces

El objetivo específico de este framework es desarrollar aplicaciones web fácil y rápidamente, permitiendo a los desarrolladores pensar en términos de componentes en lugar de solicitudes y respuestas. Dicho de otra forma, permite a los desarrolladores abstraerse de las complejidades de la programación en la web para que se puedan enfocar en la lógica de negocio. Entre las ventajas que brinda este framework están: (27)

- Internacionalización: Está construida sobre la base del soporte que ya brindaba Java, la especificación de servlets y la de JSP. Maneja el concepto de Locale (configuración local) activo. Los posibles Locales que tendrá la aplicación se definen en el archivo de configuración.
- Validación de entradas: Reside en la utilización de Validadores (Validators). A cada componente que recibe la entrada de valores por parte del usuario se le pueden registrar 0 o más validadores. También es posible llamar a los validadores de un componente en cualquier momento. Además de las validaciones de entradas, es posible realizar validaciones a nivel de aplicación o negocio.
- Construcción de componentes: JSF permite la creación de componentes propios con o sin render asociado, de acuerdo a las necesidades de los programadores; aunque esta creación no es sencilla y se necesita crear varios archivos dependiendo el tipo de componente que se desea crear.
- Manejo de Eventos: JSF implementa un modelo que permite la notificación mediante eventos y la subscripción a dichos eventos mediante listeners.
- Enlace entre la Vista y los bean de negocio: Permite establecer enlaces entre los componentes de la vista y los bean del negocio mediante tres maneras: enlace de valores, enlace de métodos y enlace de componentes.

Entre las facilidades que brinda este framework están las de separar claramente el contenido de la presentación y de la lógica, permite modificar el comportamiento de la aplicación sin conocer el

lenguaje en el que está implementado, no es necesario conocer el framework en detalle para poder comenzar a utilizarlo, existe una gran comunidad y la cantidad de herramientas de soporte van en aumento. Por otro lado este framework es dependiente del javascript para su funcionamiento y la creación de componentes es compleja.

Spring

Spring fue creado con el objetivo de dirigir el desarrollo de aplicaciones empresariales. Cualquier aplicación de Java se puede beneficiar de Spring en términos de simplicidad, pruebas y bajo acoplamiento (29). Entre las principales características de Spring están:

- Ligero: El volumen de todo framework puede ser distribuido en una única librería que solamente pesa un poco más de 2.5 MB. El máximo procesamiento requerido es insignificante. Además, Spring es no intrusivo, ya que los objetos dentro de una aplicación que utilice Spring no guardan dependencias con clases específicas dentro del framework. (29)
- Inyección de dependencias: Llamada en sus principios Inversión del Control (Inversion of Control). El principal beneficio de esta técnica es el bajo acoplamiento: los objetos son dotados pasivamente de sus dependencias hacia ellos mismos: el contenedor es el encargado de dar las dependencias a los objetos sin tener que esperar a que sean pedidas (29). Con este mecanismo se obtienen los siguientes resultados: (20)
    - Reducción del código de enlace entre los componentes de la aplicación.
    - Externaliza las dependencias, permite reconfigurarlas sin necesidad de compilar todo el código.
    - Las dependencias son manejadas en un solo lugar, facilitando la configuración de las mismas y reduciendo el margen de errores.
    - Fomenta un buen diseño de la aplicación, pues todo el diseño de la inyección de instancias está basado en interfaces.
- Orientado a aspectos: Permite el desarrollo cohesivo separando la lógica de negocio de los servicios del sistema (seguridad, manejo de transacciones). Los objetos de la lógica de negocio pueden dedicarse enteramente a sus funciones y nada más. (29)

El framework Spring está compuesto por una serie de módulos bien definidos, diseñados para suplir una necesidad específica, que tomados como un todo brindan todo lo necesario para desarrollar aplicaciones empresariales. Esta modularidad hace posible que el framework se utilice tanto o poco dadas las necesidades de una aplicación en particular. Además, Spring brinda puntos

de integración con muchos otros frameworks, ejemplos de estos son: de persistencia, Hibernate, iBATIS y Java Persistence API; de la capa de presentación, Struts, JavaServer Faces, Tapestry y DWR. (29)

Hibernate

Es la solución Object Relational Mapping (ORM) más popular en el mundo Java. Fue liberado bajo la licencia Lesser GNU Public. Constituye un motor de persistencia que implementa múltiples funcionalidades. Hace uso del lenguaje de consultas de Hibernate: Hibernate Query Language (HQL), diseñado como una mínima extensión orientada a objetos de SQL; también permite crear consultas utilizando SQL nativo o consultas basadas en criterios. Soporta todos los gestores de bases de datos SQL y se integra fácilmente con los más populares servidores de aplicaciones JEE y contendores web. (20)

Características claves:

- Persistencia transparente: Puede operar proporcionando persistencia de una manera transparente para el desarrollador.
- Modelo de programación natural: Soporta el paradigma de orientación a objetos de una manera natural (herencia, polimorfismo, composición y las librerías de colecciones de Java).
- Soporte para modelos de objetos con una granularidad muy fina: Permite una gran variedad de mapeos para colecciones y objetos dependientes.
- No necesita mejorar el código compilado: No es necesaria la generación de código ni el procesamiento del mismo en el proceso de compilación.
- Escalabilidad externa: Posee un alto rendimiento, tiene una caché de dos niveles y puede ser usado en un clúster.
- Lenguaje de consultas HQL: Este lenguaje proporciona una independencia del SQL de cada base de datos, tanto para el almacenamiento de objetos como para su recuperación.
- Soporte para transacciones de aplicación: Soporta transacciones largas (aquellas que requieren la interacción con el usuario durante su ejecución).
- Generación automática de claves primarias: Soporta los diversos tipos de generación de identificadores que proporcionan los sistemas gestores de bases de datos así como generación independiente de la base de datos, incluyendo identificadores asignados por la aplicación o claves compuestas.

Acegi

Conocido también por Spring Security, debido a que ha sido siempre un subproyecto de Spring. Es un framework para manejar la seguridad de forma declarativa para aplicaciones basadas en Spring. Provee una solución de seguridad comprensible, maneja la autenticación y autorización tanto en el nivel de la web como al nivel de invocación de métodos. Basado en el framework de Spring, Acegi toma ventaja completamente de sus técnicas de inyección de dependencias y de orientación a aspectos. Está compuesto por cinco componentes: (29)

- Security Interceptor: Previene el acceso a los recursos seguros de una aplicación.
- Authentication Manager: Encargado del proceso de autenticación.
- Access Decision Manager: Encargado de decidir cuando un usuario tiene acceso a un recurso.
- Run-As Manager: Permite reemplazar los permisos para los recursos más profundos de la aplicación.
- After-Invocation Manager: Trabaja de forma diferente a los demás componentes del framework, ya que actúa luego que se accede a un recurso. Es el encargado de mantener la seguridad una vez que un recurso protegido es accedido.

Acegi provee configuraciones de protección del canal, permitiendo redireccionar hacia un canal adecuado (http o https) de acuerdo a la solicitud. También proporciona una librería de etiquetas que puede ser utilizada en JSP, para garantizar que el contenido protegido como enlaces y mensajes sean únicamente mostrados a usuarios que posean los permisos adecuados. Por otra parte, puede manejar la información de la autenticación desde distintos medios de almacenamiento.

1.4.1. Selección de los frameworks a utilizar.

Spring es un framework que tiene el objetivo de facilitar la construcción de aplicaciones Java. Es ligero por el mínimo impacto que tiene en las aplicaciones. Puede ser utilizado en cualquier tipo de aplicación, no solamente web, como es el caso de Struts. Provee servicios similares al del uso de los componentes EJB, como manejo de objetos y gestión de transacciones declarativamente, disminuyendo el tiempo y el esfuerzo en el desarrollo de aplicaciones.

Acegi es un framework que viene integrado dentro de Spring. Provee un mecanismo de seguridad potente y fácil de configurar, permitiendo configurar las políticas de seguridad de manera que no sea necesario modificar el código de la aplicación.

Hibernate crea un puente entre el mundo orientado a objetos de las aplicaciones y el entorno relacional de las bases de datos, enajenado a los desarrolladores del código JDBC y de las

consultas SQL, las cuales, en aplicaciones complejas pueden traer consigo gran cantidad de líneas de código. Se basa en ficheros XML que mapean los objetos con las tablas en la base de datos. Constituye un mecanismo persistente transparente para las clases, las cuales no tienen que implementar ninguna interfaz ni extender de ninguna clase en específico. Crea una capa de abstracción que permite migrar de gestor de base de datos sin cambiar una sola línea de código.

1.5. Ambientes de desarrollo

Un ambiente de desarrollo, o Integrated Development Environment (IDE) en inglés, es un programa compuesto por un conjunto de herramientas para el programador. A continuación se abordarán características de fundamentales de dos de los IDE más utilizados en el mundo del software libre para la programación en Java: Eclipse y NetBeans.

Eclipse

Eclipse provee un conjunto de herramientas para administrar espacios de trabajo; construir, correr y depurar aplicaciones; compartir artefactos con un equipo y hacia versión de código. Es una plataforma que está diseñada para ser infinitamente extendida con cada vez más sofisticadas herramientas. Está construido sobre un mecanismo para el descubrimiento, integración y ejecución de módulos llamados plug-ins. Muchos plug-ins, comúnmente sin relación alguna, pueden ser instalados en una misma instancia de Eclipse, y convivir y cooperar sin problemas para ejecutar una cierta tarea. La clase de producto final incluye aplicaciones IDE, también denominados rich clients (clientes ricos), que se benefician del diseño de la plataforma de Eclipse y sus componentes. (32)

La plataforma Eclipse está habilitada para afrontar las siguientes necesidades:

- Soportar la construcción de gran variedad de herramientas de desarrollo.
- Soportar las herramientas proporcionadas por diferentes fabricantes de software independientes.
- Soportar herramientas que permitan manipular diferentes contenidos (HTML, Java, C, JSP, EJB, XML, y GIF).
- Facilitar una integración transparente entre todas las herramientas y tipos de contenidos sin tener en cuenta al proveedor.
- Proporcionar entornos de desarrollo gráfico (GUI) o no gráficos.
- Ejecutarse en una gran variedad de sistemas operativos, incluyendo Windows y Linux.

NetBeans

Es un Entorno Integrado de Desarrollo gratuito, de código abierto para desarrolladores de software. Consta de un conjunto de herramientas para crear aplicaciones profesionales tanto para escritorio como la empresa, la web y equipos móviles con el lenguaje Java, C/C++, y Ruby. Es fácil de instalar y de uso instantáneo, se ejecuta en varias plataformas incluyendo Windows, Linux, Mac OS X y Solaris. (33)

La construcción del software en NetBeans se realiza de forma modular y ofrece implementados los mecanismos de descubrimiento de nuevos módulos, resolución de dependencias, activación y desactivación de los módulos, comunicación entre los mismos, etc.; permitiendo que se haga hincapié en la lógica de las aplicaciones, dado la posibilidad que se pueda ir extendiendo su funcionalidad a medida que pasa el tiempo. (34)

Otras características son: (34)

- Los proyectos desarrollados no dejan de ser multiplataforma.
- Sistema de ventanas práctico para desarrollar las interfaces de usuario.
- Sistema de ficheros virtual en el cual se montan los diferentes módulos con los cuales se van adaptando automáticamente los menús, barra de herramientas, menús contextuales, etc.
- Soporte completo para desarrollar desde NetBeans IDE.

1.5.1. Selección del ambiente de desarrollo

Eclipse es el ambiente de desarrollo a utilizar. Aunque ambos, Eclipse y NetBeans, ofrecen similares recursos para el desarrollo de aplicaciones, la principal diferencia está en la forma en que está diseñada la interfaz de usuario. Eclipse está organizado sobre el concepto de perspectiva, el cual hace más fácil realizar el trabajo porque solamente las herramientas necesarias están al alcance de la mano, permitiendo además personalizarlas de modo que el programador tenga acceso a las mismas de forma fácil y rápida, lo que agiliza el trabajo e incrementa la productividad. Eclipse cuenta además con un grupo de características adicionales que lo convierten en una potente herramienta de desarrollo como son QuickFix y QuickDiff, y permite una fácil navegación por el código de la aplicación a través de marcadores o los marcos en los editores de texto.

1.6. Servidores de aplicaciones y contenedores web

Los servidores de aplicaciones son los que hacen funcionar la mayoría de las aplicaciones de la plataforma JEE. Estos forman parte de las arquitecturas empresariales: en ellos residen todos los componentes (objetos distribuidos de acceso remoto, páginas web, aplicaciones completas).

Los contenedores web son servidores escritos normalmente en código Java. Su principal funcionalidad radica en recibir peticiones HTTP y ejecutar las clases de Java (Servlets, JSPs). Muchos contenedores web incluyen un pequeño servidor web que les permite utilizar recursos tales como páginas HTML para dar respuestas, formando lo que se llama una aplicación web.

A continuación se mencionarán las principales características de Tomcat, GlassFish y JBoss.

Tomcat

Apache Tomcat es una implementación de las tecnologías JavaServlet y JavaServer Page (35). Es un proyecto de software libre de la Apache Sofware Foundation. Tomcat es desarrollado en un entorno abierto y participativo. Está liberado bajo Apache Software Licence, siendo totalmente libre y gratis su utilización. Al ser un contendor web consume menos recursos que los servidores de aplicaciones, pudiendo adoptarse en máquinas con poca capacidad de rendimiento.

Entre sus características principales están: (36)

- Administrador de seguridad: Ayuda a mantener un control sobre el comportamiento de las aplicaciones web mediante el uso y la configuración de un Java Security Manager.
- SSL: Permite configurar el SSL dentro del servidor para que este responda peticiones a través del protocolo seguro https.
- Clusterización: Puede habilitar sesiones de replicación en un entorno de ejecución de Apache Tomcat.
- Balanceador: Permite configurar, usar y extender el balanceador de carga.

GlassFish

GlassFish es un servidor de aplicaciones que implementa la plataforma JEE5, por lo que soporta las últimas versiones de tecnologías como: JSP, JSF, Servlets, EJBs, Java API para servicios web (JAX-WS), arquitectura Java para enlaces XML (JAXB), metadatos de servicios web para la plataforma Java 1.0, y muchas otras tecnologías. Es gratuito y de código libre, se distribuye bajo la licencia CDDL y la GNU GPL. (37)

Algunas de las características de GlassFish: (38)

- Tiene soporte para otros lenguajes aparte de Java.

- Posee una potente consola de administración gráfica, admite la clusterización desde la consola.
- Permite realizar monitoreo de llamadas desde la consola de administración gráfica.
- Provee balanceo de cargas.

JBoss

JBoss es una plataforma certificada por la JEE para desarrollar y desplegar aplicaciones empresariales, aplicaciones web y portales hechos en Java. El servidor de aplicaciones JBoss provee de todas las ventajas de J2EE 1.4, así como servicios empresariales extendidos incluyendo la clusterización, manejo de memoria caché y persistencia. Además incluye soporte para EJB 3.0, el cual es diseñado para simplificar el modo de programación de las aplicaciones empresariales en Java. (39)

Algunas de las características son: (40)

- Completo soporte J2EE 1.4 (EJB, JCA, JSP, JMX, HTTP, etc.).
- Completa implementación de la seguridad e integración con Java Authentication and Authorization Service (JAAS).
- Clusterización de objetos Java (EJB, HTTP, POJO).

Dentro de su arquitectura cuenta con una capa de aspectos basada en el modelo de programación orientada a aspectos. Esto le permite adicionar transparentemente el comportamiento suministrado por un servicio a cualquier objeto.

1.6.1. Selección del entorno de ejecución.

Tomcat es un contenedor web fácil y sencillo de administrar y configurar, no es necesario el uso de un servidor de aplicaciones ya que no se hará uso alguno de los EJB. Consume menos recursos que un servidor de aplicaciones. Es gratis y posee un gran número de usuarios y soporte en la comunidad mundial. Es compatible con la mayoría de los API de JEE. Permite lograr escalabilidad mediante clúster. Este contenedor presenta un manejador de reserva de conexiones que se configura de manera que desde la aplicación se acceden a estas conexiones, este mecanismo es más eficiente que el de realizar la conexión en el momento que requiera. Tomcat provee también una consola Web para manipular las características básicas de las aplicaciones desplegadas en él, además de mucha información del estado y monitoreo que se pueda requerir.

1.7. Sistemas gestores de base de datos

En la actualidad, prácticamente todos los programas que manejan información utilizan bases de datos debido a las ventajas que estas ofrecen sobre los otros tipos de almacenamiento en cuanto a organización, rendimiento y manejo de la información. Para el acceso, adición y procesamiento de la información almacenada en una base de datos es necesaria la utilización de un sistema gestor de base de datos. PostgreSQL y MySQL son los sistemas gestores de base de datos más utilizados dentro del mundo Open Source, a continuación se mencionan algunas de sus principales características.

PostgreSQL

Es un proyecto iniciado en la universidad de Berkeley con más de 15 años de vida. Está bajo licencia BSD, lo que le da más libertades que, por ejemplo, la GPL (41). PostgreSQL incluye características de la orientación a objetos, como puede ser: la herencia, tipos de datos, funciones, restricciones, disparadores, reglas e integridad transaccional.

Las principales características de PostgreSQL son: (41)

- Atomicidad: Asegura que la operación se ha realizado o no, y por lo tanto ante un fallo del sistema no puede quedar a medias.
- Consistencia: Asegura que sólo se empiece aquello que se puede acabar. Se ejecutan las operaciones que no van a romper la reglas y directrices de integridad de la base de datos.
- Aislamiento: Asegura que una operación no puede afectar a otras. Esto asegura que dos transacciones sobre la misma información nunca generará ningún tipo de error.
- Durabilidad: Asegura que una vez realizada la operación, esta persistirá y no se podrá deshacer.
- Corre en casi todos los sistemas operativos.
- Soporte nativo para los lenguajes más populares del medio: PHP, C, C++, Perl, Python, etc.
- Soporte de todas las características de una base de datos profesional (triggers, store procedures, funciones, secuencias, relaciones, reglas, tipos de datos definidos por usuarios, vistas, vistas materializadas, etc.).

Entre las desventajas de utilizar este sistema están el consumo de recursos y la velocidad de respuesta, pues es uno de los sistemas que más carga el servidor y es mucho más lento que la mayoría de los sistemas gestores de base de datos; aunque su velocidad parece permanecer invariable aun cuando trabaja con bases de datos grandes.

MySQL

Es un sistema gestor de base de datos relacional: almacena la información en tablas separadas en vez de ponerla toda en un gran almacén, lo que añade velocidad y flexibilidad en su funcionamiento. Está liberado bajo licencia GPL, lo que posibilita que el software sea utilizado y modificado por cualquier persona. Fue originalmente diseñado para manejar grandes volúmenes de datos mucho más rápido que los otros gestores existentes. Es catalogado como el gestor más rápido. Puede trabajar en modo cliente/servidor o en sistemas embebidos. (42)

Las principales características de MySQL son: (42)

- Trabaja en varias plataformas.
- Tiene un diseño multicapa con módulos independientes.
- Completamente multihilo, utilizando hilos del núcleo. Puede fácilmente utilizar múltiples CPU en caso de que estén disponibles.
- Las funciones SQL están implementadas utilizando una librería de clases optimizada para que sea tan rápida como sea posible. Usualmente no hay asignación de memoria.
- Manejo de diferentes tipos de datos como Date, TimeStamp, Year, Set, Enum, etc.
- Provee motores de almacenamientos transaccionales y no transaccionales.

MySQL se torna lento en bases de datos con muchas tablas a la hora de abrir, cerrar y crear una base de datos. Por otro lado si se ejecutan consultas sobre muchas tablas distintas puede que se sobrecargue un poco cuando la memoria cache de las tablas está llena.

1.7.1. Selección del sistema gestor de base de datos

PostgreSQL 8.2, es un gestor de bases de datos que tiene prácticamente todo lo que poseen los gestores comerciales de estos tiempos. Además de los tipos de datos base también soporta datos de tipo fecha, datos sobre redes (MAC, IP...), cadenas de bits, al mismo tiempo que permite la creación de tipos propios. Incluye ordenamientos en memoria y en disco más rápido, mejor escalabilidad en sistemas multi-procesador, optimización de consultas sobre datos particionados, cargas masivas más rápidas y outer joins considerablemente acelerados. Permite a los administradores crear fácilmente una copia para recuperación inmediata de su clúster de bases de datos. La construcción de índices puede ocurrir mientras las aplicaciones escriben en las tablas de la base de datos, permitiendo el afinamiento del rendimiento sin afectar la disponibilidad.

Para esta propuesta se tuvo en cuenta fundamentalmente que es un gestor de base de datos aprobado por la Dirección de Informatización de la Universidad de las Ciencias Informáticas, y que recientemente se creó la Comunidad Cubana de PostgreSQL.

1.8. Conclusiones parciales.

Para realizar el diseño de la arquitectura de software de un sistema es necesario tener en consideración una serie de aspectos como bajo que metodología se trabajará, siendo RUP la más indicada para la construcción del sistema. La plataforma Java nos provee de un conjunto de herramientas suficientes para la construcción de aplicaciones empresariales. Los patrones más utilizados en las diferentes capas de una aplicación empresarial de Java ayudan a prevenir y solucionar muchos problemas que se pueden presentar a la hora del desarrollo; pero, a su vez, pueden ser muy costosos de implementar, por lo que en vez de diseñarlos directamente es mejor utilizar un conjunto de frameworks que los implementen, de forma que se optimice el diseño y se gane en tiempo de desarrollo. Se seleccionó, del grupo de las principales herramientas y tecnologías existentes para la plataforma empresarial de Java, cuales serán utilizadas para la construcción del sistema.

CAPÍTULO 2: SOLUCIÓN PROPUESTA

En este capítulo se describe la arquitectura de SINAPSIS. Para la descripción de la arquitectura se tuvieron en cuenta un grupo de restricciones las cuales se especifican en los requisitos no funcionales (Anexo 1). La descripción de la arquitectura abarca algunas de las principales vistas arquitectónicas y se analizan y definen aspectos de importancia para la arquitectura del sistema. Las vistas que se describen son la Vista de Casos de Uso, la Vista Lógica, la Vista de Datos y Vista de Despliegue, utilizando para ello una plantilla proporcionada por ALBET para tal efecto (Anexo 2).

2.1. Objetivos y restricciones arquitectónicas

2.1.1. Distribución geográfica

Los usuarios del sistema se encuentran distribuidos por todo el país organizado en diferentes organismos, por lo que es necesario garantizar el acceso de los mismos independientemente de la zona geográfica donde se encuentren.

2.1.2. Disponibilidad de la información

La información que se genere en el sistema por cada uno de los usuarios del mismo, se debe centralizar de manera que permita obtener informes centralizados y resúmenes que ayuden a la toma de decisiones.

2.1.3. Seguridad

El sistema manejará información sensible por lo que debe contar con un sistema de seguridad que restrinja el acceso al mismo y delimite permisos a cada una de las funciones que se puedan realizar. Además se debe delimitar el alcance de cada usuario a la información de acuerdo al nivel jerárquico del mismo (Usuario de un ministerio, de un organismo, de un departamento, etc.) Las claves de acceso de los usuarios deben ser almacenadas utilizando algún algoritmo de encriptación de manera que esta información sea personal. La información sensible que maneje el sistema no debe viajar por la red en texto plano. Los servidores del sistema deben poder ser accedido sólo por los protocolos y los puntos de accesos bien definidos.

2.1.4. Trazas

Dada la gran cantidad de usuarios en el sistema y la sensibilidad de la información manejada, se hace necesario tener un registro de de cada una de la acciones realizadas en el sistema por cada usuario y en qué fecha.

2.1.5. Interacción con otros sistemas

El sistema debe ser capaz de interactuar con otros sistemas (SIPRE, SIGECOF) mediante estándares como servicios web, de manera que se puedan exponer funcionalidades como servicios de una manera flexible así como invocar funcionalidades de estos sistemas.

2.1.6. Sistemas legados

Se deben incorporar la información gestionada por un sistema anterior (Nueva Etapa), por lo que es necesario realizar una migración de datos de la base de datos anterior a la base de datos de la nueva aplicación teniendo en cuenta las diferencias de estructuras y gestores de las mismas.

2.2. Tamaño y rendimiento

2.2.1. Tamaño

El sistema estará prestando servicios en línea por todo el país, desde todos los ministerios, entes y organismos, así como otras instituciones. Debe posibilitar además la incorporación de nuevas instituciones con nuevos usuarios.

El sistema constará con una aplicación web desarrolla en la plataforma JEE (Java Enterprise Edition) plataforma empresarial de Java y cuenta con 7 módulos:

- Registro y aprobación de proyectos.
- Acciones centralizadas
- Seguimiento
- Reportes
- Administración
- Configuración

Existirá además un módulo encargado de la migración de datos correspondientes al sistema Nueva Etapa hacia la base de datos del nuevo sistema, este módulo no formará parte del centro de

la solución de software sino que será desarrollado de manera independiente y utilizada sólo para la carga inicial.

2.2.2. Niveles de concurrencia y tiempo de respuesta.

En cada organismo deben existir entre 2 (dos) y 16 (dieciséis) usuarios como promedio operando de forma centralizada y demandando servicios del servidor Web. Según estos datos y basándonos en el sistema anterior (Nueva Etapa) se estima que el sistema contenga alrededor de 10000 usuarios, según la carga de trabajo se espera en horario pico una concurrencia de 5000 peticiones en un minuto.

Independientemente al lugar en que se encuentre el cliente el tiempo de respuesta del servidor a las peticiones del usuario debe tener un máximo de 8 segundos. Para garantizar estos tiempos de respuesta en horario pico teniendo en cuenta que el peso promedio a descargar por cada solicitud es aproximadamente de 100 kbyte, es necesario un ancho de banda en la conexión del servidor de 8 Mbps.

2.2.3. Estimación del volumen de datos a manejar

Para estimar el volumen de datos se tomó como referencia la información almacenada por el sistema Nueva Etapa el cual se encuentra en explotación desde el año 2004. Así se pudo estimar el nivel de crecimiento de la data almacenada. Del estudio anterior se pudo llegar a la siguiente conclusión:

Valor aproximado total por proyecto: 0.009 MB

Valor aproximado para 5000 proyectos: 0.04 GB

Crecimiento anual de información de proyectos (nuevos ingresos y datos adicionales): 0.004 GB

A este nivel de crecimiento se requieren aproximadamente 20 años para ocupar 20 GB de almacenamiento, sólo para los datos del SINAPSIS.

Adicionalmente se requiere espacio para almacenar los sistemas operativos de los 4 servidores del centro de datos, a razón de 5 GB por cada uno sumaría un total de 20 GB.

En total, según estas estimaciones, la carga inicial de datos almacenados sería de unos 1 GB.

2.2.4. Servidor Web

Teniendo en cuenta la importancia estratégica del sistema, el nivel de concurrencia de usuarios al mismo, se diseñó la siguiente configuración del servidor web con el objetivo de lograr una disponibilidad permanente del sistema con un rendimiento aceptable.

El sistema contará con 4 servidores Web (Contenedores de Servlet) Apache Tomcat versión 5.5 utilizando la máquina virtual de java versión 6.0 (JRE 6.0), los cuales van a trabajar en clústeres, a estos les vamos a llamar servidores de aplicación; además tendrá 2 servidores Web (Apache2) que van a ser los encargados de balancear la carga entre los 4 servidores de aplicación, estos servidores serán los que darán repuestas a las peticiones de los usuarios y sus configuración en clúster permitirá dividir la carga de peticiones y aumentar la latencia a fallos del sistema permitiendo que si un servidor presenta algún problema otro se pueda hacer cargo de las peticiones, para esto se implementa un sistema de réplica de sesiones.

Como la aplicación va a tener un solo nombre (DNS de la URL) es necesario implementar en el DNS del centro donde se instale el sistema un (round-robin) o (load balancer) el cual va a tener 2 registros tipo A con el mismo nombre DNS de la aplicación pero con 2 direcciones IP diferentes que serían la de los 2 servidores Web (Apache), que van a ser los que en realidad balancean carga entre 2 servidores de aplicación cada uno. Los requerimientos de hardware estimados para el correcto funcionamiento del sistema son los siguientes.

- Micro Dual Quad Core Xeon E5440 con 12 megas de cache, 8GB de memoria RAM y 40GB de disco duro para cada uno de los 4 servidores de aplicación.
- Micro Dual Quad Core Xeon E5440, 4GB de memoria RAM para cada balanceador de carga y 40 GB de disco duro.
- Para cada servidor se usará Sistema Operativo Devian etch.

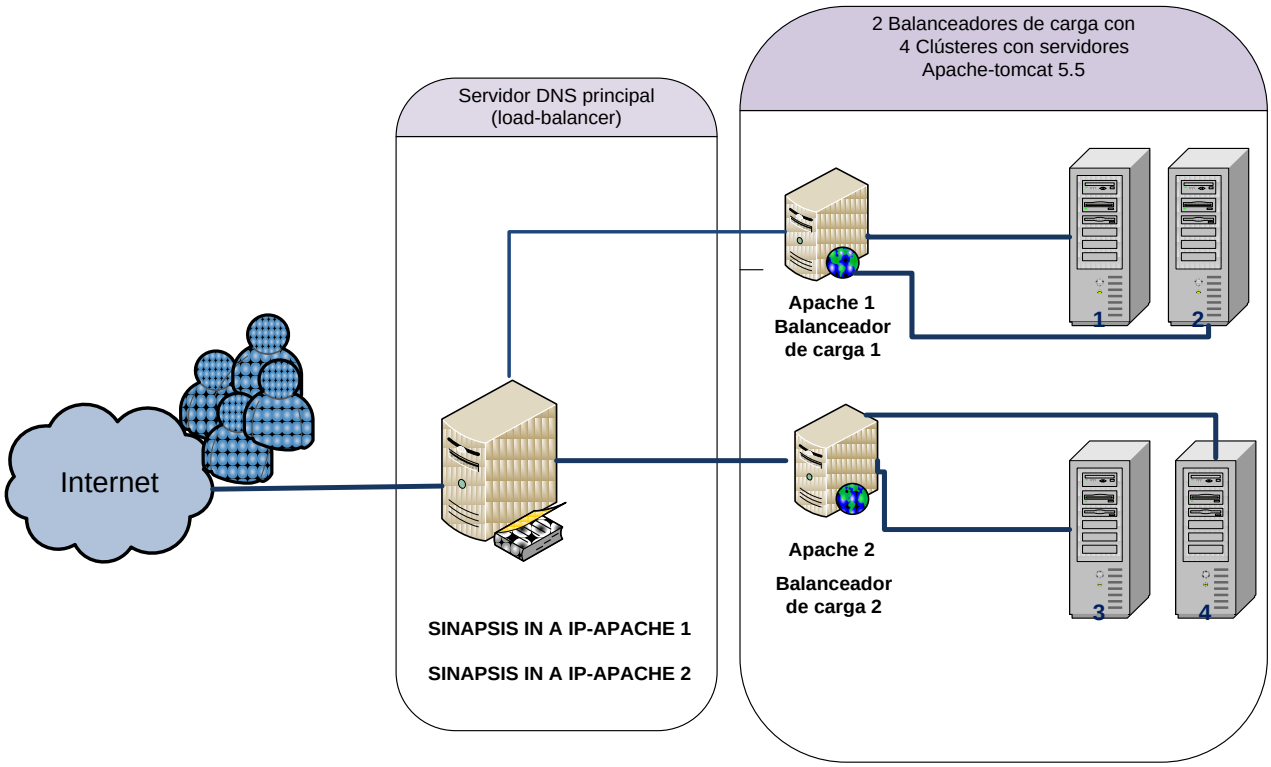

Figura 2.1 Representación de los servidores de aplicación.

2.2.5. Centro de datos

Teniendo en cuenta la cantidad de usuarios conectados concurrentemente al sistema así como la cantidad de procesamiento del mismo y con el objetivo de lograr mayor rendimiento y disponibilidad de este en la base de datos, se concibe un sistema instalado en un clúster formado por 5 servidores.

Para esto se utilizaría Cybercluster, que no es más que un multi-maestro, la solución de la replicación sincrónica basado en PostgreSQL 8.2., software a escala empresarial que proporciona tecnología de replicación de alto-extremo a las compañías que utilizan Postgres.

La base de datos estará distribuida en 3 nodos, donde Cybercluster sincroniza las escrituras al banco de datos para asegurarse que todos los nodos del banco de datos contengan los mismos datos, lo que significa que los datos son guardados consistentemente por todos los nodos del banco de datos desde cualquier punto. En adición al acceso de lectura se encuentra el balanceador de carga que asegura que todos los nodos de los bancos de datos pueden trabajar eficazmente al mismo tiempo.

La composición del clúster quedaría como se muestra en la Figura 2.2, formado por un servidor balanceador de carga, 3 nodos de base de datos y un servidor de replicación.

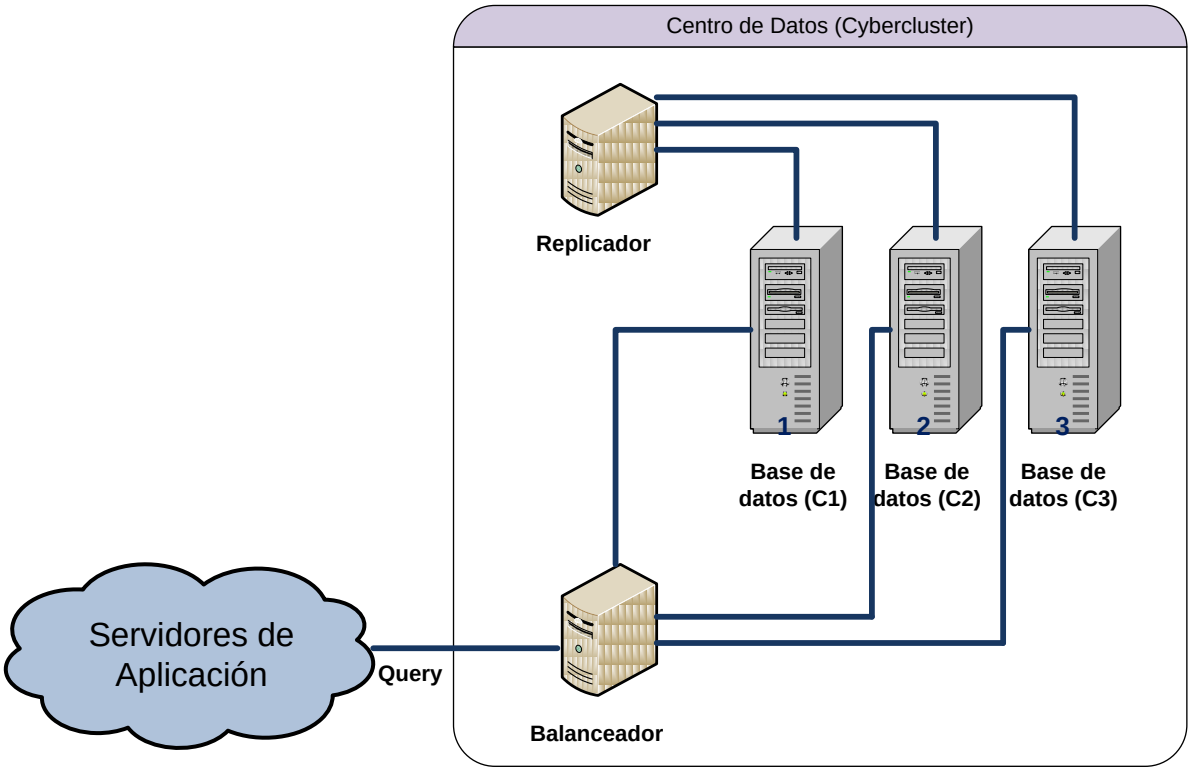

Figura 2.2 Representación del Centro de Datos

2.2.6. Los nodos de base de datos

Los clúster (nodos) son las máquinas que procesan las demandas reales que entran. Siempre que una demanda se reciba del cliente, un clúster de BD tiene que determinar si está enfrentando una demanda de lectura o escritura. La demanda de lectura es ejecutada directamente por el banco de datos y el resultado se envía al cliente. En caso de que sea una demanda de escritura, el clúster BD envía un mensaje al servidor encargado de la replicación para asegurarse que ese dato se envía a todos los nodos del banco de datos restantes dentro del conjunto.

2.2.7. Servidor de la Replicación

El servidor de la Replicación es el componente central del sistema. Toma demandas entrantes sólo de los nodos del banco de datos y copias esos cambios a todos los nodos del banco de datos en el sistema. Si el servidor de replicación descubre un problema en un nodo del banco de datos quitará la máquina de la lista de banco de datos activos y se escribirá al disco un log que contiene una descripción del error detallada. Esto tiene que ser hecho para asegurarse que los nodos del

banco de datos nunca pueden estar fuera de sincronización en caso del fracaso. Basado en los logs el administrador del sistema puede decidir entonces si es posible agregar la máquina de nuevo al grupo de nodos.

2.2.8. Servidor de Balanceador de Carga

CyberCluster contiene su propio balanceador de carga que puede usarse para distribuir la carga dentro del conjunto de nodos. La carga en el sistema es determinada por el número de preguntas activas. La máquina con el número más bajo de preguntas activas será escogida para realizar una nueva demanda. Si el balanceador de carga detecta un problema en un nodo activo del banco de datos automáticamente destaca este nodo dentro del sistema activo para asegurarse que ninguna otra demanda se enviará a esta máquina rota.

Ventajas: Alta disponibilidad del sistema. Funcionamiento autónomo. Rapidez en los tiempos de respuesta ante peticiones al sistema. Mejora la seguridad del sistema físicamente y el costo consecuente con las modificaciones necesarias. Se cuenta con Balanceo de Carga, Replicación de Datos, Tolerancia a fallos. Cuando un servidor cae y se vuelve a poner en funcionamiento automáticamente inicia la sincronización de datos.

Desventajas: Se deben garantizar condiciones de conectividad óptimas para que la aplicación pueda ser consumida sin problemas.

2.2.9. Requerimiento de los servidores

Teniendo en cuenta que cada servidor tendrá configurado un máximo de 2000 conexiones simultáneas y que en PostgreSQL cada conexión es un proceso, y cada proceso utiliza memoria la cual es reservada de acuerdo al max_connections (que indica el número máximo de conexiones simultáneas que manejará el servidor) una vez que se inicia el servidor, por lo que mientras más memoria se tiene ocupada en procesos, menos se puede usar para caché, shared_buffers, etc. Teniendo en cuenta esto un servidor con 2000 conexiones configuradas para cada valor de max_connections tenemos que tener al menos el doble para shared_buffers (parámetro que más afecta al rendimiento de PostgreSQL). Este valor, de tipo entero, indica el número de bloques de memoria o buffers de 8KB (8192 bytes) que Postgres reservará, como zona de trabajo, en el momento del arranque para procesar las consultas). Una recomendación es comenzar asignar el 10% del total de la memoria RAM para shared_buffers y a partir de ahí, ir aumentando o disminuyendo dicho porcentaje en función del rendimiento y la paginación.

Por tanto:

El 10% de 8 GB: (8388608 KB/10) = 838860,8 KB

shared_buffers: (838860,8 KB/8 KB) = 104857,6

Por otro lado existe otro parámetro importante: WORK_MEM.

Este parámetro configura el espacio de memoria que Postgres utiliza para realizar ordenaciones de tablas o de resultados parciales de consultas, sobre todo en cláusulas ORDER BY, CREATE INDEX o MERGE JOIN.

Este valor es más difícil de configurar porque depende, por un lado, de lo grande que sean las tablas o resultados que hay que ordenar, y por otro, del número de peticiones simultáneas para esa misma consulta (para cada una se empleará la misma cantidad de memoria).

Un buen comienzo es asignar entre un 2% y un 4% del total de la memoria si prevemos pocos accesos simultáneos a grandes sesiones de ordenación y mucho menor, si esperamos muchos accesos simultáneos a sesiones de ordenación pequeñas. Para nuestro caso, hemos optado por usar un 8% de la memoria:

El 8% de 8 GB: 671088,64KB (8388608 KB*8)/100

work_men = 671088,64

A partir de aquí se considera necesario para un adecuado funcionamiento del sistema utilizar:

- Micro Dual Quad Core Xeon E5440 con 12 megas de cache, 8GB de memoria RAM y 146GB SAS de disco duro para cada uno de los 3 nodos del banco de datos.
- Micro Dual Quad Core Xeon E5440, 4GB de memoria RAM para el balanceador de carga y 40 GB de disco duro.
- Micro Dual Quad Core Xeon E5440, 4GB de memoria RAM para el replicador de datos y 40 GB de disco duro.
- Para cada servidor se usará Sistema Operativo Devian etch.

2.2.10. Respaldo de la Base de Datos

Se conciben salvas automáticas de la base de datos en un servidor a parte de los 3 nodos del banco de datos. Esta salva se realizaría diariamente en un horario donde el trabajo con el sistema sea el menor posible, preferiblemente en horario nocturno. Esta salva automática crearía una especie de backup a la base de datos en el cual se guardará de forma segura toda la información existente, para de esta forma prevenir una pérdida total de los datos debido a cualquier falla que exista en la base de datos.

Para garantizar este respaldo se usará la herramienta pg_dump la cual configurándole algunos parámetros necesarios se realiza y con el programador de tareas de Linux se le asigna que ejecute cada cierto tiempo el script que tiene el código encargado de generar el bacukp guardando en el servidor toda la información existente de la base de datos.

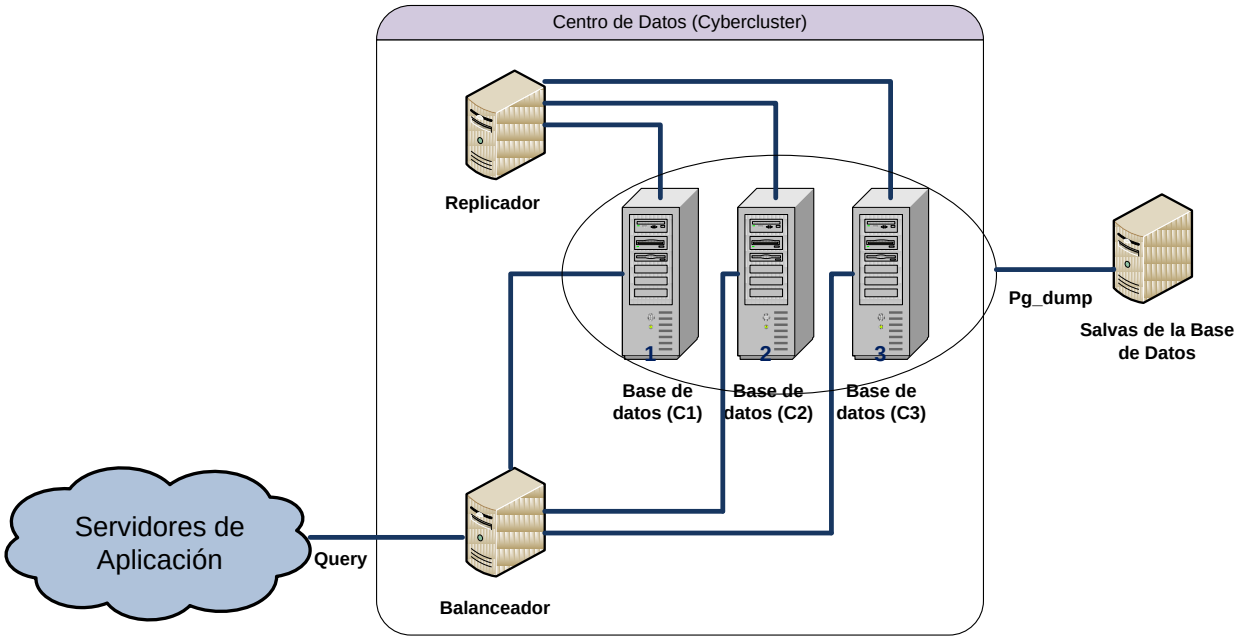

Figura 2.3 Representación del centro de datos y el servidor de salvas

2.3. Vista de casos de uso

A continuación se muestra el diagrama de casos de usos que tienen un peso significante y determinan elementos claves en la arquitectura (Arquitectónicamente significativos).

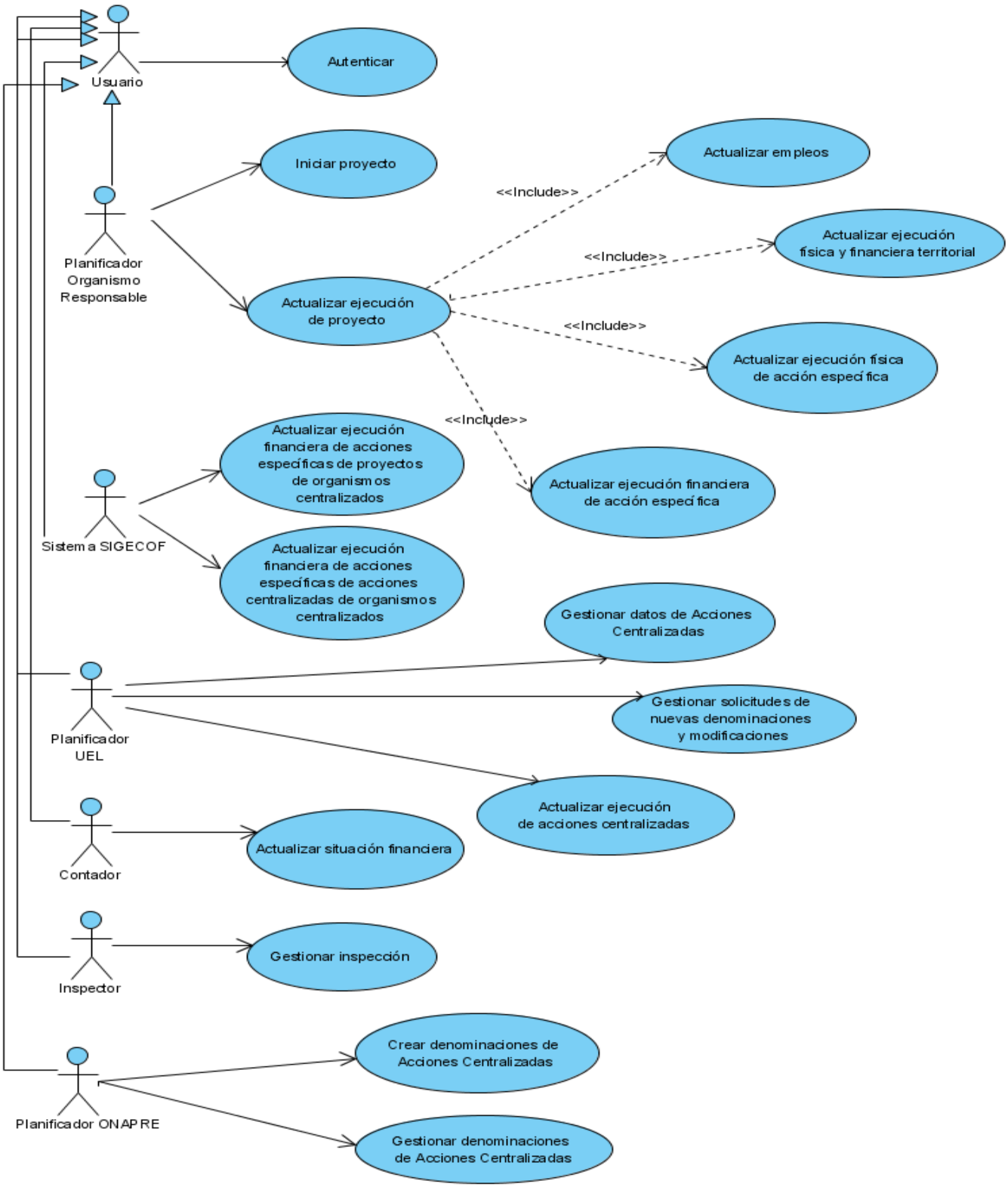

Figura 2.4 Casos de uso arquitectónicamente significativos.

2.3.1. Breve descripción de los casos de uso.

Seguidamente se describen brevemente cada uno de los casos de uso arquitectónicamente significativos.

Autenticar

El caso de uso se inicia cuando el Usuario introduce los datos para autenticarse en el sistema. El sistema verifica que todos los datos necesarios hayan sido insertados y de forma correcta, dando acceso al sistema según nivel de acceso, finalizando el caso de uso.

Iniciar proyecto.

El caso de uso se inicia cuando el planificador de un organismo decide proponer el inicio de uno de los proyectos con los que esté asociado para empezar a ejecutarlo. Termina proponiendo el inicio de los proyectos.

Actualizar ejecución.

El caso de uso se inicia cuando el planificador de un organismo se dispone a actualizar la ejecución de un proyecto. Para ello se actualizan los tipos de seguimiento que se le da al proyecto. El planificador del organismo puede guardar la actualización que realiza para actualizarla posteriormente o enviarla a revisión para que sea aprobada. Termina con proyectos con la ejecución actualizada.

Actualizar empleos.

El caso de uso se inicia cuando es invocado desde el caso de uso Actualizar ejecución de proyecto. Consiste en actualizar la ejecución de los empleos del proyecto por parte del planificador de un organismo como parte de la actualización de la ejecución de un proyecto. Termina con la actualización de los empleos del proyecto actualizada.

Actualizar ejecución física y financiera territorial.

El caso de uso se inicia cuando es invocado desde el caso de uso Actualizar ejecución de proyecto. Consiste en actualizar la ejecución física y la financiera distribuidas territorialmente por parte del planificador de un organismo, como parte de la actualización de la ejecución de un proyecto. Termina con la ejecución territorial del proyecto actualizada.

Actualizar ejecución física de acción específica.

El caso de uso se inicia cuando es invocado desde el caso de uso Actualizar ejecución de proyecto. Consiste en actualizar la ejecución física de las acciones específicas, por parte del planificador de un organismo, como parte de la actualización de la ejecución del proyecto. Termina con la actualización de la ejecución física de las acciones específicas de un proyecto actualizada.

Actualizar ejecución financiera de acción específica.

El caso de uso se inicia cuando es invocado desde el caso de uso Actualizar ejecución de proyecto. Consiste en actualizar la ejecución financiera de las acciones específicas de un ente descentralizado por parte del planificador, como parte de la actualización de la ejecución del proyecto. Termina con la ejecución financiera de las acciones específicas actualizada.

Actualizar ejecución financiera de acciones específicas de proyectos de organismos centralizados.

El caso de uso se inicia cuando el sistema SIGECOF actualiza el seguimiento financiero de las acciones específicas de los proyectos de los organismos. Consiste en actualizar la ejecución financiera de los organismos centralizados, una vez al mes por parte del sistema SIGECOF, reflejando el comprometido, causado y pagado de las fuentes de financiamiento para cada una de las partidas por imputaciones presupuestarias de las acciones específicas de proyectos.

Actualizar ejecución financiera de acciones específicas de acciones centralizadas de organismos centralizados.

El caso de uso consiste se inicia por el sistema SIGECOF cuando este se dispone a actualizar la ejecución financiera de las acciones específicas de las acciones centralizadas de los organismos centralizados. Consiste en reflejar el comprometido, causado y pagado de las fuentes de financiamiento para cada una de las partidas por imputaciones presupuestarias de las acciones específicas de acciones centralizadas

Actualizar ejecución de acciones centralizadas.

El caso de uso se inicia cuando el planificador de un organismo se dispone a actualizar la ejecución de las acciones centralizadas de su UEL. Consiste en actualizar la ejecución financiera de

las acciones específicas de las acciones centralizadas. Termina con la ejecución de las acciones centralizadas actualizada.

Actualizar situación financiera.

El caso de uso se inicia cuando el contador de un ente adscrito actualiza la situación financiera de su entidad. Consiste en actualizar las cantidades de dinero disponible y la pagada por cada fuente de financiamiento. Termina con la situación financiera de un organismo actualizada.

Gestionar inspección.

El caso de uso se inicia cuando un inspector se dispone a gestionar las inspecciones que se le realizan a los proyectos durante su ejecución. Consiste en crear, modificar, eliminar y ver los detalles de las inspecciones. Termina con inspecciones creadas, modificadas o eliminadas.

Gestionar datos de acciones centralizadas

Consiste en editar y obtener datos de las acciones centralizadas que se le asigno a cada organismo.

Gestionar solicitudes de nuevas denominaciones y modificaciones

Consiste en crear, modificar, obtener, eliminar y enviar solicitudes de creación de nuevas denominaciones y modificaciones de Acciones Centralizadas.

Crear denominaciones de acciones centralizadas

Consiste en crear nuevas denominaciones de acciones centralizadas a nivel central por la ONAPRE.

Gestionar denominaciones de acciones centralizadas

Consiste en modificar, obtener, eliminar y habilitar denominaciones de acciones centralizadas.

2.4. Vista Lógica

2.4.1. Visión general de la arquitectura – Alineamiento de paquetes, subsistemas y capas.

El SINAPSIS se desarrollará bajo la tecnología Java Enterprise Edition, se utilizará una arquitectura separada por varias capas lógicas utilizando como framework arquitectónico base a Spring Framework apoyándonos en su contenedor de objetos y su técnica de inyección de instancias para enlazar la capa de presentación y la capa de acceso a datos con la capa de negocio logrando el menor acoplamiento posible y beneficiándonos de su módulo orientado a aspecto para manejar las transacciones, la seguridad, entre otros.

En la capa de acceso a datos se utilizará el framework de mapeo objeto relacional Hibernate, así como en la capa de presentación para realizar el manejo de las solicitudes y toda la lógica de presentación se utilizará el framework SpringMVC.

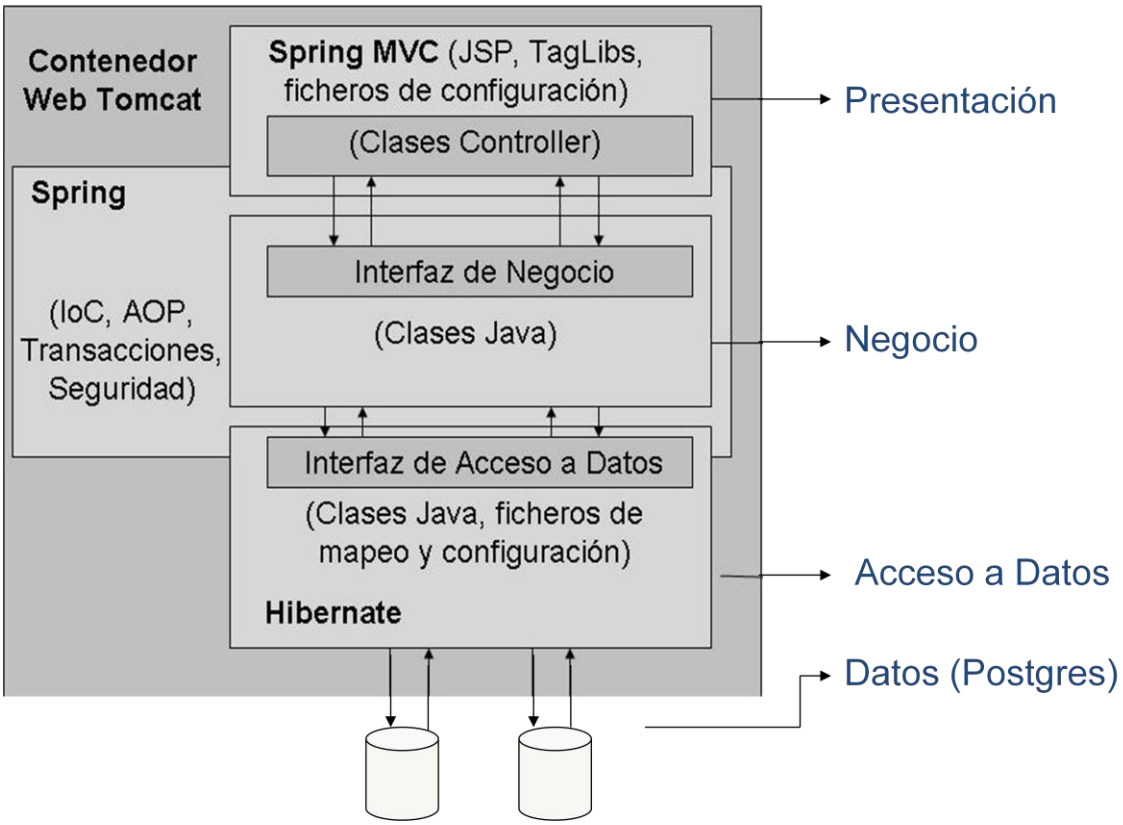

Figura 2.5 Representación de las capas lógicas de la aplicación

En esta aplicación web sobre JEE, todas las capas lógicas corren en un servidor web. No es una arquitectura distribuida aunque pudiera adaptarse sin mucho esfuerzo a un sistema que lo requiera.

Cada capa puede ser modificada tanto como sea posible sin provocar un impacto en las demás capas. Una capa no es consciente de lo que le ocurre a la capa superior, su dependencia es puramente con la capa inmediata inferior. Esta dependencia entre capas es normalmente entre interfaces, asegurando que el acople sea el más bajo posible.

Los “objetos persistentes del dominio (entidades)” representan el modelo de objetos o conceptos reales tales como una cuenta bancaria o un expediente escolar. Este conjunto de objetos puede llegar a considerarse otra capa lógica que puede presentar este tipo de arquitectura. Estos objetos de dominios tradicionalmente son pasados hasta la capa de presentación, en la cual mostrarán los datos que contienen, pero no podrán ser modificados, ya que esto sólo ocurre solamente dentro de los contextos transaccionales definidos en la capa de servicios de negocio. De esta manera no hay necesidad de uso de los Transfer Objects, como se usa en las arquitecturas tradicionales de JEE.

Seguidamente se analizará en más detalles cada una de las capas propuestas anteriormente comenzando de abajo hacia arriba.

2.4.1.1. Capa de acceso a datos

La interacción del negocio con la capa de persistencia se realizará mediante el uso de interfaces. El término de Objeto de Acceso a Datos o en inglés, Data Access Object (DAO) es ampliamente usado en el desarrollo de software.

Los DAOs encapsulan la persistencia de los objetos de dominios, proveen la persistencia de los objetos transitorios y las actualizaciones de los objetos existentes en la base de datos. Las implementaciones de los DAOs estarán disponibles para los objetos (típicamente para los objetos de negocio) haciendo uso de la inyección de dependencias con los objetos de negocio y las instancias de los DAOs, configurada en el contenedor de inversión de control de Spring Framework.

Las interfaces de los DAOs contienen básicamente los siguientes tipos de métodos:

- **Métodos para descubrir:** Estos localizan los objetos almacenados para ser usados por la capa de servicios de negocio.
- **Métodos para persistir o salvar:** Estos hacen persistentes a los objetos transitorios.
- **Métodos para eliminar:** Estos eliminan a los objetos guardados en el medio de almacenamiento (generalmente una base de datos).

- **Métodos para conteos y otras funciones agregadas:** Estos devuelven los resultados de operaciones que son más eficientes implementarlas usando funcionalidades de la base de datos (procedimientos almacenas, etcétera.) que iterar sobre los mismos objetos.

2.4.1.2. Capa de servicios de negocio

En esta capa radican los objetos de negocio o Business Objects. Los objetos de negocio separan los datos y la lógica de negocio usando un modelo de objetos.

Estas son las funcionalidades que presenta esta capa:

- **Lógica de negocio específica de procesos de negocios:** A veces es más oportuno para los objetos de dominio contener lógica de negocio aplicable a muchos casos de uso específicos. Esta definición de arquitectura los objetos de dominio no presentan ningún tipo de lógica de negocio, sino que esta responsabilidad recae sobre los objetos de negocio, permitiendo usar a los objetos de dominio como objetos de transferencia que se mueven entre las capas arquitectónicas de la aplicación.
- **Puntos de entrada muy bien definidos para las operaciones de negocio implementadas:** Los objetos de negocio brindan las interfaces usadas por la capa de presentación.
- **Control de transacciones:** Las políticas de transaccionales de la aplicación serán planteadas al sobre los objetos de negocio.
- **Ejecución de restricciones de seguridad:** Las restricciones de seguridad en esta capa ser realizará en los puntos de entradas a la capa media, es decir en los objetos de negocio.
- **Ejecución de la auditoría:** Sobre esta capa se llevará a cabo la auditoría sobre los métodos de negocio.

En la figura se muestra en más detalles el diseño básico de clases en esta capa. Por cada módulo se definirá una o más fachadas en caso de que se requiera que agrupen los métodos de negocio implementados en los manejadores o Managers. La fachada de un módulo se basa en el patrón Facade para permitir una clara división entre las capas arquitectónicas. Los Managers son las clases que se especializan en un conjunto de funcionalidades que representan el negocio sobre una o varias entidades. Los Managers son las únicas clases en la aplicación que tendrán lógica de negocio mientras que las fachadas se limitarán solamente a agrupar las funcionalidades de para ser expuestas a capas superiores.

2.4.1.3. Capa de presentación

La capa de presentación descansa sobre una capa de servicios de negocio. Esto significa que esta capa será fina y no contendrá lógica de negocio, sino simplemente lo concerniente a aspectos de presentación, por ejemplo, el código para manipular las interacciones web. Esta capa puede variar en complejidad en el transcurso del desarrollo de la aplicación, sin embargo ninguna de las capas inferiores debe ser afectada por estos cambios.

2.4.1.4. Capa Web

En la capa Web se utilizara Spring MVC como framework MVC para manejar el flujo de datos entre el cliente y las capas inferiores.

Esta capa web es la responsable de tratar con las interacciones del usuario y obtener los datos que pueden ser mostrados en un formato determinado. Estará compuesta por tres tipos de objetos:

- **Controladores:** Estos objetos son responsables de procesar las entradas del usuario en forma de peticiones HTTP, invocando las funcionalidades necesarias, expuestas por la capa de servicios de negocio y devolviendo un *modelo* requerido para ser mostrado.
- **Modelo:** Estos objetos contienen los datos resultantes de la ejecución de la lógica de negocio, los cuales deberían ser mostrados en la respuesta.
- **Vistas:** Son responsables de mostrar el modelo resultante en la respuesta de la petición. La forma de mostrar el modelo podrá ser de diferentes tipos de vistas, por ejemplo, archivos JSP, HTML, PDF, documentos de Excel, etcétera. Las vistas no son responsables de modificar los datos o incluso de obtener los datos; estas simplemente sirven para mostrar los dados del modelo que han sido suministrados por un controlador.

Esta combinación de estos tres objetos, es la aplicación el patrón arquitectónico llamado MVC (Model View Controller).

El cliente interactuará con esta capa directamente utilizando el navegador Web a través de peticiones URL. En el cliente se formará dinámicamente una pequeña capa de presentación que estará formada por código html y JavaScript. Esta capa interactuará directamente mediante peticiones basadas en la tecnología Ajax con los componentes necesarios del servidor creados bajo la tecnología de Spring MVC.

2.4.2. Estructura de paquetes.

2.4.2.1. Organización de la aplicación subsistemas y módulos.

Con el objetivo de organizar la aplicación se definieron como unidades de organización los módulos. Un módulo encapsula un conjunto de funciones que debe realizar el sistema, las cuales son agrupadas por tener características muy similares y se definen en la etapa de diseño.

Un módulo se puede comparar con una pequeña máquina que puede actuar por sí sola, siempre y cuando estén activas las demás máquinas de las que esta depende para su funcionamiento. En un módulo generalmente existen todas las capas lógicas que se definen en la arquitectura base. Cada componente que se desarrolle en un módulo tiene un lugar definido en esta estructura de paquetes.

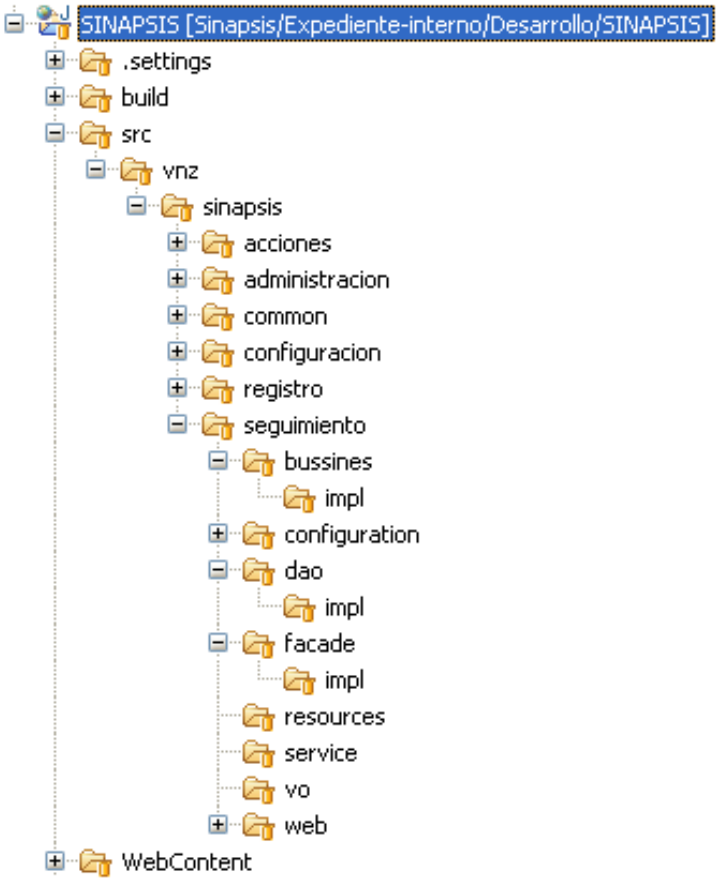

Figura 2.6 Estructura de paquetes

2.4.3. Convenciones o estándares de código y recursos.

La aplicación debe mantener un lenguaje común y uniforme, con este fin se especifican convenciones de nombres y estándares de código para los distintos recursos.

Las clases Java adoptan convenciones estándares presentadas en la Especificación del Lenguaje Java por la compañía Sun Microsystem.

Los nombre para las distintas clases java se definen a través de convenciones de nombres dependiendo de las funciones que tengan cada una de estas en la aplicación:

2.4.3.1. Clases de la capa de Acceso a Datos:

- Las interfaces que representan las operaciones sobre los objetos de acceso a datos, correspondientes al patrón de diseño Data Access Object terminan con la palabra “DAO”. Ejemplo: IndividuoDAO. Las mismas estarán situadas dentro del paquete *dao* correspondiente a cada módulo.
- Las implementaciones de las interfaces DAO comienza con el nombre de la interfaz correspondiente y terminan con el sufijo “Impl”. Ejemplo: IndividuoDAOImpl. Estas clases estarán situadas dentro del paquete *dao.impl* correspondiente a cada módulo.

2.4.3.2. Clases de la capa de Negocio:

- Las interfaces que representan las fachadas que a agrupan las funcionalidades del negocio de los módulos (se basa en el patrón de diseño Facade) terminarán con el sufijo “Facade”. Ejemplo: DatosPersonalesFacade y estarán localizadas en el paquete *facade* correspondiente a cada módulo.
- Las implementaciones de las fachadas comenzarán con el nombre de la interfaz correspondiente y terminaran con el sufijo “Impl”. Ejemplo: DatosPersonalesFacadeImpl y estarán en el paquete *facade.impl* de cada módulo.
- Las interfaces que representan un conjunto de funcionalidades de la lógica de negocio de un módulo terminarán en el sufijo “BO”. Ejemplo: IndiviuoBO y se localizarán en el paquete *bo* de cada módulo.
- Las implementaciones que se le realicen la interfaz de un Manager deben comenzar con el nombre de la interface y terminar en el sufijo “Impl”. Ejemplo: IndividuoBOImpl. Estas clases estarán en el paquete *bo.impl* de cada módulo.

2.4.3.3. Clases de la capa de Presentación:

- Capa Web:
    - Las clases que manejan el flujo de la capa de presentación, es decir, los controladores, terminarán con el sufijo “Controller” y se situarán en el paquete *web* correspondiente a cada módulo.
    - Las clases utilizadas para guardar datos procedentes de las vistas, utilizando el patrón Command definido en los patrones GoF, tendrán un nombre lógico de acuerdo a los datos que contiene seguido del sufijo “Command”. y se situarán en el paquete *web.command* correspondiente a cada módulo.
- Fachadas de Servicios:
    - Las interfaces que representan las operaciones de negocio que se van a exponer o consumir como servicios terminarán en el sufijo “Service”. Ejemplo: IndividuoService, las mismas se pondrán en el paquete *service* de cada módulo.
    - Las implementaciones de las interfaces de servicio que expondrán las funciones de negocio y las implementaciones que se van a consumir de un servicio, comenzarán con el nombre de la interfaz correspondiente y terminarán con el sufijo “Impl”. Ejemplo: IndividuoServiceImpl, y se localizarán en el paquete *service.impl* de cada módulo.

2.4.3.4. Recursos

- Las páginas HTML que constituyen recursos estáticos tendrán la extensión “.html”.
- Las páginas HTML que se mapean como URL en los *Handler Mappings* terminarán en “htm”.
- Las imágenes en formato JPEG que son estáticas en la aplicación terminarán “.jpg”.
- Las imágenes en formato JPEG que son generadas dinámicamente terminarán con la palabra “.jpeg”.

Se definen convenciones de nombre para archivos que tienen que ver con la configuración de la aplicación, los *“Application-Context”* de Spring Framework, archivos utilizados para la internacionalización, ficheros *“.properties”* o cualquier otro archivo de configuración.

Los AppContext tendrán la siguiente estructura y estarán situados en el paquete *configuration* correspondiente a cada módulo:

[sinapsis]-[módulo]-[capa lógica]-context.xml

- [módulo]: indica las unidades organizacionales utilizadas en el proyecto de forma de árbol, pueden ser tantas como la complejidad del mismo lo requiera según las estructuras definidas para cada nivel de complejidad.
- [capa lógica]: representa la capa lógica que maneja del *Application-Context* de acuerdo al tipo de *beans* que se manejan en él. Se establecen las siguientes clasificaciones explicada a continuación:
    - [dao]: Contiene los beans con las funcionales referentes a la capa de acceso a datos.
    - [web]: Encapsula todo los beans de la capa de presentación. Utilizando Spring MVC aquí se configuran componentes como Controladores, Handler Mappings, View Resolvers y todos los utilizados para conformar la lógica de esta capa.
    - [service]: Contiene los beans que funcionarán como servicio y en caso de que se necesite definir lógica para exponer los mismos se definirá en este archivo.
    - [business]: Se refiere a las operaciones de negocio de la aplicación, en este XML se definen las fachadas y otros objetos que representen la capa de servicios.

2.5. Vista de despliegue

2.5.1. Diagrama de despliegue.

El SINAPSIS se encontrará desplegado en la Universidad Bolivariana, sus servicios podrán ser consumidos desde cualquier punto con acceso a Internet y el ancho de banda requerido. En la figura se muestra como quedaría la propuesta de diseño de la infraestructura de conectividad para la aplicación, todos los servidores de aplicación y base de datos van a estar aislados en una subred independiente a la subred principal para brindar una mayor seguridad de los mismos, se conectarán a un switch capa3 (Cisco3550). Sólo serán accesibles fuera de la subred los servidores de aplicación (Balanceadores) por el puerto 80 (http) y 443 (https), lo cual se garantiza en el firewall a nivel de configuración. Los servidores de Base de Datos sólo serán accesibles desde el balanceador de base de datos por un puerto seleccionado. La administración de los servidores se realizará desde una computadora ubicada en la misma subred para evitar puertas traseras con la administración remota o desde internet.

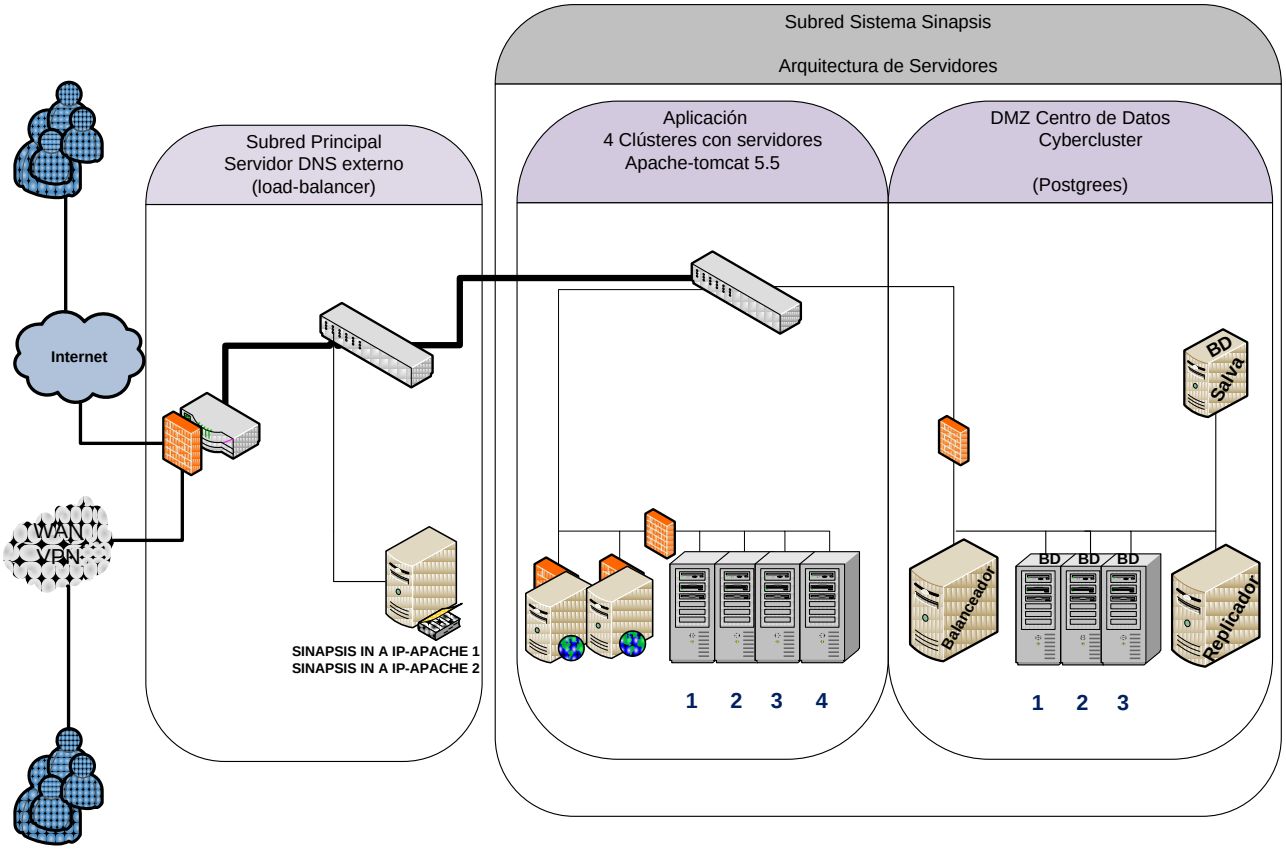

Figura 2.7 Diagrama de despliegue

2.6. Vista de Datos

Para implementar la capa de persistencia, como se explicaba en la descripción general de la arquitectura, se utilizó el framework Hibernate. Hibernate es un potente framework de mapeo objeto/relacional y servicio de consultas para Java. Es la solución ORM más popular en el mundo Java. Permite diseñar objetos persistentes que podrán incluir polimorfismo, relaciones, colecciones, y un gran número de tipos de datos. De una manera muy rápida y optimizada podremos generar bases de datos en cualquiera de los entornos soportados: Postgres, Oracle, DB2, MySQL, etcétera.

2.6.1. Principales características.

- Permite expresar consultas utilizando SQL nativo o consultas basadas en criterios.

- Soporta todos los sistemas gestores de bases de datos SQL y se integra de manera elegante y sin restricciones con los más populares servidores de aplicaciones J2EE y contenedores web, y por supuesto también puede utilizarse en aplicaciones de escritorio.
- Puede operar proporcionando persistencia de una manera transparente para el desarrollador.
- Soporta el paradigma de orientación a objetos de una manera natural: herencia, polimorfismo, composición y el framework de colecciones de Java.
- Soporte para modelos de objetos de forma elegante. Permite una gran variedad de mapeos para colecciones y objetos dependientes.
- Provee un sistema de caché de dos niveles y puede ser usado en un clúster. Permite inicialización perezosa (lazy) de objetos y colecciones.
- Proporciona el lenguaje HQL en cual provee una independencia del SQL de cada base de datos, tanto para el almacenamiento de objetos como para su recuperación.
- Presenta un potente mecanismo de transacciones de aplicación llegando incluso a permitir la interacciones largas (aquellas que requieren la interacción con el usuario durante su ejecución).
- Soporta los diversos tipos de generación de identificadores que proporcionan los sistemas gestores de bases de datos así como generación independiente de la base de datos, incluyendo identificadores asignados por la aplicación o claves compuestas.

2.6.2. Uso de elementos en el gestor de base de datos.

La mayoría de las operaciones en la aplicación se realizarán bajo el framework Hibernate en el modelo objetual. Sin embargo hay casos muy puntuales donde por razones de rendimiento y operatividad se puede llegar a usar tecnologías en el lado del gestor de base de datos como es el caso de Procedimientos Almacenados, Vistas, Triggers, y Vistas Materializadas.

Procedimientos Almacenados: Se utilizarán procedimientos almacenados cuando se deseen realizar grandes cálculos sobre datos que están dispersos en muchas entidades que resulta complejo obtenerlas utilizando Hibernate dado el costo de procesamiento que utilizaría el framework para obtener los mismos y después realizar el cálculo en el lado del servidor de aplicaciones. Estos datos no son consultados con mucha frecuencia.

Vistas: Se utilizan en casos muy parecidos a los anteriores pero en aquellos casos donde no se realicen cálculos sobre los datos. Es decir, se utilizaría solamente cuando se quiere centralizar datos

que están dispersos en muchas entidades y que sería muy costoso para Hibernate generar las consultas para obtenerlos.

Triggers: Se utilizan solamente para actualizar datos referentes a la visibilidad de los documentos y en los casos en que se determine realizar acciones de optimización o limpieza de los datos en la base de datos. Cualquier otra operación de este tipo será manejada desde la aplicación.

2.7. Seguridad

2.7.1. Autenticación y permisos de acceso

La aplicación contará con un sistema de seguridad basado en usuarios los cuales van a contener roles. La aplicación permitirá definir roles que abarquen una o varias funcionalidades que estarán definidas. Cada funcionalidad representará en el sistema una acción atómica, ej. Agregar Documento, Eliminar Ficha, etc. Por tanto al crear usuarios y asociarles a los mismos un conjunto de roles, se definen las funcionalidades a las cuales dicho usuario tendrá acceso. La autenticación de los usuarios se realizará utilizando nombre de usuario y contraseña, esta información se almacenará en la base de datos al igual que los roles. También existirán grupos de usuarios los cuales podrán ser definidos, los mismo van a contener un conjunto de roles, los usuarios que pertenezcan a un grupo de usuario se le asignarán los roles que contiene dicho grupo.

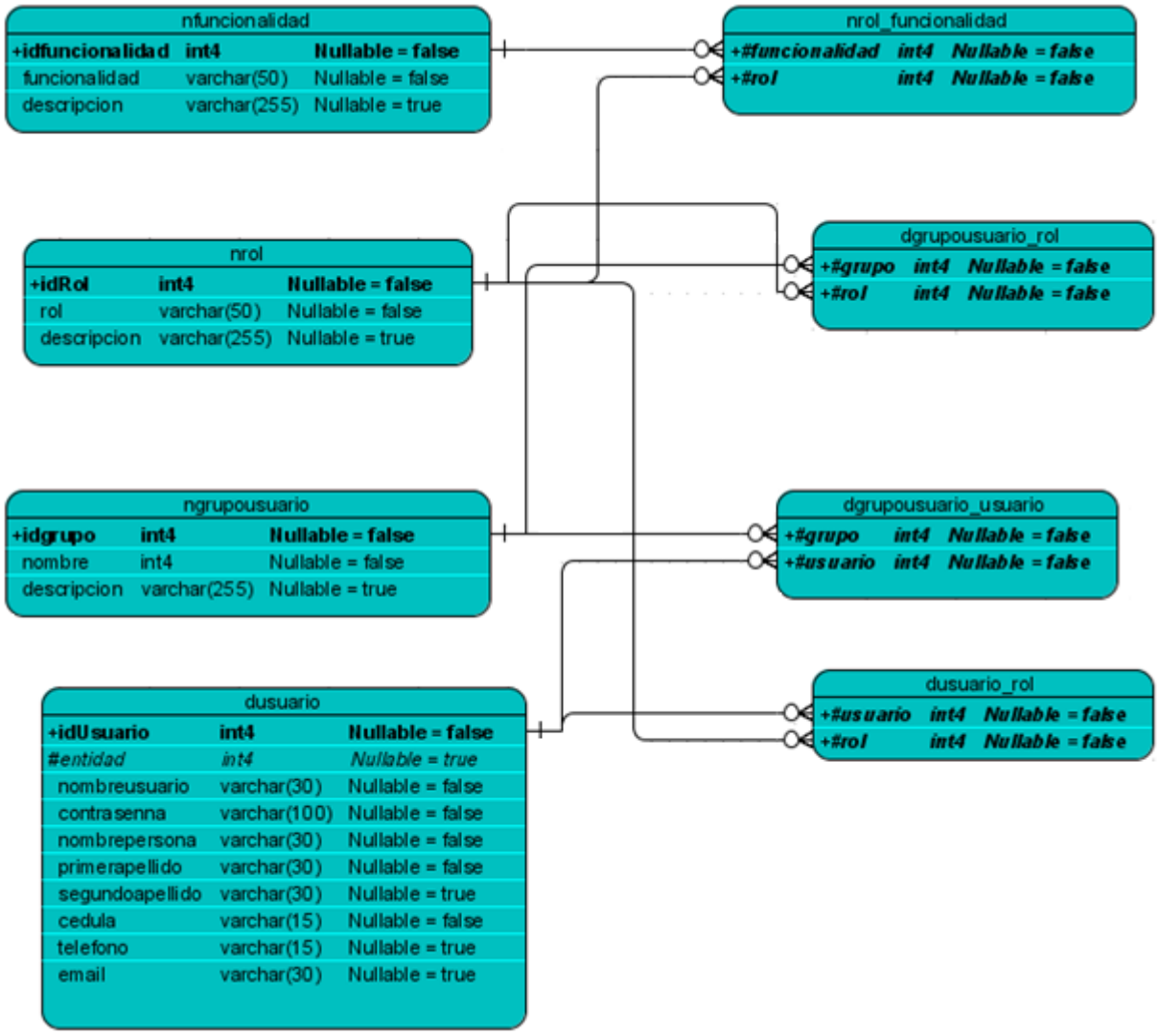

Figura 2.8 Vista del modelo de datos de autenticación

Para manejar todo este sistema de autenticación y permisos de acceso, se utilizará el framework Acegi Security contenido dentro del framework Spring (Spring Secutity). Este framework es uno de los más utilizados en la plataforma Java, a diferencia de otros mecanismos de seguridad existentes en java como JAAS, este framework al venir integrado con Spring se beneficia de su contenedor de objetos y su modulo de AOP (Programación Orientada a Aspectos) permitiendo realizar el manejo de la seguridad desde ficheros de configuración y de manera transparente al resto de la aplicación. Acegi contiene un manejador de autenticación que permite obtener la información de cualquier sistema sea una base de datos un servidor LDAP, etc. Además realiza todo el chequeo de las

credenciales incluye encriptación de contraseñas (MD5, SHA) y almacena los datos del usuario con sus roles en memoria para realizar el chequeo a la hora de acceder a los recursos seguros.

Este framework permite realizar el chequeo de permisos a nivel de filtros de URL. Acegi permite llevar esto a la aplicación de forma no intrusiva, utilizando ficheros de configuración, basta con definir utilizando expresiones regulares las URL y los roles que debe tener un usuario para acceder a cada una de ellas.

2.7.2. Seguridad en el envío de información sensible

Para el envío de información sensible hacia el servidor se utilizará SSL (HTTPs) la variante más utilizada por las aplicaciones Web, en conjunto con el framework Acegi el cual permite definir las URL que deben ser invocadas utilizando HTTPs, ocupándose del intercambio HTTP-HTTPs.

2.7.3. Manejo de las contraseñas

La aplicación exigirá un nivel adecuado en las contraseñas de los usuarios, las mismas deben ser mayores de 7 caracteres y deben contener letras, números y símbolos especiales. Las contraseñas deben ser renovadas cada cierto periodo de tiempo (2 meses).

2.7.4. Manejo de la sesión de usuario

La aplicación no permitirá más de una sesión por usuario abierta en el sistema. La sesión de usuario caducará luego de un tiempo prudencial de inactividad en el sistema (15 min).

2.7.5. Validaciones

Las validaciones de todos los datos de entrada se harán en el servidor utilizando el sistema de validaciones que tiene el framework Spring basado en expresiones regulares, aunque no quiere decir que se puedan realizar adicionalmente en el cliente utilizando lenguajes de scripts. Además de realizar las validaciones que garantizan el sentido de los datos con respecto al negocio se deben discriminar en todas la entrada de caracteres especiales principalmente: comillas simples (‘ ‘), comillas dobles (“ “), paréntesis angulares (< >), asterisco (*), y cualquier otro que no tenga sentido para la lógica de la aplicación con el objetivo de evitar la inyección de código (HTML, Script, SQL).

2.7.6. Captcha

El sistema contará con un mecanismo de Captcha para evitar las autenticaciones de programas que puedan forzar la entrada al sistema. Para ello se utilizará el framework JCaptcha, el cual provee de todo este mecanismo y además el mismo se integra con Spring Security para así complementar un sistema completo de seguridad a nivel de aplicación totalmente manejados a nivel de ficheros de configuración.

2.7.7. Trazas

Se implementará un sistema de trazas que permita registrar las acciones así como el usuario que las realizó y en qué momento, para ello se utilizará programación orientada a aspectos lo cual permitirá la implementación del mismo de forma transparente y no intrusiva en la lógica de la aplicación.

2.7.8. Seguridad en la infraestructura

Se propone la implantación de cortafuegos a nivel de servidores y de dispositivos de redes (routers, switch, etc). Estos cortafuegos deben estar restringidos permitiendo la comunicación sólo con los puertos específicos, en el caso del cortafuegos para el servidor Apache permitiría la comunicación sólo por los puertos 80 y 443 para la comunicación SSL, en el caso de los cortafuegos para los servidores Apache TomCat permitirían la comunicación sólo por los puertos 8009 para comunicarse con los demás TomCat y el 22 para la administración de los mismos restringiéndose también las PC destinadas para esta acción vía MAC o IP. En el caso de los servidores de bases de datos sus cortafuegos estarían configurados para permitir la comunicación a través del puerto 5432 y con restricciones vía IP de las PC destinadas para soportar el sistema.

2.8. Integración con sistemas legados

La integración con los sistemas legados debe realizarse a través de un grupo de interfaces desarrolladas sobre servicios web utilizando WSDL 1.1 como lenguaje estándar para la descripción de estos servicios y la creación de los contratos, garantizando el acceso dinámico a los mismos.

Estos servicios implementaran básicamente dos tipos de operaciones, operaciones Unidireccionales y de Petición/Respuesta.

Unidireccional: El Servicio recibe un mensaje y no genera ninguna respuesta.

Petición / Respuesta: El Servicio recibe un mensaje y responde con otro.

2.8.1. Características que deben cumplir los servicios

Los Servicios deben ser reusables: Todo servicio debe ser diseñado y construido pensando en su reutilización dentro de la misma aplicación, dentro del dominio de aplicaciones de la empresa o incluso dentro del dominio público para su uso masivo.

Los Servicios deben proporcionar un contrato formal: Todo servicio desarrollado, debe proporcionar un contrato en el cual figuren: el nombre del servicio, su forma de acceso, las funcionales que ofrece, los datos de entrada de cada una de las funcionalidades y los datos de salida. De esta manera, todo consumidor del servicio, accederá a este mediante el contrato, logrando así la independencia entre el consumidor y la implementación del propio servicio. En el caso de los Servicios Web, esto se logrará mediante la definición de interfaces con WSDL.

Los Servicios deben tener bajo acoplamiento: Es decir, que los servicios tienen que ser independientes los unos de los otros. Para lograr ese bajo acoplamiento, lo que se hará es que cada vez que se vaya a ejecutar un servicio, se accederá a él a través del contrato, logrando así la independencia entre el servicio que se va a ejecutar y el que lo llama. Si conseguimos este bajo acoplamiento, entonces los servicios podrán ser totalmente reutilizables.

Los Servicios deben de ser autónomos: Todo servicio debe tener su propio entorno de ejecución. De esta manera el servicio es totalmente independiente y nos podemos asegurar que así podrá ser reutilizable desde el punto de vista de la plataforma de ejecución.

Los Servicios no deben tener estado: Un servicio no debe guardar ningún tipo de información. Esto es así porque una aplicación está formada por un conjunto de servicios, lo que implica que si un servicio almacena algún tipo de información, se pueden producir problemas de inconsistencia de datos. La solución, es que un servicio sólo contenga lógica, y que toda información esté almacenada en algún sistema de información sea del tipo que sea.

2.8.2. Librerías para la creación de los servicios.

Para la creación de los servicios se propone la utilización de un grupo de librerías libres que permiten la generación de servicios web desde el propio IDE Eclipse 3.2.

- axis.jar
- commons-discovery.jar
- commons-logging.jar
- jaxrpc.jar
- log4j-1.2.8.jar

- saaj.jar
- wsdl4j.jar

Con el objetivo de permitir la comunicación segura de la información que se intercambia entre los servicios a desarrollar proponemos la implantación de un certificado digital y la restricción vía IP o MAC de las máquinas que contendrán dichos servicios.

2.9. Conclusiones parciales

En este capítulo se elaboró la propuesta de solución del sistema mediante la línea base de la arquitectura. Se definieron los requisitos no funcionales del sistema, es decir, las herramientas y tecnologías necesarias para desarrollar la aplicación, además se expusieron las diferentes vistas planteadas por RUP, imprescindibles para la comprensión y el desarrollo del sistema. Vimos en este capítulo además las principales características de las herramientas de desarrollo del software y de los diferentes frameworks de desarrollo que se utilizarán.

CAPÍTULO 3: EVALUACIÓN DE LA ARQUITECTURA

La arquitectura de software es una parte fundamental para el desarrollo de cualquier aplicación. Es conveniente, antes de aplicar cualquier arquitectura desarrollada, evaluarla para asegurar que tiene la calidad requerida, pues en torno a esta es que se estará desarrollando toda la aplicación.

En este capítulo se realiza un análisis de diferentes aspectos a tener en cuenta a la hora de realizar la evaluación de una arquitectura de software como son: la etapa en que se puede evaluar una arquitectura, los atributos de calidad a evaluar, los métodos de evaluación; se realizará luego una evaluación basada en el método seleccionado y se hará una valoración de los resultados alcanzados.

3.1. Importancia de evaluar la arquitectura de software

Evaluar una arquitectura de software sirve para prevenir todos los posibles desastres de un diseño que no cumple con los requerimientos de calidad y para saber que tan adecuada es para el sistema que se quiere construir. La evaluación de una arquitectura de software no brinda una respuesta concreta sobre si la misma es buena o mala, o una calificación determinada en la que basarse para tomar una decisión; sino que resalta dónde es que se encuentra el riesgo, es decir fortalezas y debilidades identificadas en la misma. Después de evaluar una arquitectura de software se pueden tomar algunas decisiones sobre si se puede seguir con el proyecto, si hay que reforzar algunos puntos de la arquitectura o si hay que comenzarla toda de nuevo. (43)

3.2. Etapas de evaluación de la arquitectura de software

Una buena regla para decidir cuándo hay que realizar una evaluación de una arquitectura es cuando el equipo de desarrollo comience a tomar decisiones que dependan de la arquitectura y el costo de deshacer dichas decisiones sea mayor al costo de realizar la evaluación. (43)

Dos variantes de este tipo de evaluación son la evaluación temprana y la evaluación tardía. (44)

- *Evaluación temprana*: No es necesario que la arquitectura esté completamente especificada para efectuar la evaluación. Abarca desde las fases tempranas de diseño y a lo largo del desarrollo. Es posible tomar decisiones sobre la arquitectura a cualquier nivel, puesto que pueden aparecer distintos tipos de cambios arquitectónicos, producto de una evaluación en función de los atributos de calidad esperados. (44)
- *Evaluación tardía*: La evaluación de la arquitectura se realiza cuando esta se encuentra establecida y la implementación completada. Este es el caso general que se presenta al momento de la adquisición de un sistema ya desarrollado. Se considera muy útil la

evaluación del sistema en este punto, puesto que puede observarse el cumplimiento de los atributos de calidad asociados al sistema y como será su comportamiento general. (44)

3.3. Atributos de calidad

Los factores o atributos que afectan a la calidad del software se pueden categorizar en dos amplios grupos: factores que se pueden medir directamente y factores que se pueden medir sólo indirectamente. En todos los casos debe aparecer la medición. El software debe ser comparado con una referencia y llegar a una conclusión sobre la calidad. (13)

- Operacionales u observables en vía ejecución: Aquellos atributos que se determinan del comportamiento del sistema en tiempo de ejecución. Estos describen los niveles de comportamiento del sistema durante la ejecución, entrega de los resultados esperados, velocidad y precisión de la respuesta, comportamiento durante la interacción con otros sistemas, entre otros.
- De desarrollo o no observables en vía ejecución: Aquellos atributos que se establecen durante el desarrollo del sistema. Son los atributos que de cierta manera son relevantes para la ingeniería de software, tratan los aspectos de facilidad de integrar el sistema, modificarlo o probarlo; el costo y el tiempo del desarrollo, etc.

Como SINAPSIS no se ha implementado, para la evaluación de su arquitectura solamente se pueden medir los atributos de calidad no observables en vía ejecución. A continuación se analizan las definiciones de algunos de ellos (44):

- *Configurabilidad*: posibilidad que se otorga a un usuario experto a realizar ciertos cambios al sistema.
- *Integrabilidad*: es la medida en que trabajan correctamente los componentes del sistema que fueron desarrollados separadamente al ser integrados.
- *Integridad*: es la ausencia de alteraciones inapropiadas de la información.
- *Interoperabilidad*: es la medida de la habilidad de que un grupo de partes del sistema trabajen con otro sistema. Un tipo especial de integrabilidad.
- *Modificabilidad*: es la habilidad de realizar cambios futuros al sistema.
- *Mantenibilidad*: es la capacidad de someter a un sistema a reapariciones y evolución. Capacidad de modificar el sistema de manera rápida y a bajo costo.
- *Portabilidad*: es la habilidad del sistema para ser ejecutado en diferentes ambientes de computación. Estos ambientes pueden ser hardware, software o una combinación de los dos.

- *Reusabilidad*: es la capacidad de diseñar un sistema de forma tal que su estructura o parte de sus componentes puedan ser reutilizados en futuras aplicaciones.
- *Escalabilidad*: es el grado con el que se puede ampliar el diseño arquitectónico, de datos o procedimental.
- *Capacidad de prueba*: es la medida de la facilidad con la que el software, al ser sometido a una serie de pruebas, puede demostrar sus fallas. Es la probabilidad de que, asumiendo que tiene al menos una falla, el software fallará en su próxima ejecución de prueba.

3.4. Métodos de evaluación

Hasta hace poco no existían métodos de utilidad general para evaluar arquitecturas de software. Si alguno existía, sus enfoques eran incompletos y no repetibles, lo que no brindaba mucha confianza. En virtud de esto, múltiples métodos de evaluación han sido propuestos (46). Un método de evaluación sirve de guía a los involucrados en el desarrollo del sistema para la búsqueda de conflictos que puede presentar una arquitectura y sus soluciones (44). A continuación se explican algunos de los más importantes.

Software Architecture Analysis Method

El Método de Análisis de Arquitecturas de Software (SAAM) es el primero que fue ampliamente promulgado y documentado. El método fue originalmente creado para el análisis de la modificabilidad de una arquitectura, pero en la práctica ha demostrado ser muy útil para evaluar de forma rápida distintos atributos de calidad, tales como modificabilidad, portabilidad, escalabilidad e integrabilidad. (44) (45) (46)

El método de evaluación SAAM se enfoca en la enumeración de un conjunto de escenarios que representan los cambios probables a los que estará sometido el sistema en el futuro. Como entrada principal, es necesaria alguna forma de descripción de la arquitectura a ser evaluada. Las salidas de la evaluación del método SAAM son las siguientes: (44) (45) (46)

- Una proyección sobre la arquitectura de los escenarios que representan los cambios posibles ante los que puede estar expuesto el sistema.
- Entendimiento de la funcionalidad del sistema, e incluso una comparación de múltiples arquitecturas con respecto al nivel de funcionalidad que cada una soporta sin modificación.

Con la aplicación de este método, si el objetivo de la evaluación es una sola arquitectura, se obtienen los lugares en los que la misma puede fallar, en términos de los requerimientos de modificabilidad. Para el caso en el que se cuenta con varias arquitecturas candidatas, el método produce una escala relativa que permite observar qué opción satisface mejor los requerimientos de calidad con la menor cantidad de modificaciones. (44) (45) (46)

Metodología SAAM (44) (45) (46)

El método de evaluación SAAM comprende 6 pasos:

Paso 1: Desarrollo de escenarios.

Paso 2: Descripción de la arquitectura.

Paso 3: Clasificación y asignación de prioridades de los escenarios.

Paso 4: Evaluación individual de los escenarios indirectos.

Paso 5: Evaluación de la interacción entre escenarios.

Paso 6: Creación de la evaluación global.

Architecture Trade-off Analysis Method

El Método de Análisis de Acuerdos de Arquitectura (ATAM) está inspirado en tres áreas distintas: los estilos arquitectónicos, el análisis de atributos de calidad y el método de evaluación SAAM. El método ATAM revela la forma en que una arquitectura específica satisface ciertos atributos de calidad, y provee una visión de cómo los atributos de calidad interactúan con otros; esto es, los tipos de acuerdos que se establecen entre ellos. (44) (45) (46)

El método se concentra en la identificación de los estilos arquitectónicos o enfoques arquitectónicos utilizados. Estos elementos representan los medios empleados por la arquitectura para alcanzar los atributos de calidad, así como también permiten describir la forma en la que el sistema puede crecer, responder a cambios, integrarse con otros sistemas, entre otros. (44) (45) (46)

Metodología ATAM (44) (45) (46)

El método de evaluación ATAM comprende 9 pasos, agrupados en 4 fases:

Fase 1: Presentación

Paso 1: Presentación del ATAM.

Paso 2: Presentación de las metas.

Paso 3: Presentación de la arquitectura.

Fase 2: Investigación

Paso 4: Identificación de los enfoques arquitectónicos.

Paso 5: Generación del Utility Tree.

Paso 6: Análisis de los enfoques arquitectónicos.

Fase 3: Pruebas

Paso 7: Lluvia de ideas y establecimiento de prioridad de escenarios.

Paso 8: Análisis de los enfoques arquitectónicos.

Fase 4: Reporte

Paso 9: Presentación de los resultados.

Active Reviews for Intermediate Designs

El Método de Análisis de Diseños Intermedios (ARID) es conveniente para realizar la evaluación de diseños parciales en las etapas tempranas del desarrollo. En ocasiones, es necesario saber si un diseño propuesto es conveniente, desde el punto de vista de otras partes de la arquitectura. ARID es un híbrido entre Active Design Review (ADR) y ATAM: en el caso de ADR, los involucrados reciben documentación detallada y completan cuestionarios, cada uno por separado; y en el caso de ATAM, está orientado a la evaluación de toda una arquitectura. (44) (45) (46)

De ADR, resulta conveniente la fidelidad de las respuestas que se obtiene de los involucrados en el desarrollo. Así mismo, la idea del uso de escenarios generados por los involucrados con el sistema es tomada del ATAM. De la combinación de ambas filosofías surge ARID, para efecto de la evaluación temprana de los diseños de una arquitectura de software. (44) (45) (46)

Metodología ARID (44) (45) (46)

El método de evaluación ARID comprende 9 pasos, agrupados en 2 fases:

Fase 1: Actividades Previas

Paso 1: Identificación de los encargados de la revisión.

Paso 2: Preparar el informe de diseño.

Paso 3: Preparar los escenarios base.

Paso 4: Preparar los materiales.

Fase 2: Revisión.

Paso 5: Presentación del ARID.

Paso 6: Presentación del diseño.

Paso 7: Lluvia de ideas y establecimiento de prioridades de escenarios.

Paso 8: Aplicación de los escenarios.

Paso 9: Resumen.

Modelo de Negociación WinWin

El Modelo WinWin provee un marco de referencia general para identificar y resolver conflictos de requerimientos, mediante la elicitación y negociación de artefactos en función de las condiciones de ganancia. El modelo utiliza la teoría “W”, que pretende que todo involucrado salga ganador. De esta forma, asiste a los involucrados en el desarrollo a identificar y negociar distintos aspectos, reconciliando conflictos entre las opciones de ganancias para todos. (46)

Aunque el modelo propone la resolución de posibles conflictos, no siempre es posible llegar a un acuerdo. En este sentido, el método CBAM provee medios para establecer los balances necesarios, y un marco de referencia para la discusión que puede llevar a una posible solución del problema. (46)

Cost-Benefit Analysis Method

El Método de Análisis de Costos y Beneficios (CBAM) es un marco de referencia que no toma decisiones por los involucrados en el desarrollo del sistema. Por el contrario, ayuda en la elicitación y documentación de los costos, beneficios e incertidumbre, y provee un proceso de toma de decisiones racional. Uno de los elementos que introduce el método son las llamadas estrategias arquitectónicas, que consisten en posibles opciones para la resolución de conflictos entre atributos de calidad presentes en una arquitectura. (46)

El modelo de referencia para evaluación de arquitecturas de software para este método de evaluación comienza con el método WinWin, para efectos de la elicitación de las necesidades de los involucrados y la exploración de las opciones de resolución de conflictos. (46)

Metodología CBAM (46)

Paso 1: Selección de escenarios.

Paso 2: Identificación de los conflictos entre atributos de calidad.

Paso 3: Exploración de las opciones en busca de la solución de conflictos.

Paso 4: Medición de los beneficios de los atributos de calidad.

Paso 5: Cuantificación de los beneficios.

Paso 6: Cuantificación de costos e implicaciones de calendario.

Paso 7: Cálculo del nivel de deseabilidad.

Paso 8: Alcanzar un acuerdo.

Método Diseño y Uso de Arquitecturas de Software propuesto por Bosch

El Método de Diseño de Arquitecturas de Software propone que el proceso de evaluación debe ser visto como una actividad iterativa, que forma parte del proceso de diseño, también iterativo. Una vez que la arquitectura es evaluada, pasa a una fase de transformación, asumiendo que no satisface todos los requerimientos. Luego, la arquitectura transformada es evaluada de nuevo. (46)

Metodología del método propuesto por Bosch (46)

El proceso de evaluación propuesto por Bosch se divide en dos etapas:

Etapa 1.

Paso 1: Selección de atributos de calidad.

Paso 2: Definición de los perfiles.

Paso 3: Selección de una técnica de evaluación.

Etapa 2.

Paso 4: Ejecución de la evaluación.

Paso 5: Obtención de resultados.

Método de comparación de arquitecturas basada en el modelo ISO/IEC 9126 adaptado para arquitecturas de software

La especificación de los atributos de calidad haciendo uso de un modelo basado en estándares internacionales ofrece una vista amplia y global de los atributos de calidad, tanto a usuarios como arquitectos del sistema, para efectos de la evaluación. (46)

Metodología del método basado en el modelo ISO/IEC 9126 (46)

Paso 1: Analizar los requerimientos funcionales y no funcionales principales del sistema, para establecer las metas de calidad.

Paso 2: Utilizar el modelo de calidad ISO/IEC 9126 adaptado para arquitecturas de software. Algunas métricas pueden definirse con mayor nivel de detalle.

Paso 3: Presentar las arquitecturas candidatas iniciales.

Paso 4: Construir la tabla comparativa para las arquitecturas candidatas.

Paso 5: Establecer prioridades para las subcaracterísticas de calidad tomando en cuenta los requerimientos de calidad del sistema.

Paso 6: Analizar los resultados obtenidos y resumidos en la tabla, de acuerdo con las prioridades establecidas.

3.4.1. Método de evaluación seleccionado

Un método de evaluación no es mejor que otro, sino que evalúa mejor, en ciertas condiciones, un atributo de calidad dado. El método de evaluación seleccionado está dado por las condiciones en las que está el proyecto y qué es lo que se desea evaluar del mismo.

El desarrollo de SINAPSIS se encuentra en una etapa prematura, por lo que el método que se utilizará es el ARID, el cual es más conveniente para realizar la evaluación de diseños parciales en etapas tempranas del desarrollo.

Entre las ventajas del uso del método ARID están:

- Facilidad de usar.
- Aplicación poco costosa y de gran beneficio
- Mejor evaluación de la factibilidad de la arquitectura.

El método ARID está basado en la técnica de evaluación por escenarios, los cuales describen la interacción de alguno de los involucrados en el desarrollo del sistema con este. (44) (46)

Un escenario está compuesto por tres partes: (44) (46)

- *El estímulo*: Parte del escenario que explica o describe lo que el involucrado en el desarrollo hace para iniciar la interacción con el sistema.
- *El contexto*: Describe qué sucede en el sistema al momento del estímulo.
- *La respuesta*: Describe, a través de la arquitectura, cómo debería responder el sistema ante el estímulo. Permite establecer cuál es el atributo de calidad asociado.

Las ventajas del uso de escenarios son: (44) (46)

- Simples de crear y entender.
- Poco costosos y no requieren mucho entendimiento.
- Efectivos.

Las técnicas de evaluación basadas en escenarios cuentan con dos instrumentos de evaluación relevantes, a saber: el *Utility Tree* y los *Profiles* (Perfiles). (44) (46)

Se hará uso de la técnica junto con los *Profiles*, los cuales son un conjunto de escenarios, generalmente con alguna importancia relativa asociada a cada uno de estos. El uso de los *Profiles* permite hacer especificaciones más precisas del requerimiento para un atributo de calidad. (46)

3.5. Evaluación de la arquitectura

La propuesta de arquitectura se encuentra representada en un alto nivel, o sea, muy poco detallada. La selección de los escenarios concretos para validar que la arquitectura cumpliese con requisitos específicos se realizó en base a las cualidades o capacidades que la arquitectura debe cumplir hasta el estado en que se encuentra desarrollada.

Escenario: Seguridad del sistema.

Perfil: Seguridad.

Atributo de calidad: Seguridad.

Relación atributo-escenario: La seguridad del sistema será gestionada por el framework Acegi que viene integrado con el framework Spring, para el cual se definirán las entradas a los recursos al sistema para que luego estas sean asignadas a los usuarios. Un usuario no podrá acceder a un recurso que no le sea asignado. No se permitirá a un usuario tener más de una sesión en el servidor. Al tercer intento de autenticación fallida se exigirá la autenticación junto con un código captcha, para así evitar la intromisión al sistema por fuerza bruta.

Escenario: Intercambio de información.

Perfil: Seguridad.

Atributo de calidad: Integridad.

Relación atributo-escenario: Se utilizará el framework Acegi de Spring, el cual configura la protección del canal por donde fluyen los datos. Mediante la utilización de protocolo seguro SSL se cifra el canal por donde fluyen los datos, asegurando que la información no sea alterada.

Escenario: Cambio en los requisitos.

Perfil: Mantenimiento.

Atributo de calidad: Mantenibilidad.

Relación atributo-escenario: Al ser una aplicación web solamente es necesario realizar cambios en la aplicación que se encuentra en el servidor y no en los clientes. Los cambios realizados a las clases implementadas están en dependencia de la magnitud de los cambios en los requisitos; la utilización del framework Spring, aplicando la técnica de inyección de instancias, independiza a las clases de las implementaciones de sus relaciones, lo que provee un bajo acoplamiento, necesitando solamente realizar cambios en la clase que es necesario modificar y no en las que guardan relación con esta. Las clases en las capas lógicas se relacionan solamente entre ellas y las clases de la capa

inmediatamente inferior; así un cambio en una clase del negocio, por ejemplo, no afectará a las clases de la capa de acceso a datos.

Escenario: Ampliación del medio de persistencia de datos.

Perfil: Ampliación.

Atributo de calidad: Escalabilidad.

Relación atributo-escenario: El gestor de base de datos PostgreSQL permite la creación de clúster para ampliar el espacio de almacenamiento de la información.

Escenario: Incremento de la cantidad estimada de usuarios conectados.

Perfil: Ampliación.

Atributo de calidad: Escalabilidad.

Relación atributo-escenario: La aplicación web estará ejecutándose en el contenedor web Tomcat versión 5.5. El contenedor Tomcat permite el manejo del pull de conexiones de una manera fácil, a través de configuraciones en un XML. En caso de que el servidor no sea capaz de suplir la cantidad de conexiones estimadas, el Tomcat, integrado junto al Apache, permite lograr la escalabilidad mediante clusterización.

Escenario: Migración de sistema operativo.

Perfil: Portabilidad.

Atributo de calidad: Portabilidad.

Relación atributo-escenario: El sistema será desarrollado utilizando como lenguaje Java, el entorno de ejecución será el contenedor web Tomcat, para la persistencia de datos se utilizará el sistema gestor de base de datos PostgreSQL y para la interfaz de usuario, al ser una aplicación web, se utilizará para su desarrollo lenguaje HTML y JavaScript. Cada uno de los elementos seleccionados para el desarrollo de la aplicación son compatibles con la mayoría de los sistemas operativos más populares; la migración, de Tomcat y PostgreSQL, puede realizarse ambos para un mismo sistema operativo, o cada elemento en un sistema operativo distinto sin que afecte el rendimiento; mientras que para la capa de interfaz de usuario solamente requerirá de un navegador web utilizado por el cliente.

Escenario: Gestionar usuarios.

Perfil: Uso.

Atributo de calidad: Configurabilidad.

Relación atributo-escenario: El sistema permitirá al momento de ser desplegado el manejo de los usuarios por parte de un administrador central, teniendo este permisos para adicionar, modificar y eliminar usuarios, gestionando además los recursos a ser asignados a cada uno. El administrador central, o los administradores de niveles inferiores, podrán además designar nuevos administradores en el mismo nivel o a niveles más inferiores. La asignación de las funcionalidades se manejará por roles y por grupos de roles, permitiendo una gestión de usuario más cómoda y más humana a los administradores.

En la evaluación realizada por perfiles se pudo observar la relación que existe entre algunos atributos de calidad y algunos de los escenarios que se pueden presentar en la vida útil del sistema. Algunos atributos del sistema no pudieron ser evaluados debido al estado de desarrollo en que se encuentra el mismo. A medida que se siga desarrollando el sistema otros atributos de calidad podrán ser evaluados para ver cómo responde el sistema a los mismos, y, en caso de que surja algún riesgo, mitigar los posibles problemas que puedan ir apareciendo.

3.6. Conclusiones parciales.

La evaluación de la arquitectura es de vital importancia para lograr un producto de calidad y que pueda ser aceptado por los clientes. Evaluar la arquitectura ayuda a prevenir los futuros riesgos que podría tener una aplicación y, en consideración de los mismos, tomar una decisión. De los principales métodos existentes para la evaluación de arquitecturas de software se seleccionó el ARID, debido a que el sistema se encuentra en etapas tempranas del desarrollo. Una vez que se evaluó la arquitectura se pudo observar que esta cumple de forma satisfactoria con los atributos de calidad mediante los que fue evaluada: seguridad, integridad, mantenibilidad, escalabilidad, portabilidad y configurabilidad. Aunque todavía quedan atributos de calidad por medir en correspondencia con el avanza el sistema, la arquitectura de software propuesta puede ser aplicada para el desarrollo del proyecto SINAPSIS.

CONCLUSIONES

Se realizó un estudio sobre las principales metodologías de desarrollo seleccionando RUP como la más indicada para guiar el proceso de desarrollo de la aplicación. Se realizó un análisis del estado del arte de las principales tecnologías existentes para la plataforma JEE, seleccionando cuales fueron utilizadas para el desarrollo del sistema. Con las tecnologías seleccionadas se realizó la propuesta de un modelo de arquitectura a ser aplicado en el sistema SINAPSIS, cumpliendo con los patrones y características correspondientes al modelo seleccionado, que fue conformado en el documento de arquitectura donde quedaron plasmados los aspectos fundamentales para el desarrollo del sistema. Al término de la investigación se realizó un estudio sobre los principales métodos utilizados para la evaluación de las arquitecturas de software, siendo el ARID el más indicado para evaluar la arquitectura propuesta. La evaluación de la propuesta de arquitectura arrojó que la misma contribuye a que el sistema sea capaz de cumplir con los requerimientos del cliente.

RECOMENDACIONES

Luego de haber concluido la investigación se recomienda la aplicación de la arquitectura para el desarrollo de SINAPSIS. En cada etapa de desarrollo se debe realizar una medición de la arquitectura para ver si esta es capaz de soportar las funcionalidades de los distintos módulos así como mitigar los posibles riesgos que puedan surgir. Ver los puntos de integración que pudiese tener este sistema con otras aplicaciones de gestión gubernamental en Venezuela para que se retroalimenten de la información que manejan cada una de ellas en vez de duplicarla en la nueva aplicación.

BIBLIOGRAFÍA

1. Abrahamsson, Pekka, y otros. Agile Software Development Methods. Review and Analysis. [En línea] 2002. http://www.inf.vtt.fi/pdf/publications/2002/P478.pdf. ISBN 951-38-6010-8.
2. Molpeceres, Alberto. Procesos de desarrollo: RUP, XP y FDD. [En línea] 15 de Diciembre de 2002. http://www.javahispano.org/contenidos.downloadatt.action?id=71.
3. Palacio, Juan. Gestión y Modelos para la Eficiencia en Empresas de Desarrollo de Software. [En línea] http://www.baquia.com/marketing/Gestion_y_modelos_eficiencia_software.pdf.
4. Aguilar, Catherine. Aplicación de Conceptos de Gestión de Proyectos y Gestión de Riesgos en el Desarrollo de Productos Nuevos en el Campo de la Tecnología de Información. Universidad de Puerto Rico : s.n., Diciembre 2005.
5. Jeffries, Ron, Anderson, Ann y Hendrickson, Chet. *Extreme Programming Installed.* s.l. : Addison Wesley, Octubre 2000. ISBN 0-201-70842-6.
6. Marches, Michele, y otros. *Extreme Programming Perspectives.* s.l. : Addison Wesley, Agosto 2002. ISBN 0-201-77005-9.
7. Beck, Kent. *Extreme Programming Explained.* s.l. : Addison Wesley, Septiembre 1999. ISBN 0201616416.
8. Smith, John. A comparison of RUP and XP. [En línea] 2001. ftp://ftp.software.ibm.com/software/rational/web/whitepapers/2003/TP167.pdf.
9. Crispin, Lisa y House, Tip. *Testing Extreme Programming.* s.l. : Addison Wesley, Octubre 2002. ISBN 0-321-11355-1.
10. Kruchten, Philippe. *The Rational Unified Process An Introduction, Second Edition.* s.l. : Addison Wesley, Marzo 2000. ISBN 0-201-70710-1.
11. Kroll, Per y Kruchten, Philippe. *The Rational Unified Process Made Easy: A Practitioner's Guide to the RUP.* s.l. : Addison Wesley, Abril 2003. ISBN 0-321-16609-4.
12. Rational Software Corporation. Rational Unified Process. 2003.
13. Pressman, Roger S. *Ingeniería del Software. Un Enfoque Práctico.* s.l. : Quinta Edición, 2002.
14. White, Sharon A. *The Software Architecture Process.* s.l. : Houston TX, 1997.
15. Cristiá, Maximiliano. *Introducción a la Arquitectura de Software.* Rosario : Universidad Nacional de Rosario, 2007.
16. Jacobson, Ivar, Booch, Grady y Rumbaugh, James. *El Proceso Unificado de Desarrollo de Software.* Madrid, España : Addison Wesley, 2000. ISBN 84-7829-036-2.
17. Alva Pantaleón, Edwin Saúl. *Tópicos avanzados en ingeniería de sistemas II. Desarrollo de aplicaciones Web bajo un arquitectura N Capas.* 2006.

18. Hurwitz, Judith, y otros. *Service Oriented Architecture for Dummies.* Indianapolis, Indiana : Wiley Publishing, Inc., 2007. ISBN-10: 0-470-05435-2.

19. Sun Microsystems. java.com. *What is Java Enterprise Edition (J2EE)?* [En línea] http://www.java.com/en/download/faq/j2ee.xml.

20. Jinoria Fernández, Yisel y Ruiz Rodríguez, Daynier. *Propuesta de Modelo Arquitectónico para Aplicaciones Empresariales sobre la Plataforma JEE.* Ciudad de La Habana : s.n., 2007.

21. Kruchten, Philippe. The "4+1" View Model of Software Architecture. *Architectural Blueprints.* s.l. : IEEE Software, 1995.

22. Ciberaula. *Área temática de Java.* [En línea] http://java.ciberaula.com/articulo/diseno_patrones_j2ee/.

23. Allamaraju, Subrahmanyam, y otros. *Programación Java server con J2EE Edicion 1.3.* s.l. : Anaya Multimedia, 2002.

24. Jendrock, Eric, y otros. The Java EE Tutorial. *Java EE 5 APIs.* [En línea] Sun. http://java.sun.com/javaee/5/docs/tutorial/doc/bnacj.html#bnacl.

25. Sun Microsystems. Java en Castellano. *Catálogo de Patrones de Diseño J2EE. I. Capa de Presentación.* [En línea] 1999. http://www.programacion.net/java/tutorial/patrones/1/.

26. Sun Microsystems . Java en Castellano. *Catálogo de Patrones de Diseño J2EE. Y II: Capas de Negocio y de Integración.* [En línea] 1999. http://www.programacion.net/java/tutorial/patrones2/1.

27. Degiovannini, Marcio. Asociación javaHispano. *Comparativa de frameworks web.* [En línea] 2007. http://www.javahispano.org/contenidos/es/comparativa_de_frameworks_web/.

28. Acharya, Sharad. java.net. The Source for Java Technology Collaboration. *Adopting a Java Persistence Framework: Which, When, and What?* [En línea] 2007. http://today.java.net/pub/a/today/2007/12/18/adopting-java-persistence-framework.html.

29. Walls, Craig y Breidenbach, Ryan. *Spring in Action 2nd Edition.* s.l. : Manning Publications Co., 2008. ISBN 1-933988-13-4.

30. Overview (JFreeChart Class Library (version 1.0.12)). [En línea] http://www.jfree.org/jfreechart/api/javadoc/index.html.

31. Object Refinery Limited. JFreeChart. [En línea] 2005-2009. http://www.jfree.org/jfreechart/.

32. The Eclipse Foundation. Eclipsepedia. *The Official Eclipse FAQs.* [En línea] 2009. http://wiki.eclipse.org/The_Official_Eclipse_FAQs.

33. NetBeans IDE. *Características.* [En línea] http://www.netbeans.org/features/index_es.html.

34. Funes, Luis Enrique. NetBeans Wiki. *Conociendo a NetBeans Platform: Introducción.* [En línea] 2008. http://wiki.netbeans.org/ConociendoNetbeansPlatformIntroduccion.

35. The Apache Software Foundation. Apache Tomcat. *Apache Tomcat.* [En línea] 1999-2008. http://tomcat.apache.org/.

36. The Apache Software Foundation . The Apache Tomcat 5.5 Servlet/JSP Container. *Documentation Index.* [En línea] 1999-2008. http://tomcat.apache.org/tomcat-5.5-doc/index.html.

37. Vargas, Alan. Sun Campus Ambassador UAEM. [En línea] 2009. http://blogs.sun.com/AlanVargas/entry/qu%C3%A9_es_glassfish.

38. Vargas, Alan . *GlassFish: El Servidor de Aplicaciones para todas tus Web Apps.* s.l. : Sun Mycrosystems.

39. JBoss EAP. jboss.org: community driven. [En línea] http://www.jboss.org/jbossas/.

40. JBoss EAP . JBoss Community. *JBoss Application Server.* [En línea] http://www.jboss.org/community/docs/DOC-10226.

41. Quiñones, Ernesto. PostgreSQL PE. *Introducción a PostgreSQL.* [En línea] 2007. www.postgresql.org.pe/articles/introduccion_a_postgresql.pdf.

42. MySQL AB, Sun Microsystems. MySQL. *MySQL 6.0 Reference Manual.* [En línea] 2008-2009. http://dev.mysql.com/doc/refman/6.0/en/index.html.

43. Icedo Toledo, Rosa Virginia y Trejo Vargas, Jorge Moisés. *SoftwareArchitecture Assesment.* s.l. : CIMAT, 2003.

44. Clements, Paul, Kazman, Rick y Klein, Mark. *Evaluating Software Architectures: methods and case studies.* s.l. : Addison Wesley, 2007. ISBN 978-0-201-70482-2.

45. Ionita, Mugurel T., Hammer, Dieter K. y Obbink, Henk. *Scenario-Based Software Architecture Evaluation Methods: An Overview.* Eindhoven, The Netherlands : Department of Mathematics and Computing Science, Technical University Eindhoven.

46. Camacho, Erika, Cardeso, Fabio y Núnez, Gabriel. *Arquitecturas de Software. Guía de Estudio.* 2004.

ANEXOS

Anexo 1: Documento de Especificación de requisitos

albet INGENIERÍA Y SISTEMAS

INFRAESTRUCTURA PRODUCTIVA

ESPECIFICACIÓN DE REQUISITOS DE SOFTWARE

SINAPSIS
Sistema Nacional Público para el Seguimiento de Inversiones y Sectores
Versión 0.1

Control de versiones

Fecha	Versión	Descripción	Autor
28/12/2008	0.1	Descripción de los requisitos no funcionales del sistema SINAPSIS.	George Pérez Concepción José Sevila Hidalgo
1/2/2009	1.0	Terminación del documento de requisitos no funcionales del sistema SINAPSIS	George Pérez Concepción José Sevila Hidalgo Daynier Ruiz Rodríguez

Reglas de Confidencialidad

Clasificación: Alta

Este documento contiene información propietaria de **ALBET Ingeniería y Sistemas y PDVSA**, y es emitido confidencialmente para un propósito específico.
El que recibe el documento asume la custodia y control, comprometiéndose a no reproducir, divulgar, difundir o de cualquier manera hacer de conocimientos público su contenido, excepto para cumplir el propósito para el cual se ha generado.

Estas reglas son aplicables a las 11 páginas de este documento.

1. Introducción

1.1. Propósito

Este documento tiene el propósito de mostrar todas las condiciones o capacidades que el sistema debe cumplir con sus respectivas especificaciones, así las propiedades o cualidades que el sistema deberá tener. En el mismo se especifica las propiedades del sistema que tienen que ver con las características no funcionales como son rendimiento, velocidad, uso de memoria, plataforma, fiabilidad, junto con la forma y las condiciones en que se van a medir estos requerimientos.

1.2. Alcance

Para esta etapa de levantamiento de requisitos se identificarán un conjunto de requisitos funcionales y no funcionales que responderán de manera directa a las necesidades de los clientes o usuarios del sistema a construir (SINAPSIS).

1.3. Definiciones, Acrónimos y Abreviaturas

Entidades: Se refiere a ministerios, organismos, entes, departamentos, etc.

2. Requisitos no funcionales

Para brindar una organización de los requisitos no funcionales que se especifican en este documento se ha empleado la clasificación por requisitos de calidad. A continuación mostramos las categorías en que agrupamos los requisitos no funcionales.

- Usabilidad: Requisititos que determinan las características generales de la capa de presentación del sistema en cuanto a las características de diseño gráfico de la misma, además de las facilidades para que el uso del sistema por parte del usuario final.
- Fiabilidad: Estos requerimientos están relacionados con la capacidad del usuario para confiar en las respuestas del sistema, en un sentido técnico, es decir, que la funcionalidad del sistema no se vea afectada por factores ajenos al sistema como los son los factores técnicos.
- Eficiencia: Los requerimientos clasificados en esta categoría están relacionados con tiempos de respuesta estimados, requeridos y esperados para la ejecución en línea de procesos del sistema, teniendo como base la plataforma tecnológica y escenarios específicos a los que en teoría el sistema estará expuesto y frente a los que deberá responder.
- Seguridad: Requerimientos relacionados con la confidencialidad de los datos en la

transmisión y en el almacenamiento, junto con las necesidades del sistema para evitar intrusiones no autorizadas al mismo y la capacidad para seguir eventos que comprometan esta seguridad a través del tiempo.

- Portabilidad: Estos requerimientos describen la capacidad del sistema para migrar de una plataforma hardware a otra sin que esto represente mayores traumatismos para el cliente del mismo, teniendo en cuenta los requisitos técnicos presentados y las generalidades naturales de configuración del sistema.
- Reusabilidad: Estos requerimientos consideran la capacidad de los componentes del sistema de prestar servicios a otros sistemas, de tal manera que el componente como valor adicional al cumplimiento de sus funciones pueda prestar sus servicios a otros sistemas sin que esto implique modificaciones o redefiniciones en dicho componente.
- Capacidad: Consideran los requerimientos de tecnología que permitirán al sistema ejecutar sus funciones de manera que responda a las expectativas del usuario en términos de eficiencia y eficacia en la respuesta de las operaciones. Se tienen en cuenta también las características, que harán que sea posible que el sistema funcione de manera estable durante un tiempo determinado, planeando el crecimiento del mismo.

3. Usabilidad

RNF.1 Cumplir con las pautas de diseño de las interfaces.

El sistema deberá tener una interfaz gráfica uniforme a través del mismo incluyendo pantallas, menús y opciones. Las pautas de diseño serán definidas por el equipo de diseño gráfico y se realizarán siguiendo los lineamientos de la arquitectura de información.

RNF.2 Agrupar vínculos y botones por grupos funcionales.

La consistencia de la interacción entre usuario y sistema estará determinada por el diseño de la interfaz de usuario que mantendrá los elementos como menús, banners y zona de trabajo, en posiciones fijas, además de la mayor uniformidad posible entre cuadros de texto y botones.

El sistema deberá ser de uso intuitivo, de tal forma que se reduzca los tiempos de entrenamiento, soporte y prueba por parte del usuario. La agrupación de los botones por funcionalidad determinará además la capacidad de componer la interfaz de acuerdo a las funciones requeridas por un rol determinado.

RNF.3 Los mensajes, títulos y demás textos que aparezcan en la interfaz del sistema deben aparecer en idioma español.

Tanto los títulos de los componentes de la interfaz, como los mensajes para interactuar con los usuarios, así como los mensajes de error, deberán ser en idioma español y tener una apariencia uniforme en todo el sistema. Los mensajes de error deberán ser lo suficientemente informativos para dar a conocer la severidad del error.

RNF.4 Utilizar campos de selección en la interfaz en los casos que sea posible.

El sistema deberá facilitar la entrada de datos a los usuarios, presentando campos de selección que permitan escoger los valores. Estos campos contendrán los valores posibles con los que se podrá llenar un determinado elemento en la interfaz, haciendo que el proceso de llenado de datos sea lo más intuitivo posible de tal forma que los usuarios se puedan adaptar con facilidad al sistema.
Los elementos mostrados en estos campos de valores deben ser configurables en el sistema de manera que se puedan cambiar, eliminar o agregar nuevos.

RNF.5 Contar con una ayuda en línea para guiar en el uso de la interfaz.

El sistema deberá proporcionar ayudas en línea, según la página donde el usuario se encuentre. La ayuda tendrá un nivel de detalle suficiente para que el usuario que no conoce la funcionalidad del sistema, entienda el contexto y pueda entender las operaciones del mismo, de tal forma que pueda, de manera intuitiva, determinar el paso que debe seguir para ejecutar lo que el sistema le permite. La ayuda debe contar con una descripción de cada una de las funciones posibles a realizar según el contexto y con imágenes que complementen esta información.

RNF.6 Permitir uso del teclado para realizar operaciones sobre el sistema (Permitir acceso rápido al sistema usando el teclado)

El sistema debe permitir su uso desde el teclado con funciones básicas como Tab para moverse por los campos de una página determinada, Enter para accionar un botón seleccionado. Se utilizarán tres funciones, dentro de las que se encuentran Ctrl+C para copiar una información seleccionada en un campo específico, Ctrl+V para pegar una información específica y Ctrl+X para cortar la información seleccionada.

RNF.7 Mostrar los valores de los campos numéricos utilizando separadores, según estándares.

El sistema debe ser preciso en cada una de las salidas debido al manejo de dígitos e información referente a diferentes montos asociados a los proyectos. Para esto debe seguir un estándar único en la codificación de las cifras y la separación de los dígitos, para la separación de miles se utilizará la coma y para las cifras decimales el punto (ej. 100, 000,000.00).

4. Fiabilidad

RNF.8 Prever contingencias para eventos de caída del sistema.

El sistema deberá prever contingencias que pueden afectar la prestación estable y permanente del servicio. La siguiente es la lista de las contingencias que se deben tener en cuenta y se pueden considerar críticas:

- Sobrecarga del sistema por volumen de usuarios.
- Caída del sistema por sobrecarga de procesos.
- Caída del sistema por sobrecarga de transacciones.
- Caída del sistema por volumen de datos excedido en la base o bodega de datos.

Estas consideraciones implicarán que la infraestructura técnica sobre la que se implantará el sistema garantice una alta disponibilidad del mismo.

5. Eficiencia

RNF.9 Responder en tiempos aceptables las peticiones que se realicen en el sistema.

El sistema debe ser capaz de dar respuestas a las peticiones con un nivel aceptable de desempeño. Teniendo en cuenta el nivel de concurrencia que pueda existir, debe ser capaz de prestar servicio sin que se deterioren los tiempos de respuestas

6. Seguridad

RNF.10 Permitir el intercambio de datos entre el cliente y el servidor por canales cifrados.

El sistema deberá permitir la transmisión por canales cifrados cuando se trate de información confidencial, de manera que no viaje en texto plano por la red.

RNF.11 Almacenar de manera cifrada las claves de los usuarios en la base de datos.

Las contraseñas de los usuarios no deben ser almacenadas en texto plano, esta información debe

ser privada y específica de cada uno, de manera que nadie pueda reemplazar la identidad de un usuario en el sistema.

RNF.12 Registrar cada una de las operaciones llevadas a cabo por un usuario en el sistema.
Se debe realizar un registro de cada una de las acciones que realiza un usuario en el sistema de manera que se tenga la fecha y hora en se realizó una determinada operación y quien la realizó.

RNF.13 Definir una jerarquía de usuarios para el manejo centralizado de los mismos en el sistema.
Se debe crear una jerarquía de usuarios, los cuales deben ser manejados por administradores de cada uno de los niveles. Las acciones que podrá realizar un usuario estarán determinadas por los roles que tendrá el usuario. Cada usuario pertenecerá a una entidad específica, existirán administradores en cualquier entidad que se desee.

RNF.14 Permitir la creación de roles de usuarios
Los roles serán creados por un administrador central y se podrán asignar a determinadas entidades de manera que los administradores de dichas entidades sólo puedan crear usuario con dichos roles. Los roles contendrán funcionalidades específicas del sistema que determinarán el acceso a las mismas por parte de los usuarios.

RNF.15 Permitir la creación de grupos de usuarios
Los grupos de usuarios contarán con un conjunto de roles asignados, luego un usuario puede formar parte de uno o varios grupos de usuario, a dicho usuario se le asignarán los roles definidos para cada uno de los grupos de usuarios a los que pertenece. Los grupos de usuarios se crearán a nivel central y se podrán asignar a determinadas entidades al igual que los roles.

RNF.16 Permitir la gestión y administración de usuarios en el sistema
El sistema debe permitir la creación, modificación, activación y desactivación de los usuarios del mismo. De cada usuario se debe almacenar: el nombre del usuario, nombre y apellidos de la persona, entidad a la cual pertenece, número de identificación, teléfono y correo electrónico. A cada usuario se le podrán asignar varios roles y a su vez se puede incluir en varios grupos de usuarios.

RNF.17 Definir niveles de acceso a la información en el sistema.

La información que maneja el sistema siempre va ser restringida de acuerdo a la entidad donde se originó, permitiendo sólo el acceso a la misma a dicha entidad y a las entidades rectoras de la misma.

RNF.18 Restringir el acceso al sistema.

Para acceder al sistema cada usuario debe ser autenticado, de manera que se pueda determinar si tiene acceso al mismo y en cado de ser así determinar las funcionalidades a las que tiene acceso y restringir su actividad al uso de las mismas las cuales están determinadas por los roles que contiene el usuario. Para autenticarse el usuario debe colocar: nombre de usuario y contraseña.

RNF.19 Identificar autenticaciones forzadas realizadas por programas.

El sistema debe permitir identificar las autenticaciones que puedan ser realizadas por programas con el objetivo de forzar la entrada al sistema.

RNF.20 Permitir la gestión de las contraseñas

El sistema debe permitir que cada usuario pueda cambia su contraseña, así como debe exigir que la misma tenga un nivel aceptable de complejidad. Además se debe permitir generar una contraseña de un usuario.

7. Portabilidad

RNF.21 Permitir la configuración de cada unos de los nomencladores del sistema.

Los valores de los nomencladores deben poder ser modificados, eliminados y agregados, de manera que puedan ser incorporados al sistema de manera automática.

RNF.22 Permitir configuración de jerarquía de entidades

El sistema debe permitir la configuración de cada una de las entidades que son la base de la gestión de documentos que se mueve en el mismo, de manera que se puedan modificar sus datos, eliminar y crear nuevas, además se debe permitir definir la jerarquía existente entre cada una de estas entidades.

RNF.23 Garantizar compatibilidad con navegadores de uso común

El sistema deberá ser compatible con los navegadores:

- Microsoft Internet Explorer 6.0 o superior

- Mozilla Firefox 2.0 o superior

RNF.24 Utilizar estándares de codificación

El código fuente del sistema deberá cumplir con un estándar de codificación. El estándar especificado debe considerar puntos como: Estándares de nombres utilizados en todos sus objetos: programas, formas, tablas, campos, índices, procedimientos, paquetes. Codificación de los comentarios para la generación automática de la documentación. Empleo de las características del IDE (Interface Development Enviroment) para el formato del código.

RNF.25 Diseñar un sistema compuesto por módulos, que agrupen funcionalidad.

El sistema deberá garantizar que cada subsistema, aplicación y componente tienen fronteras claramente definidas y funciones relacionadas. Cada módulo será independiente del resto.

RNF.26 Garantizar que las actualizaciones del sistema sean a nivel central (Servidor)

El sistema deberá estar orientado a que las actualizaciones sólo se hagan en el sitio del servidor, de tal manera que no sea necesario actualizar todos y cada uno de los clientes que acceden a la información del sistema.

RNF.27 Emplear el protocolo ssl (Secure Socket Layer). para el aseguramiento del canal

El sistema deberá incorporar el protocolo de transmisión segura de datos a través de Internet SSL como método de aseguramiento del canal.

8. Reusabilidad

RNF.28 Garantizar que los formatos de los archivos de salida del sistema sean compatibles con los programas más comunes.

Los ficheros que genere el sistema deben utilizar formatos estándares como (rtf, pdf, xsl) de manera que sean compatibles con las siguientes herramientas:

- Microsoft Office 2003 o superior
- Acrobat Reader 6.0 o superior
- Open Office 2.3 o superior

RNF.29 Definir un modelo tres capas para el sistema

El sistema deberá considerar en su arquitectura un modelo tres capas, donde se definen tres componentes lógicos de manera independiente: capa de presentación o interfaz de usuario, capa de lógica de negocio y capa de datos. Esta arquitectura determina una separación entre la lógica del negocio y la presentación de la aplicación, lo que permite el uso de servicios desde la capa de lógica del negocio ya sea por la capa de presentación del sistema o por otros sistemas que requieran el mismo servicio.

9. Capacidad

RNF.30 Considerar características técnicas mínimas para la ejecución en clientes

Para que un cliente de la aplicación pueda ejecutar procesos, en línea, considerados en el sistema el punto de acceso deberá cumplir con los siguientes requisitos mínimos.

- Procesador 2.0 GHz
- Memoria 512 MB.
- Disco duro 20 GB.
- Sistema Operativo Windows 98, 2000, XP o para Servidor o Linux.
- Navegador internet Explorer 6.0 o posterior, Mozilla Firefox 2.X
- Conexión a Internet. mínimo 56Kbps

Para que un cliente de la aplicación pueda observar y analizar resultados de los procesos consignados en documentos electrónicos, el punto de acceso deberá cumplir con los siguientes requisitos mínimos.

- Herramienta Microsoft Office 2003 o superior
- Herramienta Acrobat Reader 6.0 o superior
- Herramienta Open Office 2.3 o superior

Anexo 2: Plantilla para Documento de Arquitectura de Software

Documento de Arquitectura de Software

Documento de Arquitectura de Software

Rector

<Nombre del Proyecto>

<Nombre del producto>

<Versión>

Control de versiones

Fecha	Versión	Descripción	Autor
<dd/mmm/yy>	<x.x>	<detalles>	<nombre>

Reglas de Confidencialidad

Clasificación: <<Clasificación>>

Este documento contiene información propietaria de **ALBET Ingeniería y Sistemas** y/o **"<<Empresa Cliente>>"** , y es emitido confidencialmente para un propósito específico.

El que recibe el documento asume la custodia y control, comprometiéndose a no reproducir, divulgar, difundir o de cualquier manera hacer de conocimientos público su contenido, excepto para cumplir el propósito para el cual se ha generado.

Estas reglas son aplicables a las X páginas de este documento.

1. Introducción

1.1. Propósito

[Resumen del propósito de este documento]

1.2. Alcance

[Breve descripción del alcance del documento de arquitectura]

1.3. Definiciones, Acrónimos y Abreviaturas

1.4. Referencias

[Lista de documentos a los que se hace referencia en el Plan]

Código	Título
[1]	Documento 1
[2]	Documento 2
[3]	Modelo de Diseño - Módulo de Administración v0.0

2. Representación Arquitectónica

[Incluir en esta sección una descripción de que es la arquitectura de software para el sistema, y como esta es representada. La descripción de la representación aborda sobre el como la arquitectura y el diseño son representados, las convenciones de modelado y los artefactos usados para presentar la información.]

3. Objetivos y Restricciones Arquitectónicas

[Incluir en esta sección una descripción de los requerimientos y objetivos del software que tienen un impacto significativo en la arquitectura.

- *Interoperabilidad con sistemas legados*
- *Uso de estándares de datos*
- *Política de manejo de datos (cifrado)*
- *Distribución geográfica o física de la entidad a automatizar*

- *Otras restricciones que puedan atentar con el cumplimiento de los objetivos.]*

4. Tamaño y Rendimiento

[Incluir en esta sección una descripción las características de dimensiones importantes del software que afectan la arquitectura, así como las restricciones de rendimiento para la puesta a punto.

- *Prever el crecimiento de los datos.*
- *Tiempo de respuesta.*
- *Niveles concurrencia*

5. Vista de casos de uso

[Incluir en esta sección el modelo de casos de uso y de forma opcional el modelo de casos de uso del negocio]

5.1. Casos de uso arquitectónicamente significantes

[Incluir en esta sección el diagrama de casos de uso arquitectónicamente significantes. Pueden ser añadidas notas al diagrama para explicar porque un caso de uso en específico es arquitectónicamente significante.]

5.1.1. Nombre de caso de uso: Breve descripción.

5.1.1.1. Realización del caso de uso: Documentación de la realización del caso de uso, Diagrama de interacción.

6. Vista Lógica

[Incluir la descripción de las clases más importantes, su organización en paquetes y subsistemas, y la organización de estos subsistemas en capas. La descripción se realiza a través de diagramas de clases para ilustrar la relación entre las clases arquitectónicamente significantes, subsistemas, paquetes y capas.]

6.1. Elementos del modelo arquitectónicamente significantes

[Incluir un diagrama con los elementos del modelo de diseño que son arquitectónicamente significantes]

6.2. Visión general de la arquitectura – Alineamiento de paquetes, subsistemas y capaz

[Incluir en esta sección un diagrama que ilustre la organización del modelo de diseño en capas lógicas e incluir la descripción de cada una de ellas. Enumeración de las tecnologías a aplicar por cada una de las capas y subsistemas.]

7. Vista de procesos

[Incluir en esta sección la descripción de las tareas (procesos, hilos, tareas programadas, eventos y notificaciones) involucrados en la ejecución del sistema. Además de incluir la ubicación de las clases y objetos necesarios para estas tareas de ser necesario.]

7.1. Diagrama de vista de procesos que muestra la composición de los procesos e hilos, y la distribución de clase en estos procesos e hilos.

7.1.1. Nombre del proceso: Incluir una descripción sobre el role del proceso en el sistema.

7.1.1.1. Nombre del hilo: Este hilo pertenece al proceso 7.1 debe incluirse una descripción del la tarea a realizar por el mismo.

8. Vista de despliegue

[Incluir en esta sección la descripción de los nodos físicos para la mayoría de las configuraciones tanto para usuarios finales como para desarrolladores y probadores. Además ubicar las tareas (de la vista de procesos) y las capas lógicas en los nodos físicos. Esta descripción se realiza a través del diagrama de despliegue seguido por la distribución de los procesos y las capas lógicas en cada uno de los procesadores.]

8.1. Diagrama de despliegue.

8.1.1. Nombre de dispositivo: descripción de la capacidad que el dispositivo provee al sistema.

8.1.2. Nombre del procesador: descripción de la funcionalidad y capacidad del nodo.

8.2. Descripción de elementos e interfaces de comunicación

8.2.1. <<Nombre tipo de conexión>>: Características físicas de la conexión

9. Vista de Implementación

[Incluir en esta sección la colección de componentes y subsistemas de implementación mediante el modelo de implementación (diagrama de componentes). Esta vista define además, ejecutables, bibliotecas, ficheros, subsistemas, dependencias entre ellos.]

10. Vista de Datos

[Incluir en esta sección los elementos persistentes arquitectónicamente significativos en el modelo de datos. Una vista general de la tecnología usada para lograr la persistencia de las entidades del sistema además de la descripción del modelo de datos en términos de tablas, vistas, triggers y procedimientos almacenados.]

11. Calidad

[Incluir en esta sección como la arquitectura de software contribuye a la capacidad (más que a la funcionalidad) del sistema: extensibilidad, confiabilidad, portabilidad, escalabilidad, tolerancia a fallos, balance de carga, etc.]

GLOSARIO

API: Interfaz de Programación de Aplicaciones (Application Programming Interface), conjunto de funciones y procedimientos que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.

BSD: Distribución de Software Berkeley (Berkeley Software Distribution), utilizado para identificar un sistema operativo derivado del sistema Unix nacido a partir de los aportes realizados a ese sistema por la Universidad de California en Berkeley.

CASE: Ingeniería de Software Asistida por Ordenador (Computer Aided Software Engineering), diversas aplicaciones informáticas destinadas a aumentar la productividad en el desarrollo de software reduciendo el coste de las mismas en términos de tiempo y de dinero; pueden ayudar en todos los aspectos del ciclo de vida de desarrollo del software en tareas como el proceso de realizar un diseño del proyecto, calculo de costes, implementación de parte del código automáticamente con el diseño dado, compilación automática, documentación o detección de errores entre otras.

CPU: Unidad Central de Procesamiento (Central Processing Unit), "el procesador", componente en una computadora digital que interpreta las instrucciones y procesa los datos contenidos en los programas de la computadora.

GNU: Proyecto iniciado por Richard Stallman con el objetivo de crear un sistema operativo completamente libre: el sistema GNU.

GNU GPL: Licencia Pública General de GNU (GNU General Public Licence), es una licencia creada por la Free Software Foundation a mediados de los 80, y está orientada principalmente a proteger la libre distribución, modificación y uso de software. Su propósito es declarar que el software cubierto por esta licencia es software libre y protegerlo de intentos de apropiación que restrinjan esas libertades a los usuarios.

GUI: Interfaz Gráfica de Usuario (Graphic User Interface), artefacto tecnológico de un sistema interactivo que posibilita, a través del uso y la representación del lenguaje visual, una interacción amigable con un sistema informático.

HTML: Lenguaje de Marcas de Hipertexto (HyperText Markup Language), lenguaje de marcado predominante para la construcción de páginas web; usado para describir la estructura y el contenido en forma de texto, así como para complementar el texto con objetos tales como imágenes.

HTTP: Protocolo de Transferencia de Hipertexto (Hipertext Transfer Protocol), protocolo usado en cada transacción de la Web.

HQL: Lenguaje de Consultas de Hibernate (Hibernate Query Languaje), lenguaje de consultas que utiliza el Hibernate orientado a objetos y le permite al framework utilizar un lenguaje común independientemente del gestos de base de datos que se esté utilizando.

IP: Protocolo de Internet (Internet Protocol), protocolo usado para la comunicación de datos a través de una red.

Javascript: Lenguaje interpretado, no requiere compilación, utilizado principalmente en páginas Web.

JEE: Edición Empresarial de Java (Java Enterprise Edition), plataforma de programación para desarrollar y ejecutar software de aplicaciones en Lenguaje de programación Java con arquitectura de N niveles distribuida, basándose ampliamente en componentes de software modulares ejecutándose sobre un servidor de aplicaciones.

JMS: Servicios de Mensajería de Java (Java Message Service), solución creada por SUN para el uso de colas de mensajes. Este es un estándar de mensajería que permite a los componentes de aplicaciones basados en la plataforma de Java 2 crear, enviar, recibir y leer mensajes. También hace posible la comunicación confiable de manera síncrona y asíncrona.

JSE: Edición Estándar de Java (Java Standard Edition), colección de APIs del lenguaje de programación Java útiles para muchos programas de la Plataforma Java.

MAC: Control de Acceso al Medio (Media Access Control address), identificador de 48 bits que corresponde de forma única a una tarjeta o interfaz de red.

Ofimática: Equipamiento hardware y software usado para idear y crear, coleccionar, almacenar, manipular y transmitir digitalmente la información necesaria en una oficina para realizar tareas y lograr objetivos básicos.

ONAPRE: Oficina Nacional de Presupuesto es el órgano rector del Sistema Presupuestario Público de la República Bolivariana de Venezuela.

ORM: Mapeo Objeto-Relacional (Object-Relational Mapping), es una técnica de programación para convertir datos entre el sistema de tipos utilizado en un lenguaje de programación orientado a objetos y el utilizado en una base de datos relacional.

Plug-in: Complemento, aplicación que se relaciona con otra para aportarle una función nueva y generalmente muy especifica. Esta aplicación adicional es ejecutada por la aplicación principal e interactúan por medio de la API.

POJO: (Plain Old Java Object) utilizada por programadores Java para enfatizar el uso de clases simples y que no dependen de un framework en especial.

SOAP: Protocolo de Acceso a Objetos Simples (Simple Object Access Protocol), protocolo estándar que define cómo dos objetos en diferentes procesos pueden comunicarse por medio de intercambio de datos XML.

SQL: Lenguaje de consulta estructurado (Structured Query Languaje), lenguaje declarativo de acceso a bases de datos relacionales que permite especificar diversos tipos de operaciones sobre las mismas.

SSL: Protocolo de Capa de Conexión Segura (Secure Sockets Layer), protocolo criptográfico que proporciona comunicaciones seguras por una red, comúnmente Internet.

Terminal: Dispositivo hardware usado para introducir o mostrar datos de una computadora.

UML: Lenguaje Unificado de Modelado (Unified Modeling Language), lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad, lenguaje gráfico para visualizar, especificar, construir y documentar un sistema.

WML: Lenguaje de Marcado Inalámbrico (Wireless Markup Language), lenguaje utilizado para construir las páginas que aparecen en las pantallas de los teléfonos móviles y los asistentes personales digitales dotados de tecnología WAP. Es una versión reducida del lenguaje HTML que facilita la conexión a Internet de dichos dispositivos y que además permite la visualización de páginas web en dispositivos inalámbricos que incluyan la tecnología WAP.

XML: Lenguaje extensible de marcas (Extensible Markup Language), es un lenguaje creado por W3C (World Wide WEB Consortium) para describir la estructura de los datos a almacenar en un documento. La inmensa mayoría de los lenguajes de programación han implementado interfaces para trabajar con este nuevo estándar.