Universidad de las Ciencias Informáticas

Facultad 3

Propuesta de arquitectura para el proyecto SIGESPRO

Trabajo de Diploma para optar por el título de

Ingeniero en Ciencias Informáticas

Autor(es): Julio Cesar Prieto Alvarez

Tutor(es): Pedro Enrique Castiñeiras Sánchez

Dairo Reyes Rodríguez

Ciudad de la Habana

Mayo del 2009

"Saber no es suficiente, debemos aplicar. Desear no es suficiente, debemos hacer."

Johann Wolfgang von Goethe.

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 _____.

Julio Cesar Prieto Alvarez	**Pedro Enrique Castiñeiras**	**Dairo Reyes Rodríguez**
_____	_____	_____
Firma del Autor	Firma del Tutor	Firma del Tutor

DATOS DE CONTACTO

Nombre: Pedro Enrique Castiñeira Sanchez.
Descripción: Ingeniero en Ciencias Informáticas, instructor recién graduado, líder del proyecto SIGESPRO.
Correo electrónico: pesanchez@uci.cu

Nombre: Dairo Reyes Rodriguez.
Descripción: Ingeniero en Ciencias Informáticas, instructor recién graduado.
Correo electrónico: dreyes@uci.cu

AGRADECIMIENTOS

A mi mamá por luchar tanto para formarme como un hombre de bien.

A Alberto Quijano, por guiarme, junto a mi mamá, por el camino correcto de la vida.

A Alicia por aguantar todos mis resabios y estar conmigo durante estos años.

A mi tía Sandra por alentarme y darme fuerzas en los momentos duros, por ayudarme cuando realmente lo necesité.

A mi familia en general por el apoyo que me han brindado.

A mis amistades por compartir conmigo durante estos años.

A mis tutores por ayudarme tanto en los momentos difíciles por los que pasé antes de llegar aquí.

DEDICATORIA

De manera especial a mi mamá, quien ha sido madre, padre y amiga. A ella que me dio la vida y ha sabido sacrificarse para darme lo mejor. A ella que es mi orgullo y ejemplo a seguir. A ella que es la persona que más quiero y admiro, el amor de mi vida.

A Alberto Quijano, el padre que necesité en momentos muy difíciles de mi vida. El compañero y amigo con el que cuento.

RESUMEN

El Polo de Gestión Gubernamental de la Universidad de las Ciencias Informáticas se ha propuesto desarrollar un software para la gestión de proyectos a nivel gubernamental, de ahí la necesidad de realizar un diseño de alto nivel que cumpla con las necesidades de dicho polo. La calidad del software es imprescindible.

Este trabajo describe un estudio del estado del arte de los principales elementos referentes a arquitectura de software que ayudaron a definir una propuesta para SIGESPRO, teniendo en cuenta que esta garantice la seguridad de la información, escalabilidad, portabilidad y la estabilidad que requieren los sistemas de este tipo. El estudio realizado comprende además un análisis de los sistemas existentes para la gestión gubernamental, haciendo énfasis en el sistema para el Convenio integral de colaboración Cuba-Venezuela (CICCV) y Sinapsis, proyectos desarrollados por el Polo.

Asi mismo la propuesta se documenta a partir de la generación del artefacto Documento de descripción de la arquitectura, definido para el rol de arquitecto dentro de la metodología RUP propuesto por la Universidad de las Ciencias Informáticas, que se destina para tal efecto. En vista a la validación de la propuesta se empleó el método de Análisis de diseños intermedios (ARID, por sus siglas en inglés). Este permite evaluar determinados atributos de calidad en la arquitectura a partir de determinados escenarios.

PALABRAS CLAVES

Arquitectura de software, patrones de diseño, patrones arquitectónicos, estilos arquitectónicos, lenguajes de programación, leguajes de modelado, ente.

Índice

AGRADECIMIENTOS III

DEDICATORIA IV

RESUMEN V

INTRODUCCIÓN 1

1 FUNDAMENTACIÓN TEÓRICA 5

1.1. Introducción 5
1.2. Estado del Arte 5
1.3. Arquitectura de Software 6
1.3.1. Breve reseña histórica. 6
1.3.2. ¿Qué es la arquitectura de Software? 7
1.3.3. Estilos arquitectónicos 8
1.3.3.1. Ejemplos de estilos arquitectónicos 9
1.3.4. Patrones Arquitectónicos 15
1.3.4.1. Patrones más usados 16
1.4. Metodologías de Desarrollo. 18
1.5. Lenguajes de programación 23
1.6. Lenguaje Unificado de Modelado. 29
1.7. Herramientas 29
1.7.1. Marcos de Trabajo 29
1.7.2. Gestor de Bases de Datos. 34
1.8. Tendencias actuales para el desarrollo de software 35
1.8.1. ¿Qué es software libre? 36
1.8.1.1. Software Libre y Open Source 36
1.9. Conclusiones 37

2 DESCRIPCIÓN DE LA ARQUITECTURA PARA SIGESPRO 38

2.1 Introducción. 38
2.2 Descripción de la arquitectura. 38
2.2.1 Introducción 38

2.2.2 Representación arquitectónica. 38
2.2.3 Objetivos y restricciones arquitectónicas. 39
2.2.4 Tamaño y Rendimiento 41
2.2.5 Vista de Casos de Uso 43
2.2.6 Vista lógica. 51
2.2.7 Vista de despliegue 55
2.2.8 Vista de Implementación. 57
2.2.9 Vista de Datos. 59
2.3 Conclusiones. 60
3 VALIDACIÓN DE LA ARQUITECTURA 61
3.1 Introducción. 61
3.2 Etapas en que se evalúa una arquitectura 61
3.3 Atributos de calidad. 61
3.4 Técnicas de evaluación de arquitecturas. 62
3.5 Métodos de evaluación de arquitecturas. 63
3.5.1 SAAM. 63
3.5.2 ATAM. 64
3.5.3 ARID. 65
3.6 Evaluación de la arquitectura. 66
3.7 Conclusiones 68
4 CONCLUSIONES GENERALES. 69
5 RECOMENDACIONES 70
6 BIBLIOBRAFÍA 71
ANEXOS 74
GLOSARIO DE TERMINOS. 81

INTRODUCCIÓN

Para las aplicaciones computacionales existe una demanda permanente por mayor funcionalidad, mayor número de servicios, más flexibilidad y mejor rendimiento, por tanto, al diseñar un nuevo sistema de información, se deben buscar siempre formas para enlazar las soluciones ofrecidas por la tecnología disponible a las necesidades de las aplicaciones de los usuarios. Los productores de software de hoy centran su atención en Internet para el desarrollo de aplicaciones por las facilidades que este brinda.

Con el surgimiento de la World Wide Web, más conocida por WWW, se comienza a hacer uso de la esta red para fines comerciales. El envío por la red de documentos de hipertexto (páginas web) es la función más importante de WWW que le proporcionaron este carácter comercial a internet. (Acosta).

Las organizaciones buscan cada día herramientas que les ayuden en la planeación, organización, control y seguimiento de proyectos. La administración de los proyectos, se ha convertido en un proceso fundamental de las empresas. El objetivo es garantizar que los recursos estén siendo utilizados adecuadamente y que los resultados de los mismos logren el impacto esperado en la organización y de los involucrados en general.

Todo este desarrollo informático ha permitido integrar fácilmente el acceso y gestión de la información, por eso la gran mayoría de las instituciones han hecho un gran uso de estos avances, incluyendo las instituciones gubernamentales. Gobiernos como Bolivia (http://www.bolivia.gov.bo/), Venezuela (*http://*www.gobiernoenlinea.ve/*)* y Nicaragua (http://www.elpueblopresidente.com/) constituyen un ejemplo de esto.

En el contexto de la colaboración internacional el término gestión de proyectos ha encontrado una de sus más acabadas definiciones, al abordar los proyectos de desarrollo como un método sistémico de búsqueda de alternativas al interactuar con diversas organizaciones que financian proyectos que van desde agencias hasta proyectos multilaterales. (Roche)

Existen numerosas herramientas que apoyan la gestión de proyectos. Entre las más conocidas se pueden mencionar el DotProject y Microsoft Project que facilitan la planificación y seguimiento de proyectos. Para el desarrollo de actividades en un ambiente colaborativo se encuentran los Groupware, sistemas que facilitan la colaboración entre varias partes. Sin embargo todas estas herramientas tienen un costo muy elevado y no agrupan todas las necesidades en una sola herramienta. Además su objetivo fundamental es la gestión de proyectos a nivel empresarial.

Introducción

El Polo de Gestión Gubernamental (PGG) de la Universidad de las Ciencias Informáticas (UCI), como su nombre lo indica, está inmerso en el desarrollo de software para la gestión de proyectos entre entidades semejantes de varios gobiernos. Un ejemplo funcional de este tipo de proyectos es la Informatización del Convenio Integral de Cooperación Cuba – Venezuela (CICCV).

El proyecto de software Sistema de Gestión de Proyectos (SIGESPRO), surge con la idea de desarrollar un producto flexible y adaptable a diversos entornos para la gestión de proyectos, esta gestión puede ser entre gobiernos, empresas o cualquier combinación de estos. Debe contener elementos que apoyen la toma de decisiones y la interacción entre las contrapartes. La flexibilidad y adaptabilidad de este producto permitirá obtener de manera ágil productos adaptados a las necesidades de diversos clientes gubernamentales.

En el PGG se hace uso de tecnologías que favorecen el desarrollo de los proyectos del mismo. La plataforma Java Enterprise Edition (JEE) es una de las tecnologías que se utilizan en dicho polo, que con el uso de marcos de trabajo como Hibernate y Spring brindan una robustez indiscutible a las aplicaciones. Cuenta con personal capacitado en estas, constituyendo una de sus principales fortalezas.

No obstante, en la arquitectura definida para el desarrollo de los proyectos del polo existen algunos problemas que atentan contra el buen desarrollo de los mismos, provocando que el tiempo de desarrollo sea demasiado largo. Entre los factores que afectan el desarrollo ágil de estos, el más considerable es, sin lugar a dudas, la poca reutilización de subsistemas ya desarrollados para otros proyectos, esto se debe a que su implementación es muy específica para las aplicaciones que se desarrollan. La documentación de estos subsistemas es poca o nula, lo que hace que el código generado en cada proyecto apenas se puede reutilizar. Por cada proyecto hay que invertir tiempo y recursos para configurar o adaptar estos subsistemas a las nuevas aplicaciones, o en muchos casos hay que comenzar a implementarlas desde cero, trayendo consigo que se necesite mucho tiempo para la entrega de los productos.

Por tanto, se impone la necesidad de hacer un diseño de alto nivel para SIGESPRO donde la adaptabilidad y la flexibilidad de los subsistemas sea lo más importante, permitiendo mitigar o aminorar el impacto de los principales problemas que hoy afronta el PGG. Para ello es necesario que exista un bajo acoplamiento entre los elementos del sistema, pero a la vez una alta cohesión, pues deben establecerse formas de comunicación entre ellos.

La situación anteriormente expuesta conlleva al siguiente **problema,** que sirve de directriz a la presente investigación. ¿Cómo lograr una arquitectura flexible y adaptable para el sistema SIGESPRO, que permita una mayor reutilización de sus subsistemas?

La ciencia que estudia dicho fenómeno es el Proceso de Desarrollo de Software, por lo que se le considera como el **objeto de estudio** de esta investigación. La arquitectura de software para el proyecto SIGESPRO constituye el **campo de acción** por ser esta el área, dentro de dicha ciencia, donde se va a centrar la investigación.

Este trabajo persigue el siguiente **objetivo**: Definir una arquitectura flexible y adaptable para el sistema SIGESPRO.

Con esta investigación se defiende la siguiente **idea:** Si se define una arquitectura flexible y adaptable para el sistema SIGESPRO entonces permitirá una mejor reutilización de sus subsistemas

Para lograr el cumplimiento de los objetivos propuestos se trazaron un grupo de **tareas investigativas,** estas son:

- Analizar el estado del arte de arquitecturas de software, software de gestión gubernamental existentes, metodologías de desarrollo y herramientas disponibles que sirvan de apoyo para el desarrollo de software de gestión de proyectos.
- Definir los modelos o vistas de arquitectura para el sistema SIGESPRO.
- Realizar una evaluación de la efectividad de la arquitectura propuesta.

En la presente investigación se hace uso de varios **métodos científicos de la investigación** por la importancia de estos en el desarrollo de una investigación. A continuación se hace referencia a ellos:

Métodos teóricos.

- *Inducción-Deducción:* Permitió arribar a conclusiones para la determinación del problema.
- *Analítico-Sintético:* Permitió realizar un estudio exhaustivo del objeto para de esta forma extraer los elementos más importantes y arribar a conclusiones dentro de la investigación que fomentaron el objetivo.
- *Hipotético-Deductivo:* Desempeñó un papel esencial en el proceso de verificación de la idea que se defiende, ya que a partir de esta se llegaron a nuevas conclusiones y predicciones que se fomentaron con la culminación de la investigación.
- *Histórico-Lógico:* Se utilizó para analizar la trayectoria completa del objeto, su condicionamiento a los diferentes períodos de su historia, así como las etapas principales de su desenvolvimiento. Este método fue de vital importancia para la determinación del estado del arte.

Métodos empíricos.

- *Observación:* Es utilizado en todo el proceso de la investigación ya que este es el instrumento universal del científico, se realiza de forma consciente y orientada a un objetivo determinado. Este método se utilizó para obtener una caracterización detallada de las soluciones existentes para sistemas similares, con vista a valorar si a partir de ellas se pueden mitigar los principales problemas del PGG.

La razón de ser del PGG es crear sistemas de gestión de proyectos en el menor tiempo posible y que cumpla con las expectativas del cliente, de ahí la necesidad de incursionar en nuevas herramientas que pudieran ser de gran utilidad para el mismo.

La investigación tiene un gran nivel de actualidad debido a que el PGG contará con un sistema más flexible y adaptable que minimizará los recursos que se dispongan a cada proyecto, así como el tiempo de desarrollo de los mismos. El uso de las nuevas tecnologías existentes en el mundo para el desarrollo de software, similares a los que se desarrollan en el PGG.

El desarrollo de este trabajo de diploma se divide en 3 capítulos fundamentales. El **primer capítulo**, titulado “Fundamentación Teórica”, está centrado en tratar los principales elementos teóricos utilizados en cada una de las fases de la investigación, prestando especial atención a la arquitectura de software, lenguajes de programación y facilidades de las diferentes herramientas existentes para el desarrollo de proyectos similares en el Mundo y la UCI.

El **segundo capítulo**, **“**Propuesta de arquitectura para SIGESPRO”, en este capítulo se realiza la propuesta de solución al problema planteado inicialmente, regida por el artefacto Documento Descripción de Arquitectura que propone la Universidad de las Ciencias Informáticas a emplear en los proyectos productivos.

El **tercer capítulo**,” Evaluación de la propuesta de arquitectura para SIGESPRO”, en este capítulo se realizará una evaluación de la propuesta descrita en el segundo capítulo. Aquí se tratarán métodos existentes para la evaluación de arquitecturas de software, así como la aplicación de uno de ellos a la solución que se propone. El análisis de dicha evaluación es otro de los elementos importantes que aporta este capítulo.

1 FUNDAMENTACIÓN TEÓRICA

1.1. Introducción

En el presente capítulo se detallan los principales elementos teóricos tenidos en cuenta para la solución al problema anteriormente descrito. Aquí se hace un análisis del estado actual del problema que se ha planteado, así como de las principales tendencias existentes a nivel mundial para este tipo de software.

Se hace un estudio detallado de las principales tecnologías, lenguajes de programación, metodologías de desarrollo y herramientas que pudieran ser utilizadas para dar solución a dicho problema, teniendo especial cuidado en que tengan licencia permisivas para el uso comercial. El principal objetivo de este capítulo es establecer comparaciones que ayuden a determinar cuál es la solución más factible para este problema.

1.2. Estado del Arte

Existen en el mercado una gran cantidad de productos de software orientados a la administración de los proyectos, muchos de los cuales son ampliamente utilizados en las organizaciones, tales como: Microsoft Project, Primavera microsystem, y PlanView. Todas estas herramientas ofrecen diversos módulos con capacidades para definir, planear y administrar la cartera de proyectos y los proyectos definidos, que incluso se apegan a las mejores prácticas sugeridas por el Project Management Institute (PMI). (Project Management, 2004)

Primavera microsystem es un sistema de más de 20 años de experiencia, uno de los más antiguos dentro de los software de gestión de proyectos. El propietario actual es Oracle. Es utilizado en grandes empresas como AT&T y Hewlet Packad, demostrando gran efectividad en los mismos.

Planview es otro software para la gestión de proyectos ofrece potentes herramientas para la consecución de todos los objetivos empresariales, dígase orientación a resultados, optimización de costes, mitigación de riesgos. Sus principales potencias radican en:

- Gestión Financiera, con un completo e integrado sistema con todas las funcionalidades necesarias para mejorar la planificación, ejecución y consecución del presupuesto, previsiones de ventas y el seguimiento de costes y beneficios

- Portfolio Management[1]: solución construida especialmente con una orientación hacia organizaciones que desarrollan productos, con el objetivo de maximizar el rendimiento de sus portfolios.

Existen otras alternativas como el Microsoft Project y el Dotproject que también permiten la gestión de proyectos. Ninguno de estos permite la gestión de proyectos entre empresas. Estos se limitan a los proyectos que se desarrollan en el marco de la empresa.

En general estas herramientas tienen un grupo de inconvenientes, entre estas se pueden mencionar las siguientes:

- La información generada en cada una de las etapas del proyecto, debe adecuarse para transferirse a otros sistemas, generando actividades extras (Martín Santos D., 2007).
- Los resultados del seguimiento y control deben ser trabajados por separado, para generar informes que se puedan utilizar por las herramientas de inteligencia de negocios, para los que la utilicen. (Martín Santos D., 2007).

La UCI ha comenzado a incursionar en el desarrollo de sistemas para la gestión de proyectos, haciendo especial énfasis en los proyectos de gestión gubernamental, como es el caso de CICCV. El principal inconveniente de este sistema es que está desarrollado atendiendo a las necesidades de los proyectos que se realizan entre Cuba y Venezuela.

1.3. Arquitectura de Software

1.3.1. Breve reseña histórica.

La arquitectura de software es relativamente joven aunque se comienza a hablar de ella en los años 60 en algunos círculos de investigación, Edsger Dijkstra es una de las primeras figuras que habla sobre arquitectura de software. Este científico planteaba que todos los sistemas de software debían contar primeramente con una estructura preliminar antes de ser programado. Durante las próximas dos décadas la arquitectura de software fue mirada de muchas formas, todas muy lejanas de la disciplina que es hoy.

[1] Portfolio management o Gestión de cartera se utiliza para seleccionar una cartera de proyectos de desarrollo de nuevos productos para maximizar la rentabilidad o el valor de la cartera de proyectos así como apoyar la estrategia de las mismas

A fines de la década de los 80 y principios de la de los 90,se comienza a ver la arquitectura de software más ligada a la disciplina que se conoce actualmente. Muchos autores coinciden en que uno de los primeros estudios en que aparece la expresión arquitectura de software, en el sentido en que hoy lo conocemos, es sin duda el de Perry y Wolf en "Foundations for the study of software architecture" en el año 1992. A partir de este momento la arquitectura de software es considerada como disciplina por mérito propio, debido a su comportamiento como marco de referencia para satisfacer requerimientos, una base esencial para la estimación de costos y administración del proceso y para el análisis de las dependencias y la consistencia del sistema.

La proliferación de la arquitectura de software provocó que en esta misma época comenzaran a vinculársele términos como los de patrón y estilos e incluso lenguajes de descripción de la arquitectura (ADL, por sus siglas en inglés). Los patrones y los estilos surgen con la idea de fortalecer la reutilización, dado por la necesidad impetuosa de nuevos sistemas con funcionalidades más complejas y en menos tiempo.

1.3.2. ¿Qué es la arquitectura de Software?

Existen disímiles definiciones de arquitectura de software, pero ninguna es totalmente aceptada. Dentro de las definiciones existentes se pueden citar algunas que son muy difundidas en todo el mundo. Una de las más conocidas es la que se plasma en el libro Ingeniería del Software, Un enfoque práctico, la cual plantea que: *"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* (Pressman, 2001)*."*

En la definición planteada anteriormente se refiere a la arquitectura como la estructura que tendrá un sistema de software. Es importante señalar que en la definición anteriormente expuesta se entiende por extremadamente visible a las funcionalidades o servicios que pueda brindar un componente a otros.

En *An Introduction to Software Architecture*, David Garlan y Mary Shaw sugieren que la arquitectura es un nivel de diseño que se centra en aspectos cuando plantea: *"Más allá de los algoritmos y estructuras de datos de la computación; el diseño y la especificación de la estructura general del sistema emergen como una clase nueva de problema. Los aspectos estructurales incluyen la estructura global de control y la organización general; protocolos de comunicación, sincronización y acceso de datos; asignación*

de funciones para diseñar elementos; distribución física, composición de elementos de diseño; ajuste y rendimiento; y selección entre otras alternativas de diseño (Garlan, 1994)*"*

Según RUP la arquitectura es algo más que lo planteado por David Garlan y Mary Shaw, *"es la organización o la estructura de los componentes importantes del sistema que interactúan mediante interfaces, con componentes compuestos de interfaces y componentes cada vez más pequeños.* (IBM, 2003)*"*

A pesar de la cantidad de definiciones, es común entre todos los autores que la misma hace referencia a la estructura del sistema, en cuanto a componentes y la relación entre ellos. De las tantas definiciones se adoptó como definición oficial la que plantea la IEEE Std[2] 1471-2000, la cual expresa que: *"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* (International Organization of Standarization, 2007)*."*

1.3.3. Estilos arquitectónicos

El término estilo surge por la aparición de ciertas regularidades de configuración que aparecieron como respuesta a demandas comunes en los software. A estas respuestas se les da el nombre de estilos por analogía con el uso del término en la arquitectura de edificaciones.

Según Pressman (Pressman, 2001), un estilo describe una categoría de sistema que abarca un conjunto de componentes que realizan una función requerida por el sistema, un conjunto de conectores que posibilitan la comunicación, la coordinación y cooperación entre los componentes, las restricciones que definen como se integran los componentes para conformar el sistema, y los modelos semánticos que facilitan al diseñador el entendimiento de todas las partes del sistema. El estilo arquitectónico es también un patrón de construcción.

Los estilos enlazan los componentes, conectores, configuraciones y restricciones de un sistema. La descripción de la arquitectura se pudiera hacer en lenguaje común pero se recomienda que se utilicen lenguajes de descripción de arquitecturas o lenguajes formales de especificación.

Dentro de los estilos más utilizados en el mundo del software se pueden mencionar:

- Arquitectura Cliente-Servidor.
- Arquitectura basada en Tubería y filtros.
- Arquitectura en Capas.

[2] Std se refiere a estándar.

- Arquitectura Orientada a Objetos.
- Arquitectura Orientada a Componentes.
- Arquitectura Orientada a Servicios (SOA, por sus siglas en ingles)

1.3.3.1. Ejemplos de estilos arquitectónicos

Existen estilos que aunque todavía se usan han quedado un poco obsoletos, producto de la constante evolución de las tecnologías de la información y las comunicaciones (TIC). Existen otras que se usan cotidianamente por las características y las facilidades que le proporcionan a los proyectos de software.

- Arquitectura basada en Tubería y filtros.

En la arquitectura basada en tuberías-filtros, los componentes (filtros) son capaces de filtrar los datos, que interactúan con otros filtros, sólo a través de una interfaz de entrada (recibe los datos) y una de salida (libera los datos filtrados. Estos conjuntos están conectados a los componentes por medio de los conectores, los cuales se les llaman tuberías. Este estilo es lineal y no permite ramificaciones. Las reglas de *tuberías y filtros* han sido específicamente escogidas para mejorar ciertas cualidades del software.

Los componentes más importantes en este estilo son: el origen de los datos, los filtros, las tuberías y los consumidores de los datos.

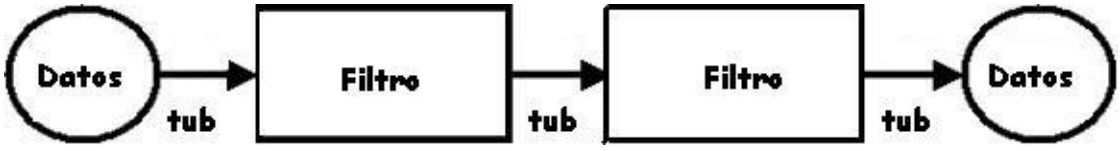


Figura 1: Estructura de filtros y tuberías

Es conveniente usar esta arquitectura cuando en el sistema es necesario realizar muchas transformaciones y estas deben ser flexible y a la vez firmes.

- Arquitectura Cliente Servidor

Esta se puede definir como una arquitectura distribuida en la cual un cliente realiza peticiones a un servidor, el cual devuelve una respuesta a dichas peticiones.

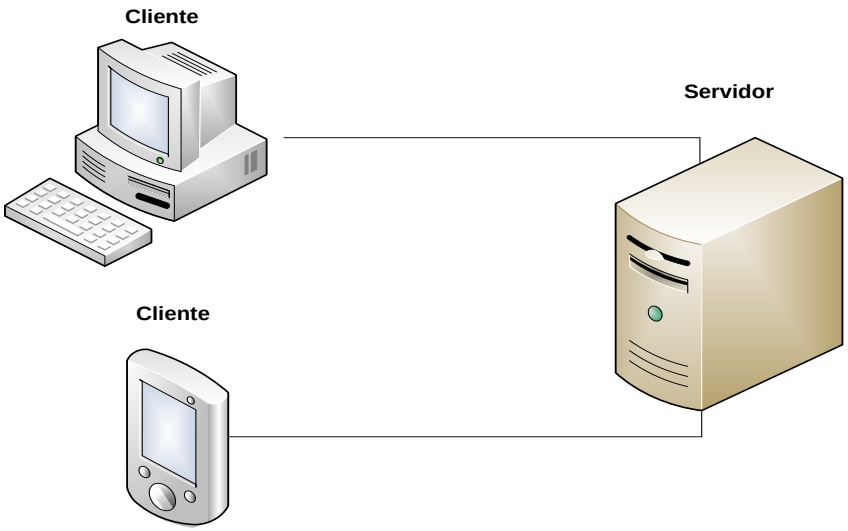


Figura 2: Arquitectura cliente servidor

El Cliente normalmente es el que maneja todas las funciones relacionadas con la manipulación y despliegue de datos, como pudieran ser:

- Administrar la interfaz de usuario.
- Interactuar con el usuario.
- Procesar la lógica de la aplicación y hacer validaciones locales.
- Generar requerimientos de bases de datos.
- Recibir resultados del servidor.
- Formatear resultados.

Todas estas funciones están desarrolladas sobre plataformas que permiten construir interfaces gráficas de usuario (GUI). Otras de las funcionalidades de los clientes es acceder a los servicios distribuidos en cualquier parte de una red.

El servidor es el encargado de atender las peticiones de los clientes sobre los recursos que administra. Generalmente este maneja todas las funcionalidades relacionadas con las reglas del negocio y los recursos de datos.

Las características de esta arquitectura traen consigo un gran número de ventajas como son:

- Bajos costos en hardware para su implementación.
- Permite la integración de sistemas operativos de cualquier índole.
- Al presentar interfaces gráficas orientadas al cliente existe una mayor interacción con el mismo.

- Contribuye además, a proporcionar soluciones locales a las organizaciones, pero permitiendo la integración de la información relevante a nivel global.

En esta arquitectura es necesario tener en cuenta la persistencia de los datos así como la seguridad de los mismos. Esto viene dado por la compartimentación de los recursos en la red.

- Arquitectura en Capas.

Es un estilo en el que se divide el sistema en capas. La principal ventaja de estilo consiste en facilitar la gestión de los cambios en un sistema. Solo basta con trabajar con el nivel que se necesite realizar el cambio sin tener que tocar el resto de los niveles.

Por lo general se divide el sistema en tres capas fundamentales, presentación, lógica de negocio y acceso a datos. Este estilo facilita que cualquier cambio que se realice sobre alguna capa sea transparente respecto a las otras capas que se implementan en el sistema. La abstracción de los desarrolladores de los otros niveles es además otra de las ventajas, estos solo necesitan conocer la interfaz que conecta cada una de las capas.

Este estilo arquitectónico es muy usado, ya que permite una gran escalabilidad en los sistemas que se desarrollan, se debe a que facilita la ampliación de los mismos. Dentro de este estilo se utiliza mucho la arquitectura en tres capaz.

- Capa de presentación: Esta capa es la que permite la interacción con los usuarios de la aplicación. Su función principal es gestionar los datos con el usuario.
- Capa de Negocio: Aquí se establecen todas las reglas del negocio que deben ser cumplidas. Esta interactúa con la capa de presentación para recibir los datos provenientes del usuario, así como para enviar respuestas a dichas peticiones. Además se relaciona con la capa de Datos para gestionar la información que se almacena en los medios de almacenamiento.
- Capa de Datos: En esta reside toda la información referente al negocio. Además es la encargada de la gestión de los mismos.

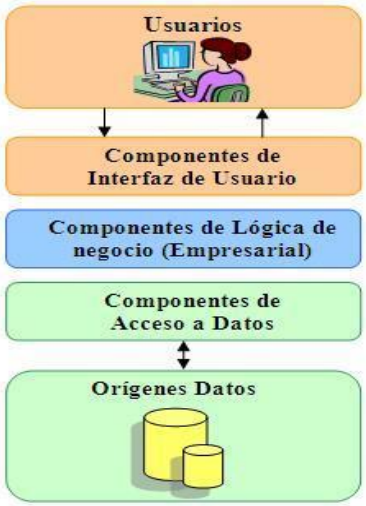


Figura 3: Arquitectura clásica por capas

- Arquitectura Orientada a objetos

En este estilo los componentes son los objetos, o más bien instancias de tipos de datos abstractos. Muchos consideran que en este estilo no se debe considerar la herencia como un conector entre los componentes pero brinda la flexibilidad necesaria para incluir otros conectores e incluso otros estilos arquitectónicos.

Entre las características más relevantes que este estilo tiene se pueden mencionar:

- Los componentes del estilo se basan en los principios del paradigma Orientado a Objetos (encapsulamiento, herencia y polimorfismo): Los objetos y sus relaciones constituyen el centro para el diseño de la arquitectura.
- Las interfaces están separadas de las implementaciones. Por lo general los objetos son transparentes a la implementación de la interfaz, incluso a los mismos objetos con los que se relacionan.

Este estilo tiene muchas ventajas una de las más notorias es la transparencia que brinda en el momento de realizar algún cambio. Para los objetos es totalmente transparente la implementación de los demás por lo que cualquier cambio que se haga en alguno de ellos no afecta a los demás, ni tampoco requiere de mucho esfuerzo.

El principal problema que tiene es que para poder interactuar con otros objetos es necesario conocer la identidad de estos, es decir que si se cambia el nombre de un método o de una clase es necesario que se actualice en cada uno de los objetos que invoquen a estos. Los problemas de efectos

colaterales en cascada es otro de los problemas más notorios ya que el efecto sobre un objeto puede afectar a otros que invoque y así sucesivamente.

- Arquitectura basada en componentes

La reutilización de componentes de software se ha convertido en una gran necesidad, de aquí que surja este estilo. Es uno de los más usados para la producción de software comercial. Por lo general, los componentes terminan siendo subsistemas que tienen una funcionalidad especifica, generalmente aparecen en forma de librerías (DLL), pero no quiere decir que esta sea la única forma de verlos. Existen muchas definiciones para el término de componentes de software.

Según Krutchen, ¨un componente es una parte no trivial, casi independiente y reemplazable de un sistema que cumple una función dentro del contexto de una arquitectura bien definida. Un componente cumple con un conjunto de interfaces y provee la realización física de ellas (Brown, 1998).¨

El *Software Engineering Institute* (SEI) de la Universidad Carnegie Mellon formuló una definición con el propósito de consolidar las diferentes opiniones acerca de lo que debía ser un componente de software: "*un componente es una implementación opaca de funcionalidad, sujeta a composición por terceros y que cumple con un modelo de componentes* (Bashman, 2000)*.*"

La mayoría de las definiciones giran sobre la idea de que es la implementación de una funcionalidad de un sistema de software, que puede ejecutar varias funcionalidades liberadas. Estos proveen una interfaces para visualizar las funcionalidades que implementan.

La reutilización de los componentes es una de las características más importantes de este estilo. Pero estos deben tener características bien definidas como:

- *Identificable*: un componente debe tener una identificación clara y consistente que facilite su catalogación y búsqueda en repositorios de componentes.
- *Accesible sólo a través de su interfaz*: el componente debe exponer al público únicamente el conjunto de operaciones que lo caracteriza (interfaz) y ocultar sus detalles de implementación. Esta característica permite que un componente sea reemplazado por otro que implemente la misma interfaz.
- *Servicios invariantes*: las operaciones que ofrece un componente, a través de su interfaz, no deben variar. La implementación de estos servicios puede ser modificada, pero no deben afectar la interfaz.

- *Documentado*: un componente debe tener una documentación adecuada que facilite su búsqueda en repositorios de componentes, evaluación, adaptación a nuevos entornos, integración con otros componentes y acceso a información de soporte. (Puebla, 2008)

La facilidad de reutilización de software es una ventaja que lleva a este estilo a un escalón superior sobre otros estilos tradicionales como el orientado a objetos. El desarrollo de la pruebas es otra de las ventajas en este estilo, ya que facilita la prueba de cada componente antes de hacer las pruebas para el sistema completo. Cuando existe un débil acoplamiento entre componentes, el desarrollador es libre de gestionar componentes según sea necesario, sin afectar otras partes del sistema se logra que el mantenimiento sea más sencillo.

La desventaja principal de este estilo radica que en que si sale una versión más actualizada de los componentes que se utilizan puede que haya que implementar nuevamente los otros que lo utilicen.

- Arquitectura orientada a Servicios

La Arquitectura basada en Servicios (SOA, por sus siglas en ingles) está diseñada para establecer una integración con otras aplicaciones, independientemente del modo de acceso a las funcionalidades que esta brinde. Normalmente cuando se habla de SOA se trata el término de servicios web, lo cual no es necesario para implementar una arquitectura SOA.

Los servicios para comunicarse entre si se basan en una definición formal independiente de la plataforma y del lenguaje de programación, como el lenguaje de descripción de servicios web *(*WSDL, por sus siglas en inglés*)*. La definición de interfaces permite encapsular la implementación, lo que la hace independiente del desarrollador, del lenguaje de programación o de la tecnología de desarrollo. (Puebla, 2008)

SOA permite mejorar la toma de decisiones ya que facilita unificar la información con mejor calidad, no existen limitaciones con las tecnologías de la información y facilita una mayor capacidad de respuesta a los clientes ya que las aplicaciones son más flexibles, y seguras.

El comportamiento de la red constituye unos de los problemas más relevantes que proporciona esta arquitectura. Ejemplo de ello es el tamaño de los mensajes respecto a estado de la red, mala configuración y la inseguridad en las comunicaciones.

Después de haber estudiado cada uno de los estilos anteriormente mencionados se ha decidido utilizar una arquitectura por capas tradicional, tres capas. Esta decisión viene dada por la escalabilidad que este le proporciona a los sistemas de software. La arquitectura por capas facilita el desarrollo en

paralelo, logrando así un mejor aprovechamiento del tiempo de desarrollo. La capacidad de mantenimiento que brinda este estilo es otro de los motivos que favorecen la decisión.

1.3.4. Patrones Arquitectónicos

En el desarrollo de proyectos de software existen muchos problemas que son comunes, como respuesta a esta situación surgen los patrones arquitectónicos. Estos patrones son la combinación problema-solución, para una situación determinada. Muchos han sido los patrones arquitectónicos que se han creado desde los inicios de la arquitectura de software.

Es interesante referirse a la definición de patrones que se da en el libro Patterns of Enterprise Application Architecture, del cual se cita: *"Cada patrón describe un problema que ocurre una y otra vez en nuestro medio ambiente y, a continuación, se describe el núcleo de la solución a ese problema, de tal manera que puede utilizar esta solución un millón de veces más, sin hacerlo de la misma manera dos veces* (Fowler, 2002)*."*

Existen muchos patrones arquitectónicos, desde los que tratan problemas de manejo de Datos, presentación de los datos hasta el manejo de sesiones y concurrencia. En el libro "Patterns of Enterprise Application Architecture" se agrupan los patrones, dependiendo de la naturaleza problema-solución, en 10 grupos, de los cuales se pueden mencionar:

- Layering: Patrones para el apoyo de arquitecturas por capas.
- Domain Logic Organization: Formas de organizar los objetos del dominio.
- Mapping to Relational Databases: Se relaciona con la comunicación entre la lógica del dominio y los repositorios de datos.
- Web Presentation: La presentación Web es uno de los desafíos que han tenido que sortear en los últimos años las aplicaciones empresariales. Los clientes delgados Web proveen muchas ventajas, siendo una de las principales la facilidad de distribución ya que no es necesario que se instale en las estaciones clientes.
- Concurrency: Manejo de la concurrencia. Las aplicaciones actuales basadas en tecnologías Web tienen grandes necesidades de gestión de la concurrencia.
- Sesion State: Patrones para el manejo de la sesión en servidores Web.
- Distribution Strategies: Distribución de objetos en múltiples emplazamientos, basada en conocimientos empíricos de clientes.

Dentro de esta clasificación existe un sin número de patrones que se utilizan a diario y sin embargo se desconoce su existencia, debido a que muchos frameworks y herramientas abstraen a los

desarrolladores del problema. Dentro de estos patrones es necesario hacer referencia a algunos que por sus características es necesario tener en cuenta para dar solución al problema que se describió.

1.3.4.1. Patrones más usados

Dentro de estos grupos existen un grupo de patrones que tienen gran uso en los sistemas de hoy, un ejemplo de ellos es el modelo vista controlador (MVC) el cual está asociado al grupo de patrones de presentación Web. Un ejemplo de este es el módulo Spring MVC del marco de trabajo Spring.
El patrón Mapeo de los Datos, el cual entra en el grupo de los patrones de manejo de datos, es otro de los que más se utilizan; un ejemplo de ello son los marcos de trabajo Hibernate (para la tecnología JEE[3]) y Doctrine (para PHP[4]).

- Patrón Modelo Vista Controlador

El patrón MVC separa el negocio de la presentación y las acciones de almacenamiento de los datos. Para lograr esto se establecen tres clases diferentes, las cuales son:

- Modelo: Esta maneja todo lo relacionado con la lógica del negocio.
- Vista: Maneja la visualización de la información.
- Controlador: Estos objetos son responsables de procesar las entradas del usuario en forma de peticiones HTTP.

La vista y el controlador dependen del modelo, el cual no depende de las otras clases. Esta separación permite independizar el modelo de la representación visual. En ocasiones la vista y el controlador aparecen fusionados en una sola. Esta característica facilita el cambio de las vistas o las interfaces del sistema gracias al bajo acoplamiento que existe entre el modelo y la vista.

[3] La Java 2 Platform, Enterprise Edition (J2EE) define el estándar para el desarrollo de aplicaciones empresariales.

[4] PHP es un lenguaje interpretado de alto nivel embebido en páginas HTML y ejecutado en el servidor.

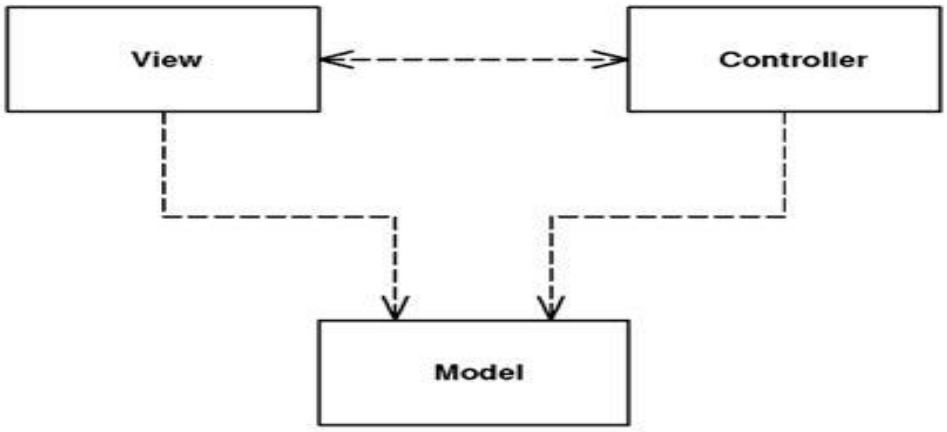


Figura 4: Modelo Vista Controlador

- Patrón mapeo de datos.

Por lo general el diseño de la base de Datos (BD) no guarda relación con el modelo de los datos, esto en parte viene dado por la limitante del modelo relacional de no permitir relaciones de herencia. El lenguaje que tratan los objetos tampoco es el mismo conque las BD tratan las entidades. Para dar solución a esto surge el patrón de mapeo de los Datos.

El mapeo de los datos es una capa de software que separa los objetos en memoria de la base de datos. Su responsabilidad consiste en la transferencia de datos entre los dos y también para aislarlos unos de otros. Los objetos no tienen por qué saber que hay una base de datos para almacenar la información de ellos, no necesitan ningún código de interfaz SQL para hacerlo, y ciertamente ningún conocimiento del esquema de base de datos.

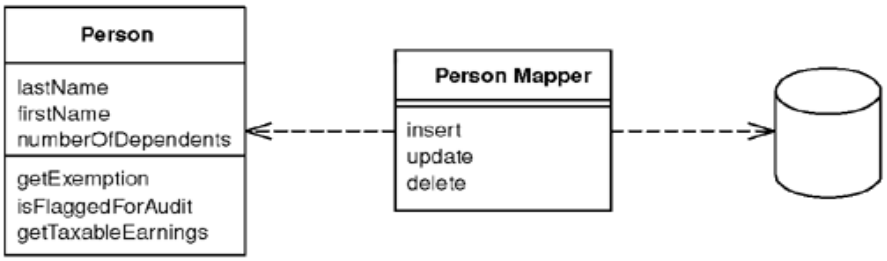


Figura 5: Patrón Mapeo de Datos

Se propone el uso de los patrones MVC y mapeo de datos por la robustez y la reusabilidad que estos le darían al sistema. En el caso del MVC permitiría que la lógica del negocio sea más reusable ya que estarían separadas la presentación y la lógica del negocio.
El constante cambio en los gestores de Bases de Datos es algo que golpea fuertemente los sistemas actualmente. El uso del patrón mapeo de datos, permite abstraer los componentes del gestor de bases de datos, haciendo los mismos más reutilizables. Permite además minimizar al máximo el mantenimiento de los componentes favoreciendo así la gestión de los cambios.

1.4. Metodologías de Desarrollo.

El desarrollo de software es riesgoso y muy difícil de controlar, pero si no se aplica una metodología que guíe el desarrollo del mismo lo que se obtendrá la final es insatisfacción por parte de los clientes y por ende el equipo de desarrollo. Una metodología de desarrollo de software es un proceso en el que se define "quién" está haciendo "qué", "cómo" y "cuándo".
Con ánimos de buscar una metodología que se ajuste al medio en que se desarrollará el proyecto se analizaron algunas de las más usadas en el mundo. Dentro de estas están Programación extrema (XP, por sus siglas en inglés), Proceso Unificado de Rational (RUP, por sus siglas en inglés) y Microsoft Solution Framework (MSF).

- XP

Es una de las metodologías de desarrollo de software más exitosas en la actualidad, utilizada para proyectos de corto plazo, un equipo de desarrollo y un intervalo de tiempo para la entrega pequeños. La metodología consiste en una programación rápida o extrema, cuya particularidad es tener como parte del equipo, al cliente, pues es uno de los requisitos para llegar al éxito del proyecto.

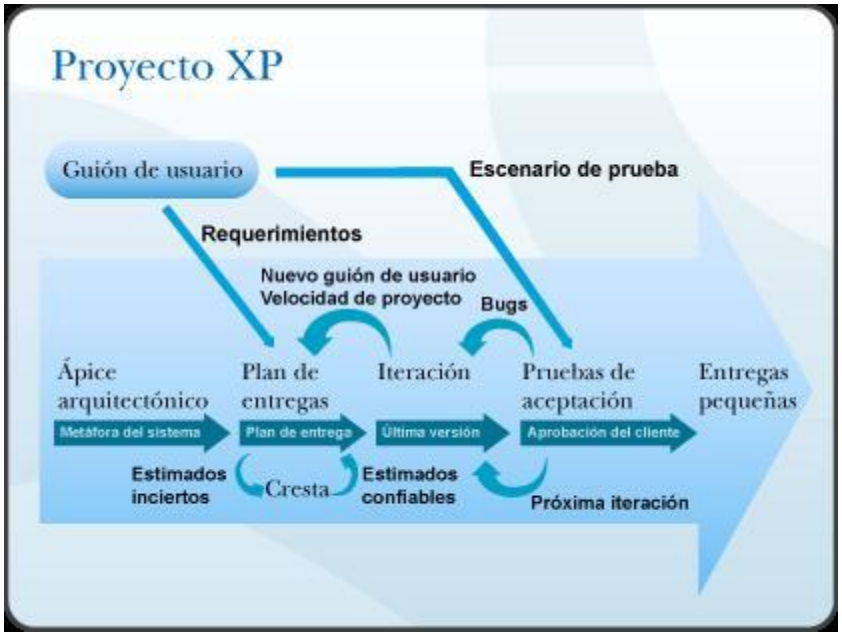


Figura 5: Metodología Extreme Programing

Esta metodología está basada en:

- Pruebas Unitarias: se basa en las pruebas realizadas a los principales procesos, de manera tal que se pueda prevenir posibles errores que en el futuro pudieran ocurrir.
- Refabricación: se basa en la reutilización de código, para lo cual se crean patrones o modelos estándares, siendo más flexible al cambio.
- Programación en pares: Una particularidad de esta metodología es que propone la programación en pares, la cual consiste en que dos desarrolladores participen en un proyecto en una misma estación de trabajo. Cada miembro lleva a cabo la acción que el otro no está haciendo en ese momento.

Para un desarrollo exitoso con esta metodología es necesario que exista:

- La comunicación, entre los clientes y los desarrolladores.
- La simplicidad, al desarrollar y codificar los módulos del sistema.
- La retroalimentación, concreta y frecuente del equipo de desarrollo, el cliente y los usuarios finales. (Sanchez, 2004)

- MSF

Esta es una metodología flexible e interrelacionada con una serie de conceptos, modelos y prácticas de uso, que controlan la planificación, el desarrollo y la gestión de proyectos tecnológicos. MSF se centra en los modelos de proceso y de equipo dejando en un segundo plano las elecciones tecnológicas.

Figura 7: Metodología MSF

MSF tiene las siguientes características:

- Adaptable: es parecido a un compás, usado en cualquier parte como un mapa, del cual su uso es limitado a un específico lugar.
- Escalable: puede organizar equipos tan pequeños entre 3 o 4 personas, así como también, proyectos que requieren 50 personas a más.
- Flexible: es utilizada en el ambiente de desarrollo de cualquier cliente.
- Tecnología Agnóstica: porque puede ser usada para desarrollar soluciones basadas sobre cualquier tecnología.

MSF se compone de varios modelos encargados de planificar las diferentes partes implicadas en el desarrollo de un proyecto: Modelo de Arquitectura del Proyecto, Modelo de Equipo, Modelo de

Proceso, Modelo de Gestión del Riesgo, Modelo de Diseño de Proceso y finalmente el modelo de Aplicación. (Sanchez, 2004)

- RUP

El Proceso Unificado 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 y diferentes tamaños de proyecto.

RUP está basado en componentes[5], lo cual quiere decir que el sistema de software está formado por componentes interconectados a interfaces[6] bien definidas. RUP utiliza UML (Lenguaje Unificado de Modelado) para la esquematización del software en cada una de las fases del desarrollo. Los creadores de esta metodología proponen el uso de UML como lenguaje de modelado, pues este constituye una parte esencial dentro de esta metodología.

Si hubiese que definir cuáles son las características fundamentales que diferencian a RUP de otras metodologías sin lugar a dudas habría que mencionar que esta es dirigida por casos de uso, centrado en la arquitectura e iterativo e incremental. (Ivar Jacobson, 2000)

- Dirigido por casos de uso: Un caso de uso es un fragmento de funcionalidad del sistema que proporcional al usuario un resultado importante. Estos representan los requisitos, funcionales, y los usuarios relacionados con ellos, del futuro sistema. Además guían su diseño, implementación y pruebas. (Ivar Jacobson, 2000)
- Centrado en la arquitectura: RUP además de utilizar los Casos de Uso para guiar el proceso, presta especial atención al establecimiento temprano de una buena arquitectura que no se vea fuertemente impactada ante cambios posteriores durante la construcción y el mantenimiento. (Ivar Jacobson, 2000)
- Iterativo e incremental: RUP propone dividir el proyecto en partes más pequeñas o mini proyectos. Esto permite lograr un equilibrio entre Casos de Uso y arquitectura durante cada mini proyecto, así durante todo el proceso de desarrollo. Cada mini proyecto se puede ver como una iteración (un recorrido a lo largo de todos los flujos de trabajo fundamentales) del cual se obtiene un incremento que produce un crecimiento en el producto. (Ivar Jacobson, 2000)

[5] Parte física y reemplazable de un sistema que se ajusta y proporciona la realización de un conjunto de interfaces.

[6] Una interfaz es una colección de operaciones que son utilizadas para especificar un servicio a un componente.

RUP propone cuatro fases para todo el proceso, dentro de estas se realizan varios flujos de trabajo, en dependencia del esfuerzo en cada fase. Esta metodología cuenta con nueve flujos de trabajo, seis de ingeniería y tres de soporte. En la figura siguiente se muestra cómo varía el esfuerzo asociado a las disciplinas según la fase en la que se encuentre el proyecto.

Los flujos de trabajo propuesto son los siguientes:

- Flujos de ingeniería:
  - Modelamiento del negocio: Se describe el negocio, define las actividades a automatizar y el personal que participa en ellas.
  - Levantamiento de requisitos: Se define que debe hacer el sistema, identificando las funcionalidades que desea el usuario y las restricciones que esto impone.
  - Análisis y diseño: de describe como el sistema debe cumplir las funcionalidades previstas y las restricciones, que detalladamente indicando que es lo que se va a programar.
  - Implementación: se comienza a desarrollar el sistema en términos de clases y objetos.
  - Pruebas: identificar y eliminar los errores y defectos surgidos durante el proceso de desarrollo.
  - Despliegue: se pone un *release* del producto en manos de los usuarios finales.
- Flujos de soporte:
  - Gestión de configuración y cambios: se describe como llevar a cabo el control de los artefactos generados en el proyecto, de manera que se mantenga al tanto a todo el equipo con las últimas versiones.
  - Gestión de proyectos: engloba las actividades encaminadas a lograr un producto que realmente satisfaga las expectativas del cliente.
  - Gestión de entornos: contiene las actividades que describen procesos y herramientas que empleará el equipo de trabajo y el proceso para llevar a cabo dichas actividades en la organización (Ivar Jacobson, 2000).

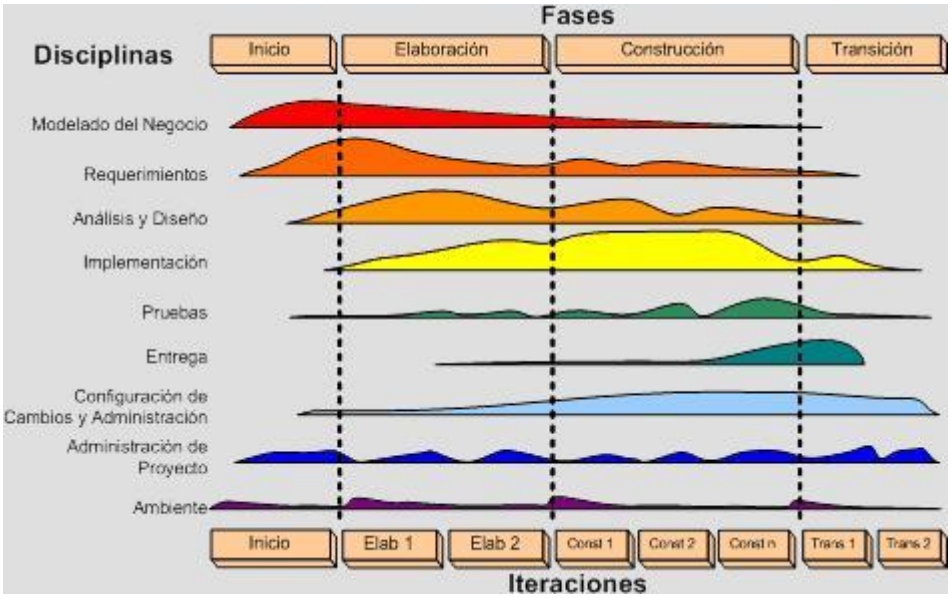


Figura 8: Proceso de desarrollo según RUP.

1.5. Lenguajes de programación.

Los sistemas computacionales no se escriben en lenguaje natural (como escribimos o hablamos), por lo que se necesita un lenguaje que permita la comunicación entre la computadora y el desarrollador. Al lenguaje, en que se escriben los programas y que la computadora entiende se le llama lenguajes de programación.

Los primeros lenguajes de programación eran inentendibles. Solo algunos pocos especialistas en la rama podían hacer pequeños programas debido a la complejidad de escribir en estos lenguajes. Con el tiempo y gracias al continuo desarrollo los lenguajes que iban surgiendo se acercaban más al lenguaje natural. Los compiladores son los que se encargan de traducir estos lenguajes al lenguaje de máquina.

El proceso de compilación lo primero que se analiza es lo que se escribe en el lenguaje que se especifica (Java, PHP, Python, etc). Luego se analiza si la estructura sintáctica de lo que se ha escrito es correcta para proceder a la compilación. El resultado de esta compilación es la traducción del programa escrito al lenguaje de máquina. (Programación de computadoras. Introducción al paradigma OO, 2007)

Desde el surgimiento de los lenguajes de programación han sido muchos los que han surgido, cada uno con sus características especificas. Cada lenguaje tiene ventajas y desventajas que lo hacen más o menos potente ante una necesidad de software determinada, por lo que no se puede afirmar que exista un lenguaje que cubra todas las necesidades requeridas por un software. De aquí que la

importancia de que a la hora de seleccionar el lenguaje de programación para un proyecto de software se haga un análisis de estos a partir de las necesidades de software.

Son muchos los lenguajes que pudieran satisfacer las necesidades del proyecto, pero se decidió solo realizar un análisis de algunos de los que en la UCI se utilizan. Entre estos se pueden mencionar:

- Python.
- C#.
- C++.
- Java.
- PHP.

❖ Python

Python ha sido diseñado por Guido van Rossum y está en un proceso de continuo desarrollo por una gran comunidad de desarrolladores. Aproximadamente cada seis meses se hace pública una nueva versión de Python. Esos cambios no son radicales, sino que se le hacen mejoras, tratando de mantener la compatibilidad con versiones anteriormente desplegadas. Son muchas las características de este lenguaje que lo hacen tan potente, entre las que se pueden mencionar:

- Lenguaje de programación multi-paradigma: Permite varios estilos de programación como la Orientada a Objetos y la estructural.
- Portabilidad: Python se ejecuta en muchos sistemas operativos como Windows, GNU/Linux y Mac .(Dayley, 2006)
- Integración: Este puede integrarse con muchas plataformas. Un ejemplo es Jython que permite usar Python en cualquier plataforma de Java. Además facilita que los programadores de este lenguaje puedan acceder a las librerías de .NET, a través de IronPython. (Dayley, 2006)
- Fácil: Es fácil el aprendizaje. La sencillez de las sintaxis brinda facilidad para crear y debuguear programas. (Dayley, 2006)
- Potente: Existe mucho código escrito en Python como acceso a datos, edición de audio/video, Interfaces de usuarios, así como desarrollo web. (Dayley, 2006)

A pesar de las facilidades que este lenguaje proporciona, es criticado por algunos desarrolladores. Uno de los aspectos más criticados es que este es débilmente tipado, es decir, una variable que se declara puede tomar lo mismo un valor numérico que una cadena de caracteres.

Otros de los puntos debatidos por los desarrolladores es la importancia que se le atribuye a la identación del código en los programas escritos en este lenguaje. La identación puede causar

molestias a la hora de desarrollar pero sin embargo pudiera ser muy importante a la hora de aprender a programar.

- C++

El lenguaje C++ es una mejora al lenguaje C. En este se le agregan funcionalidades que C no tenía, tales como: POO, excepciones, sobrecarga de operadores, templates o plantillas. Existen ciertas características que lo posiciona entre uno de los más potente para el desarrollo de software. Entre las características más notables se pueden mencionar:

- Programación orientada a objetos: Esta característica va acorde con el paradigma OO, fue uno de los primeros lenguajes que permitió programar según este paradigma.
- Portabilidad: Puede compilar prácticamente el mismo código C++ en casi cualquier tipo de ordenador y sistema operativo sin realizar ningún cambio, esto se debe a que muchos sistemas operativos lo usan para implementar muchas funcionalidades.
- Programación modular: En C++ el código puede ser compuesto por diversos archivos de código fuente que se compilan por separado y luego unidos. Permite ahorrar tiempo ya que no es necesario recompilar la aplicación completa al realizar un solo cambio.
- Velocidad: El código resultante de una compilación de C++ es muy eficiente, de hecho, debido a su dualidad como lenguajes de alto nivel y de bajo nivel. (Souli, 2009).

A pesar de estas características este lenguaje tiene muchos problemas que pueden provocar molestias para los programadores. Una de las más notables es que el uso de la memoria queda completamente en manos del programador. El manejo de punteros y memoria permite un mejor control de la memoria y una buena administración de recursos de computadora, pero también puede llevar a colapsar la aplicación ya que todo el manejo de los mismos queda en manos del programador.

- C#

La sintaxis y estructuración de C# es muy parecida a la de C++ o Java, puesto que la intención de Microsoft es facilitar la migración de códigos escritos en estos lenguajes a C# y facilitar su aprendizaje a los desarrolladores habituados a ellos. Entre las características más reconocidas de C# se encuentran:

- Sencillez: Al ser el lenguaje nativo de Microsoft, elimina muchos elementos que son innecesarios para .Net. Un ejemplo de ello es que no utiliza marcos, tampoco la herencia múltiple, elimina el uso de operadores tales como (::).

- Modernidad: C# incorpora en el propio lenguaje elementos que a lo largo de los años ha ido demostrándose que son muy útiles para el desarrollo de aplicaciones como la instrucción foreach, que permite recorrer colecciones con facilidad**.**
- Orientación a objetos: Como todo lenguaje de programación de propósito general existente en la actualidad. En lo referente a la encapsulación C# añade un cuarto modificador llamado internal, que puede combinarse con protected e indica que al elemento a cuya definición precede sólo puede accederse desde su mismo ensamblado. Respecto a la herencia C# sólo admite herencia simple de clases.
- Gestión automática de memoria: C# dispone de un recolector de basura, por lo que no es necesario incluir instrucciones de destrucción de objetos.
- Sistema de tipos unificado: En C# todos los tipos de datos que se definan siempre derivarán, aunque sea de manera implícita, de una clase base común llamada System.Object, por lo que dispondrán de todos los miembros definidos en ésta clase.
- Eficiente: En principio, en C# todo el código incluye numerosas restricciones para asegurar su seguridad y no permite el uso de punteros aunque se puede violar estas restricciones mediante el modificador unsafe. (Seco, 2006)

- Java.

Java es uno de los lenguajes más usados para el desarrollo de software. Este ofrece toda la funcionalidad de un lenguaje potente, pero sin las características menos usadas y más confusas de lenguajes, como los punteros de C++. Dentro de las características más importantes de este se puede mencionar:

- Orientado a Objetos: Como la mayoría de los lenguajes, este cumple con el paradigma de la orientación a objetos.Posee una arquitectura neutral: La compilación de los software hechos en java es completamente independiente de la arquitectura de la maquina donde se ejecute. Para java lo único que es verdaderamente dependiente del sistema es la Máquina Virtual de Java (JVM) y las librerías fundamentales que sean usadas. La neutralidad de la arquitectura mide en gran medida el grado de portabilidad de este lenguaje.
- Seguro: Se elimina el uso de los punteros con el objetivo de eliminar el uso indebido de recursos en memoria. Además la maquina virtual se encarga de verificar que el software no posea fragmentos de código ilegal (código que falsea punteros, viola derechos de acceso sobre objetos o intenta cambiar el tipo o clase de un objeto) antes de ejecutarlo.

- Interpretado y compilado a la vez: Java es compilado, en la medida en que su código fuente se transforma en una especie de código máquina (bytecodes), semejantes a las instrucciones de ensamblador. Por otra parte, es interpretado, ya que los bytecodes se pueden ejecutar directamente sobre cualquier máquina a la cual se haya instalado la maquina virtual de java.

❖ PHP

PHP es un lenguaje interpretado de alto nivel embebido en páginas HTML y ejecutado en el servidor. El código PHP es interpretado en el servidor Web y genera código HTML y otro contenido que el visitante puede ver. PHP es considerado por muchos uno de los lenguajes más usados para sistemas web a través de la arquitectura LAMP (Linux + Apache + MySQL + PHP), aunque las cosas han cambiado desde que MySQL fue adquirida por Sun Mycrosystems. Dentro de sus características más notorias se pueden ver:

- Alto rendimiento: PHP es muy eficiente, mediante el uso de un único servidor, puede servir millones de accesos al día.
- Interfaces para una gran cantidad de sistemas de base de datos diferentes: Dispone de una conexión propia a todos los sistemas de base de datos. Además de MySQL, puede conectarse directamente a las bases de datos de PostgreSQL, mSQL, Oracle, dbm, filepro, Hyperwave, Informix, InterBase y Sybase, entre otras. El uso de ODBC (del inglés *Open Database Connectivity Standard*, Estándar de conectividad abierta de base de datos) permite establecer una conexión a cualquier base de datos que suministre un controlador ODBC.
- Bibliotecas incorporadas para muchas tareas Web habituales: Como se ha diseñado para su uso en la Web, PHP incorpora una gran cantidad de funciones integradas para realizar útiles tareas relacionadas con la Web. Puede generar imágenes GIF al instante, establecer conexiones a otros servicios de red, enviar correos electrónicos, trabajar con *cookies* y generar documentos PDF, todo con unas pocas líneas de código.
- Bajo coste: PHP es gratuito.
- Facilidad de aprendizaje y uso: La sintaxis de PHP se basa en otros lenguajes de programación, principalmente en C y Perl.
- Portabilidad: PHP está disponible para una gran cantidad de sistemas operativos diferentes. Se puede escribir código PHP en todos los sistemas operativos gratuitos del tipo Unix, como Linux y FreeBSD, versiones comerciales de Unix, como Solaris e IRIX o en las

diferentes versiones de Microsoft Windows. Su código funcionará sin necesidad de aplicar ninguna modificación a los diferentes sistemas que ejecute PHP.

- Acceso al código abierto: Dispone de acceso al código fuente de PHP. A diferencia de los productos comerciales y de código cerrado, si desea modificar algo o agregar un elemento al programa, puede hacerlo con total libertad. No necesitará esperar a que el fabricante publique parches, ni tendrá que preocuparse porque el fabricante cierre sus puertas o decida abandonar el producto.

A pesar de las facilidades de estos lenguajes existen otros factores que se debe tener en cuenta para seleccionar un lenguaje de programación para un proyecto de software. La experiencia acumulada por los especialistas de un proyecto, las licencias para el desarrollo de software, disponibilidad de documentación y herramientas que permitan un desarrollo rápido y con mejor calidad son algunos de estos factores que no se pueden obviar.

Por lo general, cuando se inicia un proyecto de software se hace necesario capacitar al personal en determinados temas. Por eso a la hora de afrontar un nuevo proyecto es necesario que se analice si el lenguaje de programación que dominan los desarrolladores cubre con las necesidades del proyecto, para de esta forma ganar en tiempo. Para el PGG este factor si influye en gran medida, debido a la similitud de sus proyectos y los resultados alcanzados. Viendo este factor el candidato podría ser java.

Las licencias para el desarrollo sobre estos lenguajes es factor muy importante y delicado a tener en cuenta.

De los lenguajes anteriormente descritos C# es el único que pudiera traer problemas. Aunque existen alternativas que permiten su uso, estas no permiten explotar al máximo las características de este lenguaje. Solo con decir que es el lenguaje nativo de la plataforma Microsoft .NET, y tener sus herramientas privativas, costosas y sobre el sistema operativo Windows, es suficiente para darse cuenta de las limitantes para Cuba y por ende para la UCI.

Analizando las características de todos estos posibles lenguajes y teniendo en cuenta los factores que anteriormente se mencionan se ha decidido usar java como el lenguaje de programación para el desarrollo de este sistema. Esto está condicionado, además de las características mencionadas, por la experiencia del PGG en este lenguaje y su plataforma en general. Las herramientas existentes para esta plataforma, los marcos de trabajo existentes y la documentación de los mismos ratifican la decisión.

1.6. Lenguaje Unificado de Modelado.

El Lenguaje Unificado de Modelado (UML) es un lenguaje para visualizar, especificar, construir y documentar los artefactos de un sistema software. Es un estándar del Grupo de administración de objetos (OMG por sus siglas en inglés). Está compuesto por diversos elementos gráficos que se combinan para conformar diagramas. Debido a que el UML es un lenguaje, cuenta con reglas para combinar tales elementos.

La finalidad de los diagramas es presentar diversas perspectivas de un sistema, a las cuales se les conoce como modelo. El modelo UML de un sistema es similar a un modelo a escala de un edificio junto con la interpretación del artista del edificio. Es importante destacar que un modelo UML describe lo que supuestamente hará un sistema, pero no dice cómo implementar dicho sistema. (Shcmuller, 2000).

RUP propone el uso de este lenguaje de modelado para la descripción de los artefactos propuestos por sus flujos de trabajo en cada una de las distintas fases. El surgimiento de este lenguaje casi a la par de RUP y por los mismos creadores hace que fusionen perfectamente, aunque no obliga dicha fusión.

1.7. Herramientas

1.7.1. Marcos de Trabajo

Un Marco de trabajo o Framework define una arquitectura adaptada a las particularidades de un determinado dominio de aplicación, definiendo de forma abstracta una serie de componentes y sus interfaces, y estableciendo las reglas y mecanismos de interacción entre ellos (Perera, 2007).Estos pueden incluir soporte de programas, bibliotecas y un lenguaje interpretado entre otros software para ayudar a desarrollar y unir los diferentes componentes de un proyecto. El objetivo fundamental de estos es facilitar el desarrollo de software, permitiendo así minimizar el tiempo de desarrollo o invertir más tiempo en los procesos ingenieriles.

Existen muchos marcos de trabajo que apoyan el desarrollo de aplicaciones en Java. Esta disponibilidad esta propiciada por el carácter libre u open source de estos. A continuación se hace una propuesta de marcos de trabajo que se seleccionaron para la solución propuesta.

- Spring MVC

Spring MVC es un módulo del marco de trabajo Spring. Como su nombre lo indica este módulo implementa el patrón arquitectónico MVC. Este es una de las alternativas a Struts, el primer marco de trabajo en implementar MVC para java. Dentro de las características más notables de este están:

- Configuración sencilla y rápida, incluyendo referencias fáciles a través de los contextos, tales como controladores web a objetos del negocio y validadores.
- Adaptabilidad, la no injerencia. Permite usar cualquier controlador de subclase que se necesite para una situación determinada, en lugar de derivados de un único controlador para todo.
- Una simple pero potente biblioteca de etiquetas JSP conocida como "Spring tag library", que brinda soporte a características tales como el enlace de datos y temas. La costumbre de etiquetas para permitir la máxima flexibilidad en términos del código de marcas.
- Las "JSP form tag library" que presentó spring 2.0 permite crear formas más sencillas dentro de las JSP.
- Beans cuyo ciclo de vida es el alcance de la actual solicitud HTTP o sesión HTTP. Esto no es una característica específica de spring MVC en sí, sino más bien de la WebApplicationContext container (s) que utiliza spring MVC.

Existen otras alternativas de marcos de trabajo similares a este, como JSF, Struts y trapestry.

- Java Server Faces (JSF) es un estándar de la Sun Microsystems para la capa de presentación en las aplicaciones web. Forma parte de la especificación J2EE y se crea como una evolución natural de los marcos de trabajo actuales, hacia un sistema de componentes (Juan Medin Piñeiro, 2006).
- Aunque Struts abrió el camino hacia MVC en Java para desarrollos web hoy no es muy bien visto desde el punto de vista de la flexibilidad. Existen otros marcos de trabajos que comparten las mismas potencialidades de este pero con más elegancia potencia y flexibilidad.
- Tapesrtry es uno de los marcos de trabajo más potentes y complejos existentes para esta capa, pero la curva de aprendizaje muy pronunciada y no dispone de una gran comunidad ni la documentación que tiene JSF o Spring MVC.

La decisión de utilizar Spring MVC está condicionada, además de sus características, por la experiencia del PGG con este y a su vez es una de las propuestas arquitectónicas de la UCI para los proyectos que se desarrollan sobre la plataforma JEE.

- DOJO

DOJO es una herramienta open source para el trabajo con DHTML (HTML[7] dinámico) escrito en JavaScript. Esta herramienta se compone de un conjunto de librerías al estilo Java. Se centra en abstraer al desarrollador de las complejidades del DHTML y de las discrepancias existentes entre navegadores, que hacen que el código JavaScript a utilizar sea diferente. Otras de las características fundamentales de este son las siguientes:

- Maneja incompatibilidades entre navegadores.
- Soporta AJAX.
- Oculta el manejo del XMLHttpRequest.
- Sistema de eventos orientado a aspectos.

Los principales problemas con Dojo que está todavía en sus inicios (la versión 1.0 fue liberado sólo en febrero de 2008) y la documentación disponible es todavía algo limitada. Pero para el caso de la UCI existe un grupo de soporte que garantiza la preparación necesaria para el desarrollo de proyectos con ella. (Draper, 2008).

Existen otras herramientas, que facilitan la implementación de la presentación de los sistemas web, como es el caso de ICEfaces.

ICEFaces es un marco de trabajo de código abierto. Permite incluir aplicaciones web con Ajax, abstrayendo al programador, esto se debe a que una vez que se agrega el componente, el marco de trabajo genera el código Ajax automáticamente. Además este no necesita agregarle plugins al navegador o applets para que los componentes se visualicen en el navegador. Dentro de las principales ventajas se pueden ver las siguientes:

- ✓ Soporta una gran cantidad de servidores web, como el apache tomcat.
- ✓ Provee plugins para varios IDEs como eclipse y netbeans.
- ✓ Integra elementos de seguridad como la prevención de código malicioso (inyección de código SQL), además es efectivo en la prevención de fallos en los submits de los formularios.(ICEfaces, 2007)

Optar por ICEfaces sería subutilizar Spring MVC ya que ICEfaces está basado en JSF el cual se analizó anteriormente. Otra posible solución sería usar EXT 2.0. Este marco de trabajo es muy bueno y

[7] HTML, siglas de HyperText Markup Language (Lenguaje de Marcas de Hipertexto), es el lenguaje de marcado predominante para la construcción de páginas web.

en el PGG se tiene buena experiencia con el, pero el Centro de Consultoría de la UCI propone el uso de DOJO para las aplicaciones que se desarrollen sobre la plataforma J2EE, brindando además soporte a este, por lo que se propone su uso.

- Spring

Spring es un marco de trabajo usado en la capa de negocio. Proporciona un conjunto de librerías que facilitan el desarrollo de las aplicaciones. Entre sus potencialidades está el contenedor de inversión de control, la introducción de aspectos, plantillas de utilidades para Hibernate, iBatis y JDBC así como la integración con JSF.

Spring permite un acoplamiento entre los diferentes componentes de la aplicación, pero sin crear dependencias alguna entre ellos, apenas con él mismo. La dependencia con él es tan pequeña que este podría ser retirado sin que se hagan muchos cambios en el código, solo que habría que sustituir las funcionalidades utilizadas dentro de este, ya sea mediante la implementación de estas u otro marco de trabajo similar.

Entre las alternativas tenidas en cuenta se pueden mencionar Avalon, Picocontainer, Nanocontainer y HiveMind. A pesar de la variedad hay elementos en cada uno de estos que mostraron inferioridad respecto a Spring.

- Avalon: Es uno de los marcos de trabajo más antiguos. A pesar de su antigüedad este presenta un diseño menos flexible y elegante que spring. Otro de los problemas de este es que la comunidad es muy pequeña, lo cual repercute en la documentación existente del mismo.
- Picocontainer y Nanocontainer: Las funcionalidades que estos proporcionan son menos que las que spring brinda. La utilidad de estos es mejor vista en aplicaciones J2ME.
- HiveMind: Es uno de los preferidos por la comunidad, después de spring. Spring lo supera en tres áreas fundamentales:
  - Conjunto de librerías.
  - Apoyo de la comunidad.
  - Elegancia de uso.

- Acegi

Acegi es un gestor de seguridad que está diseñado para ser usado especialmente por spring. Este proporciona una capa que envuelve diversos estándares de seguridad presentes en java y ofrece

además una forma unificada de configuración a través de un descriptor XML (Juan Medin Piñeiro, 2006).

Este cubre las dos capas superiores. A nivel de presentación captura todas las peticiones mediante la implementación de un filtro y a nivel de métodos mediante interceptación a través de AOP. Para ambos caso permite aplicar los criterios de seguridad que trae o implementar otros criterios a través de interfaces diseñadas para tal fin.

- Hibernate

Hibernate es un motor de persistencia de código abierto. Permite mapear un modelo de clases a un modelo relacional sin imponer ninguna restricción en estos diseños. La documentación referente a este es abundante, ya sea mediante libros o a través de artículos publicados por la gran comunidad que este tiene detrás. JBoss respalda a hibérnate desde el punto de vista comercial proporcionando servicios de soporte, consultoría y formación. Este marco de trabajo está en constante desarrollo, incorporando las nuevas ideas que van surgiendo en el campo de la persistencia.

A pesar de la selección de hibérnate se debe reconocer que existen otros marcos de trabajo que merecieron ser valorados, estas son iBatis y Torque.

- iBatis: Este no es exactamente un mapeador objeto-relacional. Es un sistema para mapear objetos contra una base de datos mediante objetos de acceso a datos (DAO, por sus siglas en inglés), donde debe ser escrito todo el código SQL. Carece de los mapeos que ofrece hibérnate, así como la potencia de los mismos.
- Torque: Al igual que iBatis mapea mediante objetos de acceso a datos. La potencia de este no se compara con la de hibernate a nivel de mapeos ni de utilidades como la carga perezosa. Fuera de los proyectos que este desarrolla no es muy usado (Juan Medin Piñeiro, 2006).

- JCaptcha.

Un Captcha (un acrónimo de **"**completely automated public Turing test to tell computers and humans apart") es un tipo de prueba desafío-respuesta utilizado en informática para determinar si el usuario es humano. El término fue acuñado en 2000 por Luis von Ahn, Manuel Blum y Nicholas J. Hopper de la Carnegie Mellon University, y John Langford de IBM. Un tipo común de Captcha requiere que el usuario escriba las letras de una imagen distorsionada y/o oculta la secuencia de letras o números que

aparece en la pantalla. Debido a que el examen es administrado por un ordenador, en contraste con el estándar de prueba de Turing que es administrado por un humano, un Captcha es a veces descrito como una prueba de Turing inversa.

Por definición, las letras cifradas tienen las siguientes características:

- Son completamente automatizado. Esto evita la necesidad de mantenimiento o intervención humana en la prueba, con beneficios evidentes en el coste y la fiabilidad.
- El algoritmo utilizado es público. De esta forma la ruptura de un captcha pasa a ser un problema de inteligencia artificial y no la ruptura de un algoritmo secreto. (OnPedia, 2009)

Los Captchas son utilizados para evitar que bots[8] puedan utilizar ciertos servicios. Por ejemplo, para que no puedan participar en encuestas, registrarse para usar cuentas de correo electrónico

JCaptcha es un marco de trabajo que tiene implementadas todas las funcionalidades, anteriormente especificadas, para los proyectos que se desarrollan en java. Este es open source, por lo que su uso es libre. Este se integra con Acegi, permitiendo así una mayor seguridad para los sistemas.

1.7.2. Gestor de Bases de Datos.

La necesidad de persistir información en SIGESPRO obliga la selección de alguna herramienta que facilite la gestión de los datos persistentes. Ante esta necesidad se tomó la determinación de utilizar un gestor de bases de datos. Dentro de los analizados se seleccionó PostgreSQL, la selección estuvo condicionada por varias razones, entre ellas se encuentran:

- PostgreSQL define cada tabla como una clase en el paradigma OO. Permite herencia entre tablas y sus funciones y operaciones pueden ser polimórficas. (Korry Douglas, 2005)
- Es una solución Open Source. (Korry Douglas, 2005)
- Las transacciones en PostgreSQL protege los datos de la concurrencia múltiple a través del full transaction processing. El modelo de la transacción usado por PostgreSQL es basado en el multi-version concurrency control (MVCC). Es una tecnología que PostgreSQL usa para evitar bloqueos innecesarios a los usuarios. (Korry Douglas, 2005)
- PostgreSQL implementa integridad referencial apoyando las relaciones de llave primaria y foránea así como los triggers o disparadores. (Korry Douglas, 2005)

[8] Se le llaman bots a los sistemas virtuales de software que se hacen pasar por usuarios, humanos, para acceder a determinados servicios de un sistema.

- Las funciones se pueden escribir en varios lenguajes. POstgreSQL es capaza de interpretar funciones implementadas en Tcl, Perl, Python, C y C++. Además tiene su propio lenguaje, PLP/SQL. (Korry Douglas, 2005)
- Permite un gran número de conexiones concurrentes. Por defecto permite 100 pero esta puede ser configurable, solo depende de la RAM del servidor. (PostgreSQL, 2008)
- Recorridos de Mapas de Bits. Los índices son convertidos a mapas de bits en memoria cuando es apropiado, otorgando hasta veinte veces más rendimiento en consultas complejas para tablas muy grandes.
- Particionamiento de Tablas. El optimizador de consultas es capaz de evitar recorrer secciones completas de tablas grandes, a través de una técnica conocida como Exclusión por Restricciones. Esta característica mejora tanto el rendimiento como la gestión de datos para tablas de varios gigabytes.
- Bloqueos Compartidos de Registros: El modelo de bloqueos a través de la adición de candados compartidos a nivel de registro para llaves foráneas. Estos candados compartidos mejoran el rendimiento de inserción y actualización para muchas aplicaciones Procesamiento de Transacciones En Línea (OLTP: Online Transaction Processing) de gran concurrencia.

Otras alternativas pudieran ser SQL Server, Oracle y MySQL.

- En el caso de SQL Server y Oracle a pesar de ser unos gestores muy potentes son herramientas privativas y su adquisición es muy costosa.
- MySQL es Open Source, muy rápido y robusto para aplicaciones pequeñas y medianas. El objetivo fundamental de MySQL es la velocidad por lo que le resta importancia a algunas características para el manejo de los datos, como para transacciones, rollback's y subconsultas. por lo que se vuelve menos eficiente que PostgreSQL cuando el volumen de datos es considerable. (Pecos, 2007)

1.8. Tendencias actuales para el desarrollo de software

El software libre es muy usado actualmente para el desarrollo de aplicaciones computacionales por las facilidades que este brinda a una industria de software. Se puede decir que es una alternativa idónea para esta solución debido a que la misma está acorde con las necesidades actuales del país ya que el mismo se encuentra desarrollando sus proyectos sobre esta tendencia. Además el país se encuentra en un proceso de migración hacia la misma.

1.8.1. ¿Qué es software libre?

El software libre está centrado en 4 libertades básicas que establece la Free Software Fundation (FSF). Estas libertades son:

- Libertad 0: Libertad de usar el programa. Esta se refiere a que el usuario, una vez que lo adquiere puede usarlo para lo que desee sin tener que consultar con nadie.
- Libertad 1: Libertad de estudiar el funcionamiento del software y adaptarlo a las necesidades. Esto brinda la posibilidad de adaptar los sistemas a las necesidades del usuario, así como extender sus funcionalidades.
- Libertad 2: Libertad de distribuir copias. La distribución de las copias está en manos del usuario, el cual es el propietario del software una vez que lo adquiere.
- Libertad 3: Libertad para mejorar el software y hacer públicas las mejoras. Una vez que se hagan las mejoras a algún software estos deben liberarse bajo las mismas condiciones a la comunidad.

El término software libre sugiere que la distribución de los mismo debe ser gratis, aunque en muchos casos sucede así no es obligatorio que sea así por lo que no se puede asociar el termino de software libre con software gratis ya que conservando su carácter de libre, puede ser distribuido comercialmente. Análogamente, el software gratuito incluye en algunas ocasiones el código fuente; no obstante, este tipo de software no es libre en el mismo sentido que el software libre, a menos que se garanticen los derechos de modificación y redistribución de dichas versiones modificadas del programa.

1.8.1.1. Software Libre y Open Source

Aunque en la práctica el software Open Source (código abierto) y el software libre comparten muchas de sus licencias, la FSF opina que el movimiento Open Source es filosóficamente diferente del movimiento del software libre. Los defensores del término open source ven que este evita la ambigüedad del término inglés free en free software. El término "open source" fue acuñado por Christine Peterson del think tank Foresight Institute, y se registró para actuar como marca registrada para los productos de software libre.

El movimiento del software libre hace especial énfasis en los aspectos morales o éticos del software, viendo la excelencia técnica como un producto secundario deseable de su estándar ético. El

movimiento Open Source ve la excelencia técnica como el objetivo prioritario, siendo la compartición del código fuente un medio para dicho fin. Por dicho motivo, la FSF se distancia tanto del movimiento Open Source como del término "Open Source".

Aunque el término "Open Source" elimina la ambigüedad de Libertad frente a Precio (en el caso del Inglés), introduce una nueva: entre los programas que se ajustan a la Open Source Definition, que dan a los usuarios la libertad de mejorarlos, y los programas que simplemente tiene el código fuente disponible, posiblemente con fuertes restricciones sobre el uso de dicho código fuente. Mucha gente cree que cualquier software que tenga el código fuente disponible es open source, puesto que lo pueden manipular. Sin embargo, mucho de este software no da a sus usuarios la libertad de distribuir sus modificaciones, restringe el uso comercial, o en general restringe los derechos de los usuarios.

1.9. Conclusiones

En el presente capítulo se trataron diferentes conceptos que son de vital importancia para dar solución al problema planteado. El concepto más importante tenido en cuenta es el de arquitectura de software, debido a que constituye el centro de la investigación.

Además se realizó un estudio detallado de los principales estilos y patrones arquitectónicos que permitan lograr un diseño de alto nivel más robusto y escalable posible, pero a la vez que se ajuste a las características del sistema. Finalmente la decisión se inclinó al uso de una arquitectura en capas. Con la aplicación del patrón modelo-vista-controlador y mapeo de datos se pretende lograr una mejor escalabilidad y mantenibilidad a SIGESPRO.

Además se estudiaron otros aspectos como las metodologías de desarrollo, lenguajes y otras herramientas que complementan la arquitectura de este sistema. Se opto por el uso de la metodología RUP, utilizando UML para el modelado de los artefactos propuestos. El desarrollo se realizará sobre la plataforma J2EE y las herramientas utilizadas serán open source por lo que el costo en herramientas será el mínimo.

2 DESCRIPCIÓN DE LA ARQUITECTURA PARA SIGESPRO

2.1 Introducción.

En este capítulo se propone una solución al problema planteado anteriormente, la misma está regida por el artefacto documento descripción de la arquitectura que propone la UCI a emplear en los proyectos productivos. Este artefacto de trabajo, propuesto por RUP, proporciona una visión general arquitectónica completa del sistema, mediante una serie de vistas arquitectónicas diferentes para representar diferentes aspectos del sistema.

2.2 Descripción de la arquitectura.

2.2.1 Introducción

El presente capítulo muestra una descripción de la arquitectura que se ha adoptado para implementar SIGESPRO. Esta descripción de la arquitectura se hará a través de uno de los de los artefactos propuestos por RUP que se denomina de igual manera. Esta descripción se hace a través de las 4+1 vistas propuestas por dicha metodología. Estas incluyen los elementos arquitectónicamente significativos.

2.2.1.1 Propósito

La descripción de la arquitectura de un sistema tiene como propósito fundamental proporcionar una comprensión arquitectónica del sistema a cada uno de los miembros del proyecto mediante el uso de las distintas vistas. Además proporciona organización durante el desarrollo del sistema en cuestión, así como una correcta evolución del mismo. Y por último y no menos importante, fomentar la reutilización.

2.2.1.2 Alcance

Con este capítulo se proporciona una visión general de la arquitectura del proyecto SIGESPRO. Este es extensible a todos los involucrados dentro del mismo, debido a que influye en la toma de decisiones arquitectónicas dentro del sistema.

2.2.2 Representación arquitectónica.

Teniendo en cuenta el análisis realizado en el capítulo anterior, en conjunto con las características del sistema que se pretende desarrollar, se tomaron una serie de decisiones arquitectónicas con vista a

lograr un mayor nivel de abstracción en el diseño de SIGESPRO. Dentro de estas decisiones que se tomaron están:

- Centrar la propuesta sobre los lineamientos de desarrollo sobre la plataforma Java en la UCI.
- Utilizar una arquitectura por capas por cada uno de los subsistemas que se desarrollen.

2.2.3 Objetivos y restricciones arquitectónicas.

El sistema debe tener en cuenta un conjunto de requisitos (no funcionales) que garanticen la seguridad, portabilidad, escalabilidad y la integrabilidad de sus componentes. Se hace especial énfasis en los que tienen un impacto significativo en la arquitectura.

- Requerimientos de Hardware
  - Estaciones de Trabajo
    - Periféricos.
    - Tarjeta de red.
    - 256 Mb de memoria RAM.
    - Uninterruptible Power Supply (UPS).
  - Servidores de aplicación.
    - Periféricos.
    - Microprocesador Dual Core.
    - 2 GB de memoria RAM.
    - Tarjeta de red.
    - UPS.
    - 2 GB de espacio libre en disco.
    - Tarjeta de red.
    - UPS.
  - Servidor de Bases de Datos.
    - Periféricos.
    - Microprocesador Dual Core.
    - 2 GB de memoria RAM.
    - Disco Duro con capacidad de 160 GB o más.
    - Tarjeta de red.
    - UPS.
- Requerimientos de software.
  - Estaciones de Trabajo

- Sistema Operativo: Windows 98 o superior, cualquier distribución GNU/Linux.
- Navegador Web: Internet Explorer 6.0 o superior, Mozilla Firefox 2.x o superior.

➢ Servidor de aplicaciones.

- Sistema Operativo: Windows 98 o superior, cualquier distribución GNU/Linux.
- Máquina Virtual de Java: Java Development Kit, versión 1.6 (JDK)
- Servidor Web: Apache Tomcat.

❖ Redes.

Este es un requisito indispensable para el uso de la aplicación. Esta debe tener el ancho de banda suficiente como para soportar las conexiones concurrentes requeridas.

❖ Seguridad

La información que manejará el sistema no es dominio público por lo que se debe contar con un sistema de seguridad que delimite el acceso de cada usuario a las funciones que este puedan realizar. Para esto se debe tener en cuenta la estructura de accesos que establezca el cliente. 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 solo por los protocolos y los puntos de accesos bien definidos.

- ➢ Por parte de la plataforma java el marco de trabajo Acegi es el encargado de la seguridad de los subsistemas. Esta seguridad abarca autenticación y accesos a determinados privilegios del sistema.
- ➢ Se debe almacenar las trazas de las actividades realizadas por los usuarios.
- ➢ No se debe permitir otro acceso al servidor de bases de datos que no sea a través del sistema.
- ➢ El sistema debe permitir hacer copias de seguridad.

❖ Portabilidad, escalabilidad, integrabilidad.

- ➢ El sistema debe ser multiplataforma.
- ➢ El sistema debe hacer un uso racional de los recursos de las computadoras clientes.
- ➢ Los subsistemas deben contener la estructura completa (tres capas) con el fin de fomentar la reutilización de estos en otros sistemas.
- ➢ Debe permitir adicionar y eliminar subsistemas sin que los cambios sean considerables.

- Restricciones de acuerdo a las estrategias de diseño.
  - Se aplicará el paradigma de orientación a objetos para el diseño de las capas de los subsistemas.
  - Se explotaran las funcionalidades de los marcos de trabajo propuestos en cada una de las capas del estilo propuesto.
    - Capa presentación: DOJO.
    - Capa de negocio: Spring.
    - Capa de Acceso a Datos: Hibernate.

2.2.4 Tamaño y Rendimiento

En esta sección se hace una descripción de las dimensiones que puedan afectar a la arquitectura del software, así como las restricciones de rendimiento para la puesta a punto del proyecto SIGESPRO. El análisis se centrará en PostgreSQL debido a que es la herramienta que garantizará la persistencia del sistema.

- Crecimiento de los Datos.

El gestor que se propone para la solución es PostgreSQL. En el capítulo anterior se hizo un análisis de las principales ventajas de este, pero a continuación se muestra una tabla que resume las principales características en cuanto al manejo de los datos.

Limite	Valor
Tamaño máximo de la Base de Datos	Ilimitada
Tamaño máximo de fila	1.6 TB
Tamaño máximo de campo	1 GB
Máximo de filas por tabla	Ilimitada
Máximo de columnas por tablas.	250 - 1600 depende del tipo de dato de las columnas
Máximo de llaves por tabla	Ilimitada

Potencialidades de PostgreSQL (Wiki de PostgreSQL, 2008)

Independientemente de las características de PostgreSQL se hace necesario medir el crecimiento de los datos que serán manipulados por el sistema. Para este análisis se tendrá en cuenta el crecimiento

del sistema CICCV y SINAPSIS en conjunto con el análisis de la información brindada por los casos de uso del sistema de SIGESPRO.

Teniendo en cuenta que la creación de un proyecto implica un crecimiento de los datos de 3414 bytes por cada proyecto y de 660 bytes aproximadamente por cada actividad vinculada a un proyecto, sin tener en cuenta los recursos que esta lleva (660 bytes aproximadamente por cada recurso). A partir de estos valores y teniendo en cuenta los requerimientos y la experiencia de otros proyectos similares del PGG se puede concluir que para un caso extremo de 10000 proyectos por año (32.6 MB) e igual número de actividades (12.6 MB) por plan de trabajo el crecimiento sería de 45.2 MB por cada año.

- Tiempo de respuesta

PostgreSQL contribuye en gran medida a que los tiempos de respuesta sean mejores. Esto se logra a través de un grupo de funcionalidades que este incluye, entre estas las más significativas son:

- Recorrido de mapa de bits.
- Particionamiento de tablas.
- Bloqueo compartido de registros.

- Niveles de concurrencia.

En cada país pueden existir varios usuarios operando de forma centralizada y demandando servicios del servidor Web. Según estos datos y basándose en el sistema anterior (CICCV, el cual cuenta aproximadamente con 600 usuarios en total, esto es la suma de los ministerios de dos países, por lo que se puede concluir que esta será la cifra promedio por cada país) se estima que el sistema contenga alrededor de X*600 usuarios, siendo X el número de países participantes en cada convenio, según la carga de trabajo se espera en horario pico una concurrencia de 6000 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[9]. 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.

Una de las grandes ventajas de PostgreSQL es su capacidad de manejo de gran cantidad de conexiones concurrentes. Esto lo maneja a través del Multi-version Concurrency Control (MVCC).

[9] Esos valores han sido obtenidos teniendo en cuenta la experiencia de los proyectos del PGG.

2.2.5 Vista de Casos de Uso

Esta vista se encarga de los casos de uso y los actores más importantes del sistema. Estos casos de uso más importantes, o arquitectónicamente significativos como se denominan en la metodología, son aquellos que sirven para validar la arquitectura propuesta y además describen las funcionalidades imprescindibles del sistema.

2.2.5.1 Subsistema de presentación.

Este subsistema es el encargado de gestionar todo lo referente a un proyecto bilateral, desde la propuesta hasta la aprobación para una presentación en una reunión bilateral. Dentro de las principales tareas que se realizan en este están la creación de propuestas de proyectos así como el proceso de aprobación por cada uno de los niveles establecidos. Además facilita conformar el listado de proyectos que se presentarán en una reunión bilateral determinada.

A partir de un análisis del modelo de casos de uso del sistema y la respectiva descripción de cada uno de ellos se determinó un grupo de casos de uso arquitectónicamente significativos. Estos casos de uso se muestran en la siguiente figura.

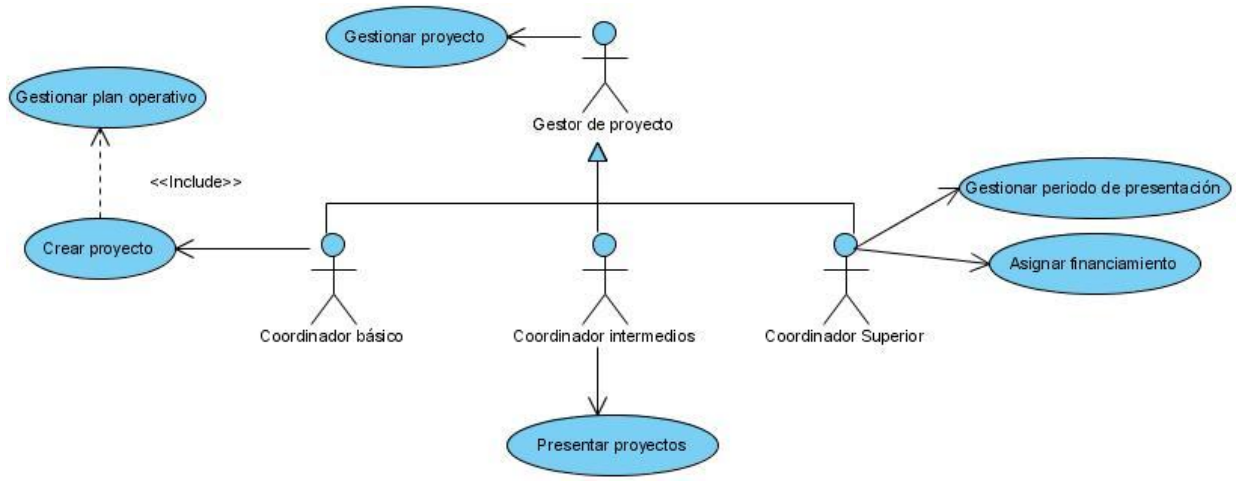


Figura 9: Diagrama de Casos de Uso arquitectónicamente significativos del subsistema presentación.

- Caso de uso crear proyecto.

Descripción: El caso consiste en crear un proyecto que será presentado a un Coordinador Básico Contraparte.

Precondiciones: El sistema debe estar instalado y ejecutándose correctamente. El actor debe estar autenticado con los permisos necesarios.

- Caso de uso Gestionar proyecto.

Descripción: El caso de uso consiste en modificar, eliminar un proyecto. También se puede aceptar o rechazar tanto una modificación como eliminación de proyecto realizada por la contraparte.
Precondiciones: El sistema debe estar instalado y ejecutado correctamente. El actor debe estar autenticado con los permisos necesarios.

- Caso de uso Gestionar plan operativo.

Descripción: El caso de uso consiste en crear o modificar un plan operativo para un proyecto que se esté definiendo.
Precondiciones: El sistema debe estar instalado y ejecutado correctamente. El actor debe estar autenticado con los permisos necesarios. Se debe haber ejecutado previamente el caso de uso Crear Proyecto.

- Caso de uso presentar proyecto

Descripción: El caso de uso consiste en seleccionar de los proyectos que ya están concebidos cuales serán presentados en el período de Presentación de Proyectos, es decir en la Reunión de Convenio. Además el Coordinador Intermedio contraparte al que propone la presentación del proyecto debe aceptar o rechazar que se presente el proyecto en el período de presentación.
Precondiciones: El sistema debe estar instalado y ejecutado correctamente. El actor debe estar autenticado con los permisos necesarios.

- Asignar financiamiento.

Descripción: El caso de uso consiste en seleccionar los proyectos que han sido aceptados por los Coordinadores Superiores e irles asignando el financiamiento especificando las fuentes de financiamiento y el monto por cada una de ellas. Además el Coordinador Superior contraparte al que propone debe aceptar o rechazar la propuesta.
Precondiciones: El sistema debe estar instalado y ejecutado correctamente. El actor debe estar autenticado con los permisos necesarios.

- Caso de uso Gestionar Periodo de Presentación

Descripción: El caso de uso consiste en crear, modificar y eliminar un Período de Presentación de Proyectos. También se puede aceptar o rechazar tanto la creación como una modificación y una eliminación realizada por la contraparte.

Precondiciones: El sistema debe estar instalado y ejecutado correctamente. El actor debe estar autenticado con los permisos necesarios.

2.2.5.2 Subsistema Seguimiento de proyectos.

El subsistema Seguimiento de proyectos tiene como principal objetivo controlar el desarrollo de los proyectos en ejecución, para ello los proyectos deben estar contratados. El control se manifiesta a partir del monto total de desembolso, el tiempo de ejecución y el presupuesto total del proyecto por concepto de recursos. Al finalizar el proceso se obtiene un proyecto terminado.

Dentro del proceso de seguimiento se enmarcan tres actividades, la ejecución física es donde se representa la parte física del proyecto por concepto de recursos y presupuesto, destinado al cumplimiento de las actividades planificadas en el tiempo establecido para cada una de ellas; la ejecución financiera comprende las modificaciones realizadas en el monto a desembolsar para ambas partes por la realización de las actividades; las transacciones financieras se ejecutan cuando se gestionan las solicitudes de pagos en los momentos pactados.

Por la complejidad de este subsistema se decidió dividir el diagrama por paquetes de casos de uso, teniendo en cuenta las funcionalidades, para lograr un mejor entendimiento del mismo, quedando de la siguiente manera.

Figura 10: Representación del diagrama de casos de uso del sistema por paquetes de casos de uso.

Para el paquete de ejecución física se determinaron los siguientes casos de uso arquitectónicamente significativos.

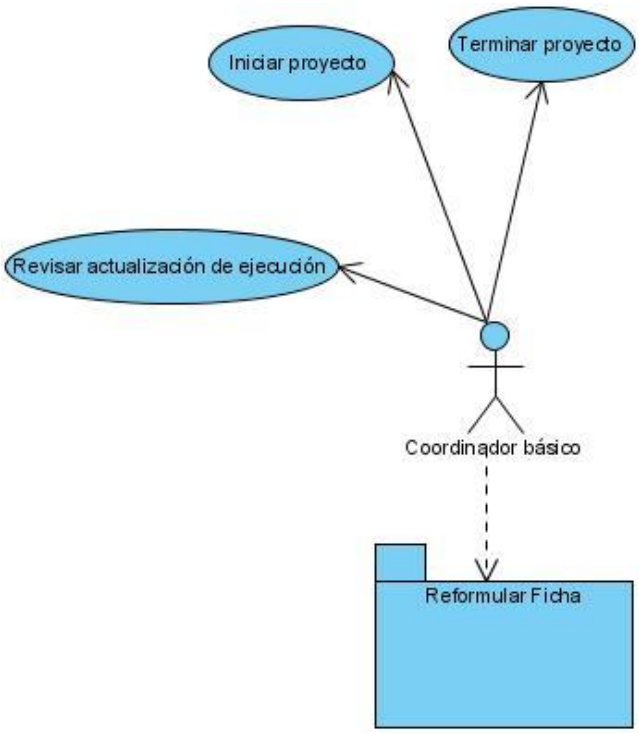


Figura 11: Casos de Uso arquitectónicamente significativos del paquete Ejecución Física.

- Caso de Uso Actualizar ejecución física.

Descripción: El caso de uso consiste en cambiar el estado de ejecución de una de las actividades del plan operativo, las mismas pueden estar en los siguientes estados: Sin ejecutar, En ejecución, Cerrada, Terminada.

Precondiciones: El sistema debe estar instalado y ejecutado correctamente. El usuario debe estar autenticado con los permisos necesarios. Tienen que existir en el plan operativo actividades que estén en los siguientes estados: Sin ejecutar y En ejecución.

- Caso de uso iniciar proyecto.

Descripción: El caso de uso consiste en proponer el inicio de uno de los proyectos con los que el ente ejecutor esté asociado.

Precondición: El sistema debe estar instalado y ejecutado correctamente. El usuario debe estar autenticado con los permisos necesarios. Tienen que existir fichas técnicas de proyectos en el estado: Ficha técnica de proyecto sin iniciar.

- Caso de uso Terminar proyecto.

Descripción: El caso consiste en proponer el fin de uno de los proyectos con los que un ente ejecutor

esté asociado.

Precondición: El sistema debe estar instalado y ejecutado correctamente. El usuario debe estar autenticado con los permisos necesarios. Todas las actividades del plan operativo tienen que estar terminadas o cerradas física y financieramente.

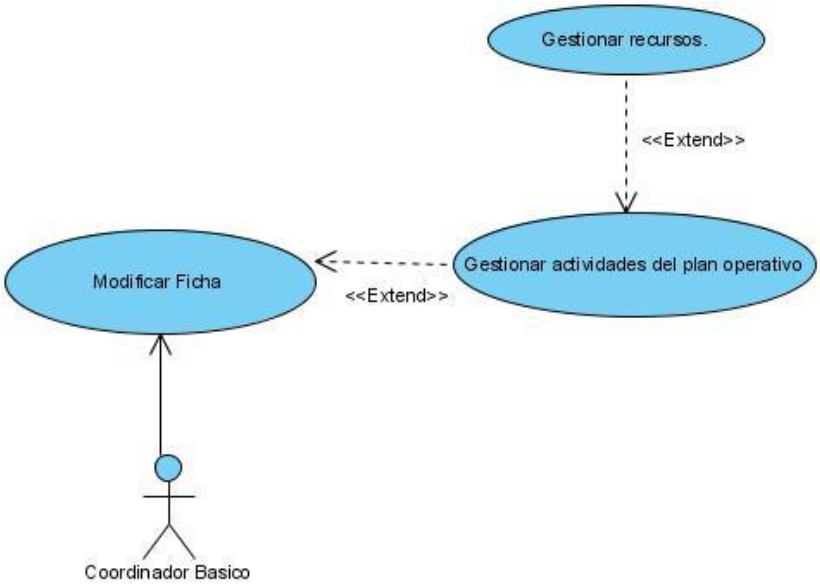


Figura 12 Casos de Uso arquitectónicamente significativos del paquete Reformular ficha.

- Caso de uso modificar ficha.

Descripción: El caso de uso consiste en reformular algunos de los datos contenidos en la ficha del proyecto, la reformulación está enmarcada a realizar modificaciones en datos como las metas, los objetivos y la modalidad del proyecto entre otras.

Precondiciones: El sistema debe estar instalado y ejecutado correctamente. El usuario debe estar autenticado con los permisos necesarios.

- Caso de uso gestionar actividades del plan operativo.

Descripción: El caso de uso consiste en re-planificar las actividades comprendidas en el plan operativo, la re-planificación está enmarcada a la creación, modificación o eliminación de las actividades.

Precondiciones: El sistema debe estar instalado y ejecutado correctamente. El usuario debe estar autenticado con los permisos necesarios.

- Caso de uso gestionar recursos.

Descripción: El caso de uso consiste en agregar modificar o eliminar los recursos asociados a una actividad específica del plan operativo.

Precondiciones: El sistema debe estar instalado y ejecutado correctamente. El usuario debe estar autenticado con los permisos necesarios.

Para el paquete de ejecución financiera se determinaron los siguientes casos de uso arquitectónicamente significativos.

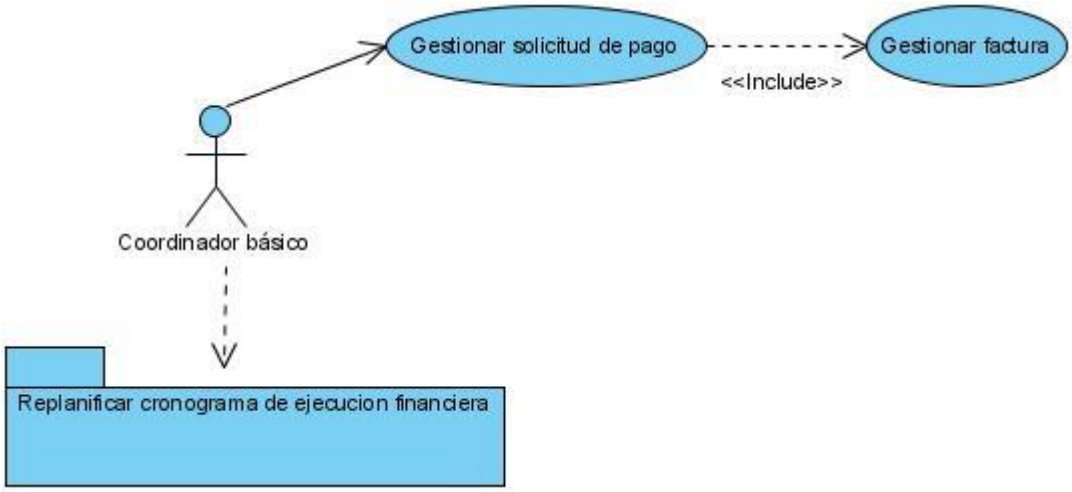


Figura 13: Casos de Uso arquitectónicamente significativos del paquete Ejecución Financiera.

- Caso de uso Gestionar solicitud de pago.

Descripción: El caso de uso consiste en crear, modificar o eliminar una solicitud de solicitud de pago correspondiente a un intervalo de fecha determinado.

Precondición: El sistema debe estar instalado y ejecutado correctamente. El usuario debe estar autenticado con los permisos necesarios.

- Caso de uso Gestionar factura.

Descripción: El caso de uso consiste en agregar, modificar o eliminar una factura a la solicitud de pago.

Precondición: El sistema debe estar instalado y ejecutado correctamente. El usuario debe estar autenticado con los permisos necesarios.

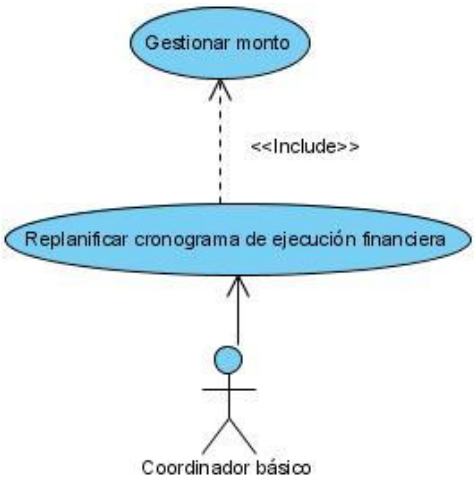


Figura 14: Casos de Uso arquitectónicamente significativos del paquete Re-planificación del cronograma de ejecución financiera.

- Caso de Uso Re-planificar plan de ejecución financiera.

Descripción: El caso de uso consiste en modificar el cronograma de ejecución financiera del proyecto.
Precondiciones: El sistema debe estar instalado y ejecutado correctamente. El usuario debe estar autenticado con los permisos necesarios. El cronograma de ejecución financiera debe estar generado.

- Caso de uso Gestionar monto.

Descripción: El caso de uso consiste en agregar, modificar o eliminar un monto a desembolsar.
Precondiciones: El sistema debe estar instalado y ejecutado correctamente. El usuario debe estar autenticado con los permisos necesarios.

Para el paquete seguimiento del contrato se determinaron los siguientes casos de uso arquitectónicamente significativos.

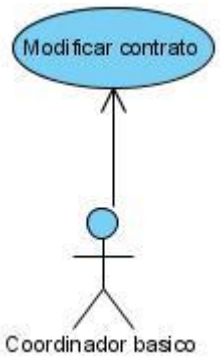


Figura 14: Casos de Uso arquitectónicamente significativos del paquete seguimiento del contrato.

- Caso de uso Modificar contrato.

Descripción: El caso de uso consiste en modificar un contrato facilitándole al usuario la información necesaria del mismo.

Precondición: El sistema debe estar instalado y ejecutado correctamente. El usuario debe estar autenticado con los permisos necesarios. Existencia de al menos un contrato.

2.2.5.3 Subsistema Contratación.

El presente subsistema es el encargado de gestionar el proceso de contratación de los proyectos que han sido aprobados y financiados. El módulo Contratación, tiene como principal objetivo elaborar el contrato de aquellos proyectos que tienen financiamiento. El control se manifiesta a partir del preforma del contrato. Al finalizar el proceso se obtiene como resultado el contrato firmado y el Plan de Ejecución Financiera para cada uno de los proyectos incluidos en el contrato.

A partir de un análisis del modelo de casos de uso del sistema y la respectiva descripción de cada uno de ellos se determinó un grupo de casos de uso arquitectónicamente significativos. Estos casos de uso se muestran en la siguiente figura.

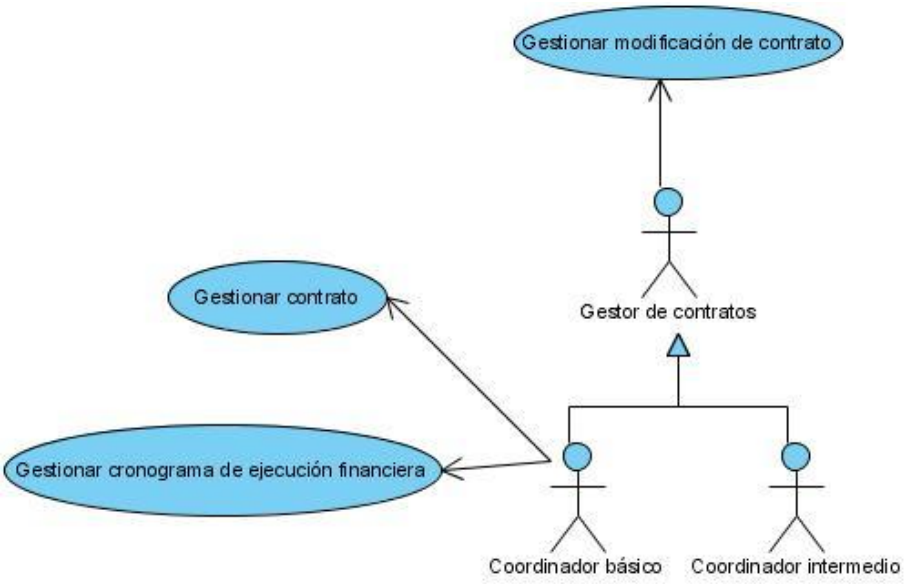


Figura 15: Diagrama de Casos de Uso arquitectónicamente significativos del subsistema contratación.

- Caso de uso Gestionar modificación de contrato.

Descripción: El caso de uso consiste en modificar un contrato por un gestor de contrato, esta modificación tiene que estar aceptada o rechazada por su contraparte.

Precondiciones: El sistema debe estar instalado y ejecutándose correctamente. El actor debe estar autenticado con los permisos necesarios.

- Caso de uso gestionar contrato.

Descripción: El caso de uso consiste en crear una propuesta de contrato, por el coordinador básico, que será presentada al ente contraparte. Esta propuesta podrá ser eliminada si es rechazada. En caso de que esté aprobado por los coordinadores básicos, la eliminación será propuesta por uno de ellos y deberá ser aceptada o rechazada por el ente contraparte.
Precondiciones: El sistema debe estar instalado y ejecutándose correctamente. El actor debe estar autenticado con los permisos necesarios.

- Gestionar cronograma de ejecución financiera.

Descripción: El caso de uso consiste en modificar el cronograma de Ejecución Financiera por un coordinador básico, el cual tiene que ser aceptado o rechazado por su contraparte.
Precondiciones: El sistema debe estar instalado y ejecutándose correctamente. El actor debe estar autenticado con los permisos necesarios.

2.2.6 Vista lógica.

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

En esta vista se debe 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.
A continuación se muestra la división por subsistemas del proyecto SIGESPRO. Esta división por subsistemas está dada por la necesidad de reutilización de cada uno de estos en otros sistemas del PGG. Además facilita el mantenimiento del sistema.

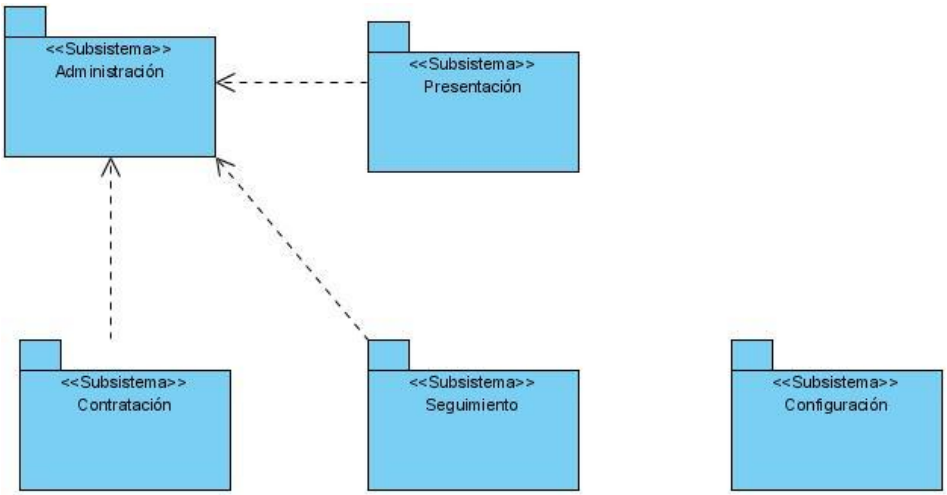


Figura 16: SIGESPRO concebido por subsistemas.

- Administración: Este subsistema es el encargado de garantizar la seguridad en cada uno de los otros subsistemas. Este chequea los privilegios de los usuarios ante una actividad que este pretende realizar y permite o deniega el acceso a la misma.
- Presentación: Este contiene la realización de los casos de uso correspondientes al proceso de presentación de un proyecto bilateral hasta que este se aprueba.
- Seguimiento: Este subsistema implementa los casos de uso relacionados con el proceso de seguimiento de los proyectos que han sido aprobados en una reunión bilateral.
- Contratación: Este subsistema comprende el proceso de contratación de los proyectos que han sido aprobados y financiados durante el proceso de presentación.
- Configuración: Este subsistema no comprende ningún proceso dentro del negocio, solo se limita a la configuración de este. Esta configuración se enmarca sobre las actividades o procesos de negocio que varían entre los posibles clientes.

Teniendo en cuenta el análisis realizado en el capítulo uno sobre los estilos arquitectónicos se utilizará una arquitectura en capas para cada una de los subsistemas que anteriormente se expusieron. Se utilizarán varios marcos de trabajo que integrados proporcionarán la robustez necesaria en cada uno de los subsistemas. Estos marcos de trabajo son Spring para la capa de negocio, Hibernate para la capa de acceso a datos y DOJO (para el diseño de las páginas web) y el módulo Spring MVC para la lógica de presentación.

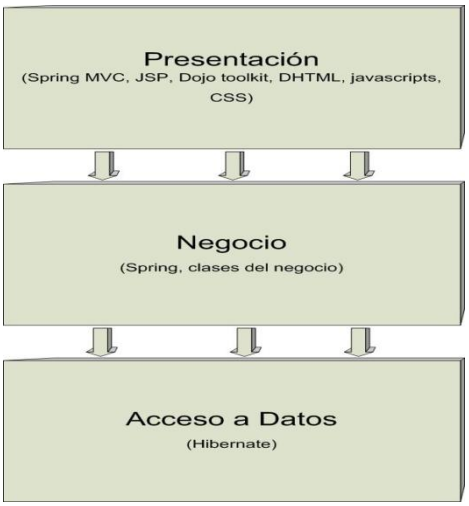


Figura 17: Representación de las capas lógicas.

- Presentación: Esta capa encapsula todas las clases y componentes de DOJO y Spring MVC. Es la encargada de la comunicación entre el usuario y las demás capas. Aquí se hace uso del patrón MVC.
  - ✓ Controladores: Estos objetos son responsables de procesar las entradas del usuario en forma de peticiones HTTP.
  - ✓ Modelo: Estos objetos contienen los datos resultantes de la ejecución de la lógica de negocio.
  - ✓ Vistas: Son responsables de mostrar el modelo resultante en la respuesta de una petición.
- Capa de negocio: Esta capa encapsula toda la lógica del negocio. Esta lógica se implementa a través de objetos de negocio. Los objetos de negocio separan los datos y la lógica de negocio usando un modelo de objetos.
- Capa de acceso a Datos: La interacción del negocio con la capa de persistencia se realizará mediante Objetos de acceso a Datos (DAO, por sus siglas en inglés). Los DAOs son ampliamente usados 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. Estos DAOs utilizan Hibernate con el fin de lograr una mayor abstracción entre la base de datos y el negocio.

2.2.6.2 Estructura de paquetes para cada subsistema.

Cada uno de los subsistemas del sistema tendrá una estructura de paquetes similar. Esta estructura se corresponderá con cada una de las capas que se proponen. Se pretende que cada subsistema pueda ser utilizado en su conjunto, por lo que contendrá dentro de si todas las capas propuestas. Para lograr esto y una mejor organización dentro del sistema se propone la siguiente estructura para los subsistemas de SIGESPRO.

- Paquete *DAO*: Este paquete contendrá todos las interfaces que soportará los objetos de acceso a datos del subsistema. Dentro de este existirá un paquete *DAO.impl* que implementará las interfaces contenidas dentro del paquete *DAO*.
- Paquete *Dominio*: Este paquete contendrá la implementación de todas las clases de domino del subsistema. Estas clases serán usadas por Hibernate para el mapeo de los datos.
- Paquete *AdministradorNegocio*: Este paquete contendrá las interfaces que soportan la lógica del negocio del subsistema. Dentro de este existirá un paquete *AdministradorNegocioImpl* que implementará las interfaces descritas en *Administrador.Negocio*.
- Paquete *ControladorWeb*: Este paquete Contiene toda la lógica de presentación.
- Paquete *PruebasU:* este paquete contendrá todas las funcionalidades correspondientes a las pruebas de unidad que se le realizan al subsistema.
- Paquete *Fachadas*: Contendrá todas las interfaces de las fachadas del subsistema. Dentro de este existirá un paquete *FachadasImpl* que contendrá todas las implementaciones de las interfaces descritas.
- Paquete *Servicios*: este paquete contendrá los componentes necesarios para la interacción con otros sistemas.

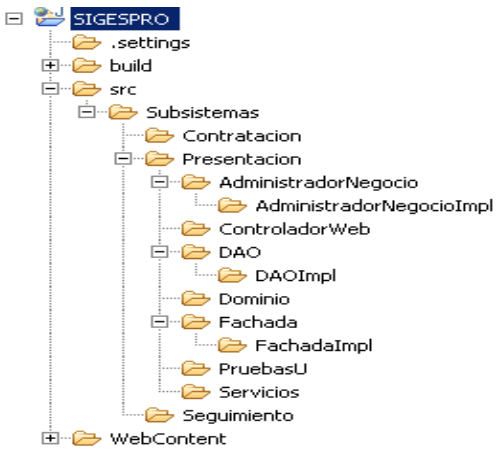


Figura 18: Estructura de los paquetes del subsistema Presentación.

2.2.6.3 Elementos arquitectónicamente significativos.

Para SIGESPRO existen un grupo de componentes que son arquitectónicamente significativos. Los servicios del negocio y los objetos de acceso a Datos (DAO) constituyen los componentes más significativos del negocio, desde el punto de vista arquitectónico. Otros elementos no menos significativos son las interfaces de los servicios de negocio y de los DAO.

- Servicios del negocio: Implementan toda la lógica del negocio del sistema.
- DAO: Facilitará la comunicación e intercambio entre los datos entre los objetos del sistema y el sistema relacional de la base de datos. Gestionará todo el proceso de persistencia de las entidades lógicas.

2.2.7 Vista de despliegue

En esta vista se muestra una distribución física de cada uno de los elementos tecnológicos que la conforman, así como la relación entre los mismos. Muestra la distribución de estos con el fin de satisfacer los requerimientos no funcionales.

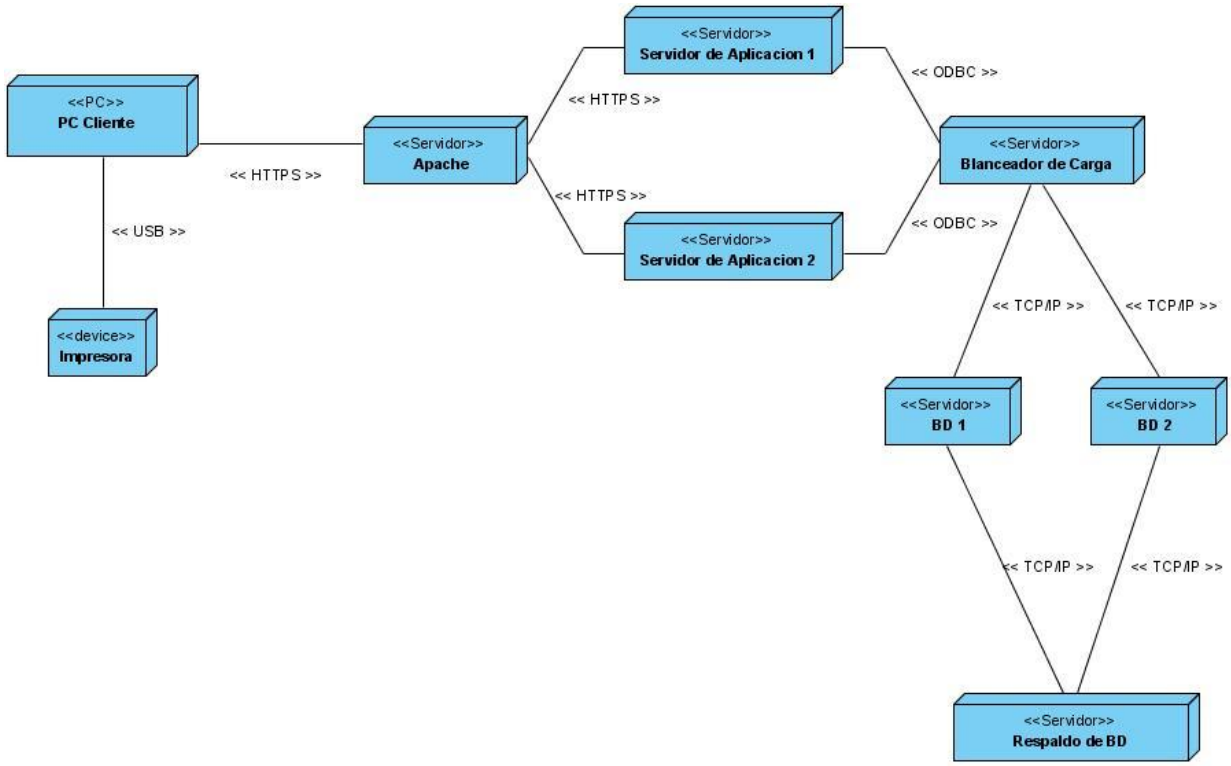


Figura 19: Diagrama de despliegue de SIGESPRO

2.2.7.1 Descripción de los elementos y las interfaces de comunicación.

- Nodos
  - Impresora: Mediante este dispositivo los usuarios podrán imprimir los reportes que los mismo entiendan deban conservarse impresos.
  - Computadora cliente: A través de este nodo se hará la interacción de los usuarios con el sistema. Existirán tantos nodos como este según entienda el cliente.
  - Servidor Apache: Este servidor es el encargado de recibir las peticiones de conexión al sistema y establecer la misma con alguno de los servidores de aplicación. Esta comunicación dependerá de la carga que tengan estos servidores de aplicación.
  - Servidor de aplicación: En este nodo estará alojado el sistema. Aquí se manejará todo lo referente a los requerimientos funcionales.
  - Balanceador de carga: Este nodo se encargará de distribuir la información del sistema que persistirá a cada uno de los servidores de BD.
  - Servidores de Bases de Datos: Este nodo contendrá la Base de Datos del sistema así como toda la información que se maneja en el mismo.
  - Servidor de aplicación 2: En este nodo estará alojado el sistema. Aquí se manejará todo lo referente a los requerimientos funcionales.

- Interfaces de comunicación
  - USB: Este puerto de conexión se utiliza para conectar dispositivos a una computadora. En este caso permitirá la comunicación entre la impresora y la computadora cliente.
  - HTTPS: Protocolo seguro de transferencia de hipertexto, utiliza el cifrado de paquetes basado en las SSL, garantizando así un canal seguro de comunicación entre la computadora cliente y los servidores apache y de aplicación. Garantiza la interoperabilidad con disimiles sistemas gestores de base de datos. Se integra perfectamente con PostgreSQL.
  - TCP/IP: La familia de protocolos de internet conjunto de protocolos TCP/IP, en referencia a los dos protocolos más importantes que la componen. El protocolo de control de transmisión (TCP) garantiza que los datos serán entregados en su destino sin errores y en el mismo orden en que se transmitieron. EL protocolo de Internet (IP) es no orientado a conexión usado tanto por el origen como por el destino para la comunicación de datos a través de una red. (Babylon, 2008)

2.2.8 Vista de Implementación.

En la vista anterior se describen los paquetes lógicos que conforman el sistema. Mediante esta vista se pretende ilustrar las funciones que se consideran de mayor importancia y las que deben ser desarrolladas por los grupos de programación del Proyecto; no es objetivo de este trabajo documentar las funcionalidades de los marcos de trabajo que se utilizarán, que además están muy bien documentadas en diferentes sitios web oficiales acerca del desarrollo de estos.

Cada uno de los subsistemas definidos anteriormente encapsulan uno o varios componentes, dependiendo de la capa lógica, que se interrelacionan entre ellos para contribuir a la solución de los elementos de la aplicación y todos estos elementos dependen de la JDK (Java Development Kit) que debe de estar instalada donde esté la aplicación para su funcionamiento. La siguiente figura ilustra lo anteriormente explicado.

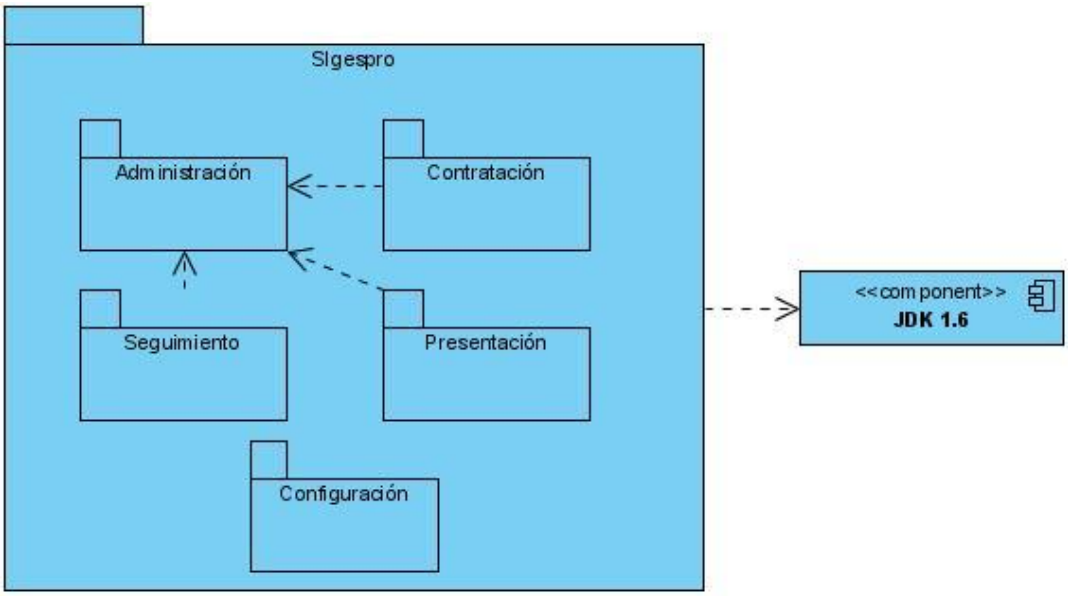
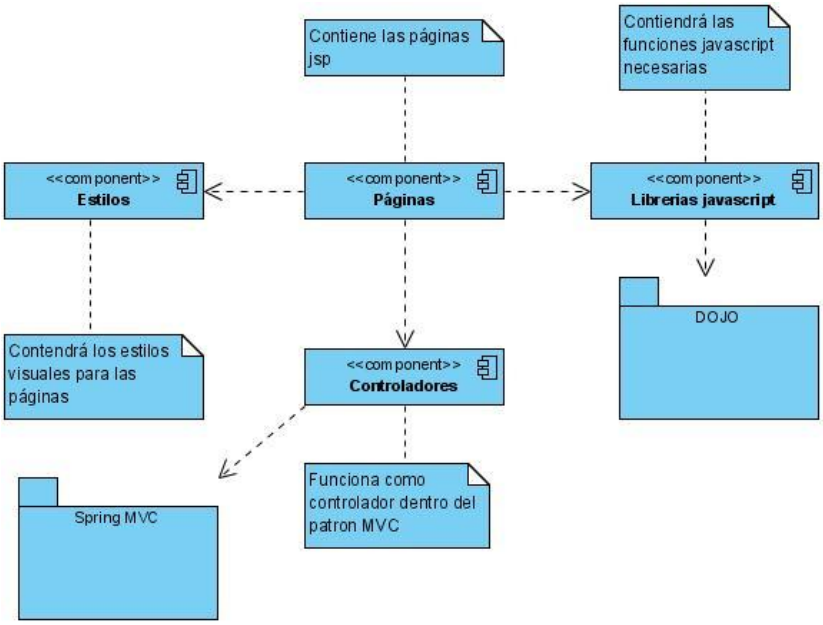


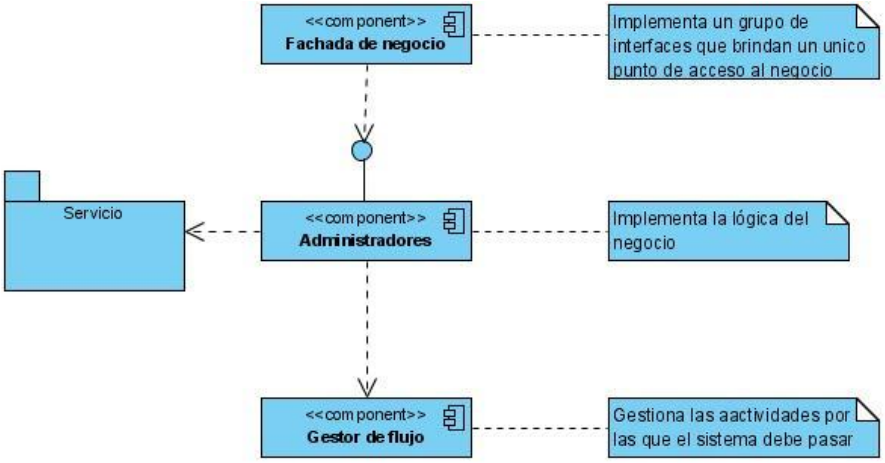
Figura 20: Componentes usados por el sistema en general.

Cada uno de los subsistemas está distribuido por capas y a su vez estas están organizadas por paquetes de aplicación. A continuación una vista de cada uno de estos paquetes.

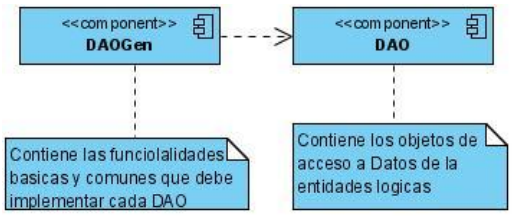
- Presentación.



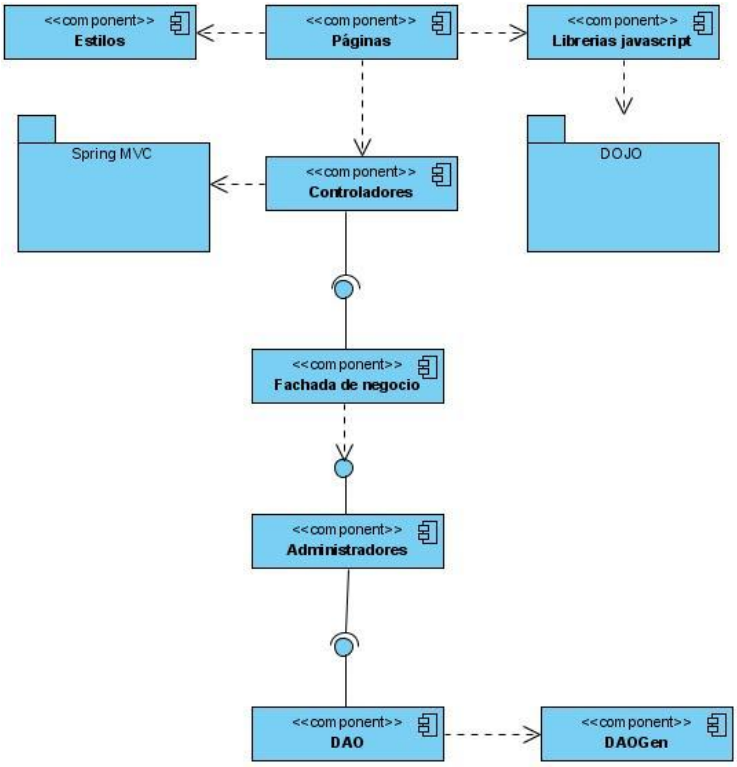
- Negocio



- Acceso a Datos



A continuación una vista general del sistema SIGESPRO.



2.2.9 Vista de Datos.

En esta sección se tratan 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.

El acceso a los datos que serán almacenados en la base de datos (BD) será mayormente consultado en la lógica del negocio a través de consultas elaboradas en HQL[10]. De esta forma se facilita una migración futura de gestor de BD, en caso que el cliente lo requiera.

Existen casos en los que los resultados requeridos en el negocio conllevan a un procesamiento complejo de los datos, que además puede que estén dispersos, por lo que el costo de procesamiento de los mismos pudiera ser considerable. Es por ello que se realizará ese procesamiento en el servidor de BD a través de funciones.

[10] Lenguaje de Hibernate para hacer consultas a una BD. Este es transparente al gestor de BD que se utilice.

Las vistas se utilizarán en casos similares a los de las funciones, pero donde no se realicen cálculos sobre los datos. Es decir, se utilizaría solamente cuando se quiere centralizar datos que están dispersos y que sería muy costoso en cuanto al procesamiento del marco de trabajo (Hibernate).
Los triggers serán utilizados en casos en que se requiera de algunas funcionalidades muy específicas como la réplica entre los servidores del clúster de BD. Sobre todo serán usados por el gestor de flujo Cualquier otra operación será realizada a través de la aplicación.
Se procurará gestionar el modelo de datos de cada uno de los subsistemas en esquemas de base de Datos diferentes con el fin de incrementar el bajo acoplamiento. La decisión de esta separación viene dada por la dinámica del sistema, esto quiere decir que cada cliente puede prescindir de alguno de los subsistemas o necesitar incluir algún otro, un único modelo de datos sería un problema en el momento de la configuración de este modelo de datos del sistema.
El modelo de datos del sistema SIGESPRO contendrá un esquema público que contendrá las tablas comunes para cada uno de los subsistemas. La relación de información de los proyectos, dígase las fichas no podrán faltar aquí.

2.3 Conclusiones.

Este capítulo describe la propuesta de arquitectura para el sistema SIGESPRO. Esta propuesta está guiada por el Documento de descripción de la arquitectura, artefacto propuesto por la metodología RUP. Se describen los principales aspectos a tener en cuenta durante el desarrollo, que garantizarán el correcto funcionamiento del sistema.
Se contemplan un conjunto de elementos que ayudan a definir la funcionalidad del sistema, como es la descripción de los casos de uso arquitectónicamente significativos. En la vista lógica se realizó una estructura de paquetes, teniendo en cuenta el estilo arquitectónico y patrones utilizados. La vista de despliegue muestra la infraestructura de tecnologías que se necesita para el correcto funcionamiento de SIGESPRO, haciendo énfasis en los principales componentes por cada una de las capas.

3 VALIDACIÓN DE LA ARQUITECTURA

3.1 Introducción.

La calidad de un software depende en gran medida de la calidad de la arquitectura que se proponga para el mismo. Para medir esta se hace necesario utilizar métodos existentes, teniendo en cuenta un conjunto de atributos de calidad asociados a esta propuesta. En este capítulo se hace una evaluación de la propuesta descrita en el capítulo 2.

3.2 Etapas en que se evalúa una arquitectura

La evaluación de la arquitectura se puede realizar en cualquier etapa del proceso de desarrollo, sobre todo cuando en el proyecto se comienzan a tomar decisiones que dependen de la arquitectura. Siempre se debe tener en cuenta que el costo de evaluar esa arquitectura debe ser menor que el costo de deshacer una decisión que dependa de dicha arquitectura.

Existen dos variantes que agrupan todos los momentos en que se puede evaluar una arquitectura, estas son evaluación temprana y evaluación tardía.

- Evaluación temprana: En esta variante no es necesario que la arquitectura no esté completamente especificada para ser evaluada. Abarca desde las fases tempranas del desarrollo del proyecto hasta el final del mismo. Este puede sufrir cambios en la arquitectura en cualquier nivel puesto que pueden existir cambios, producto de una evaluación realizada.
- Evaluación tardía: En esta se establece la evaluación cuando la arquitectura está definida y el proyecto se encuentra implementada. A este nivel es muy importante la evaluación debido a que da una medida del cumplimiento de los atributos de calidad en el sistema y da una medida de cómo será su comportamiento. (Bosh, 2000)

3.3 Atributos de calidad

No es posible evaluar una arquitectura si no se sabe que cualidades de esta se desea evaluar realmente. A estas cualidades se le llaman atributos de calidad, aunque muchas veces estos son imprecisos. Estos se clasifican en observables en vía de ejecución y no observables en vía de ejecución. Como su nombre lo indica, los primeros determinan el comportamiento en tiempo de ejecución y los segundos se establecen durante el desarrollo del sistema.

Como el sistema SIGESPRO no se ha desarrollado en este trabajo solo se hará alusión a los atributos no observables en vía de ejecución. Dentro de estos atributos se hace indispensable tener en cuenta los siguientes:

- Configurabilidad: Es la posibilidad de realizar cambios en el sistema por algún usuario experto.
- Integrabilidad: Da la medida del acoplamiento de los componentes que fueron desarrollados de manera separada.
- Integridad: Es la ausencia de alteraciones inapropiadas de la información.
- Interoperabilidad: Es un grupo especial de integrabilidad, con la diferencia que este se refiere a la habilidad de operar con otros sistemas.
- Modificabilidad: Es la habilidad de que tiene el sistema de soportar cambios futuros.
- Mantenibilidad: Es la habilidad de someter un sistemas a reparaciones y evolución de manera rápida y a bajo costo.
- Portabilidad: es la habilidad que tiene el sistema de ejecutarse en diferentes ambientes de computación. Estos pueden ser hardware, software o la combinación de ambos.
- Reusabilidad: Es la capacidad que tiene el sistema de reutilizar sus componentes en otros sistemas.
- Escalibilidad: Es la posibilidad de ampliar el diseño arquitectónico o de los datos del sistema.
- Capacidad de pruebas: Es la medida de facilidad del software para ser sometido a pruebas y este pueda demostrarlas. (Bosh, 2000)

Conociendo los atributos de calidad a medir en la arquitectura todavía no se está en condiciones de medir la calidad de una arquitectura si no se utiliza alguna técnica existente para estas evaluaciones.

3.4 Técnicas de evaluación de arquitecturas.

Las técnicas de evaluación de arquitecturas existentes permiten hacer una evaluación cuantitativa de los atributos de calidad a nivel arquitectónico. Sin embargo muchas veces esta evaluación se realiza cualitativamente, producto del costo de la cuantitativa. Para esta evaluación existen varias técnicas, como:

- Evaluación basada en escenarios[11]: Este consiste en crear escenarios donde exista un contexto determinado y una respuesta, descrita a través de la arquitectura que describe como debería responder el sistema.
- Evaluación basada en simulación: Consiste en la implementación de componentes de la arquitectura y la implementación a cierto nivel de abstracción del contexto del sistema donde se supone va a ejecutarse. Una vez implementados estos se pueden usar un conjunto de escenarios para evaluar los atributos de calidad.
- Evaluación basada en modelos matemáticos: Esta técnica es usada para validar atributos de calidad operables. Este puede ser combinado con la técnica de simulación donde la entrada de uno es el resultado del otro.
- Evaluación basada en experiencia: Esta evaluación es propiciada por arquitectos del proyecto o externos a este. Está basada en la experiencia y la intuición de estos.

De las técnicas de validación que se han mencionado se utilizará la evaluación basada en escenarios, debido a que facilita un mejor entendimiento de los atributos de calidad. Esta decisión está condicionada por el bajo costo de aplicación, el fácil entendimiento y la efectividad de esta.

3.5 Métodos de evaluación de arquitecturas.

Los métodos de evaluación de arquitecturas permiten identificar los conflictos entre las arquitecturas y las soluciones que se desarrollan. Existen varios métodos que ayudan a esta identificación, entre estos se encuentran Software Architecture Analysis Method (SAAM), Architecture Trade-off Analysis Method (ATAM) y Active Review for Intermediate Designs (ARID).

3.5.1 SAAM.

El método de análisis de arquitecturas de software (SAAM, por sus siglas en inglés) es uno de los primeros que fue ampliamente promulgado y documentado. Originalmente fue creado para el análisis de modificabilidad de la arquitectura, pero ha demostrado ser muy útil para evaluar ciertos atributos de calidad, tales como modificabilidad, portabilidad, escalabilidad e integrabilidad (R. Kazmar, 2001).
Este consiste en la enumeración de escenarios que representarán los probables cambios a los que el sistema estará sometido en el futuro. Como entrada principal necesitará la descripción de la arquitectura de dicho sistema, la cual será evaluada.

[11] Un escenario es una breve descripción de la interacción de uno de los involucrados en el desarrollo del sistema con este.

La evaluación de este método está compuesta por seis (6) pasos, los cuales son:

1. Desarrollo de escenarios.
2. Descripción de la arquitectura.
3. Clasificación y asignación de la prioridad de los escenarios.
4. Evaluación individual de los escenarios indirectos.
5. Evaluación de la interacción entre los escenarios.
6. Creación de la evaluación global.

Dependiendo del objetivo de la evaluación será el resultado de la misma. Por ejemplo, si el objetivo es evaluar una sola arquitectura, este método enumera los lugares más vulnerables a fallos dentro de la misma, en términos de los requerimientos de modificabilidad. Para el caso de que se deseen evaluar varias arquitecturas con el fin de seleccionar una que satisfaga mejor los requerimientos de calidad y con la menor cantidad de modificaciones posibles.

3.5.2 ATAM**.**

EL método de análisis de acuerdos de arquitectura (ATAM, por sus siglas en inglés) está centrado en tres aéreas distintas, el estilo arquitectónico, el análisis de los atributos de calidad y el método SAAM. Su nombre surge del hecho de que revela la forma específica en que una arquitectura cumple con algunos atributos de calidad, así como la interacción de estos con otros. (R. Kazmar, 2001)

El método se centra en identificar el estilo arquitectónico o enfoque arquitectónico utilizado. Estos representan los métodos aplicados en la arquitectura para cumplir con los atributos de calidad.(R. Kazmar, 2001)

La metodología de ATAM comprende nueve pasos divididos en cuatro fases.

Fase 1: Presentación.

1. Presentación del ATAM.
2. Presentación de las metas del negocio.
3. Presentación del negocio.

Fase 2: Investigación y análisis.

4. Identificación de los enfoques arquitectónicos.
5. Generación del Utility Tree[12].

[12] Utility tree es un esquema en forma de árbol que representa los atributos de calidad de un sistema de software, refinados hasta el establecimiento de escenarios que especifican suficiente detalle la prioridad de cada uno.

6. Análisis de los enfoques arquitectónicos.

Fase 3: Pruebas.

7. Tormenta o lluvia de ideas y establecimientos de la prioridad de los escenarios.
8. Análisis de los enfoques arquitectónicos.

Fase 4: Reportes.

9. Presentación de los resultados.

3.5.3 ARID.

El método de Análisis de diseños intermedios es conveniente para realizar revisiones parciales en etapas tempranas del desarrollo. Es conveniente saber si el 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. ADR es utilizado para la evaluación de diseños detallados de unidades de software como los componentes o módulos.

En el caso de ADR proporciona una documentación detallada del diseño y completan cuestionarios. En el caso de ATAM se realiza una evaluación de la arquitectura en su conjunto. Por esta combinación de filosofías surge ARID para la evaluación temprana de arquitecturas de software. El método de evaluación comprende nueve pasos divididos en dos fases.

Fase 1: Actividades previas.

1. Identificación de los encargos de la revisión.
2. Preparar el informe de diseño.
3. Preparar los escenarios bases.
4. Preparar los materiales.

Fase 2: Revisión.

5. Presentación del ARID.
6. Presentación del diseño.
7. Lluvia de ideas y establecimiento de prioridad de escenarios.
8. Aplicación de los escenarios.
9. Resumen.

No se puede decir que un método es mejor que otro, en cambio si se puede afirmar que en determinadas condiciones y para determinados atributos de calidad existen métodos que evalúan la arquitectura de manera más eficiente que otros.

3.6 Evaluación de la arquitectura.

Teniendo en cuenta lo anterior expuesto y que SIGESPRO se encuentra en los primeros del proceso de desarrollo de software se utilizará ARID, ya que permite realizar evaluación de la arquitectura de diseños parciales en etapas tempranas del desarrollo. Es sencillo de utilizar y el costo de evaluación es realmente bajo.

- Evaluación del escenario Gestionar usuario.
  - ✓ Atributo de calidad: Configurabilidad.
  - ✓ Perfil: Configuración.
  - ✓ Evaluación de la relación atributo–escenario: Mediante el marco de trabajo Acegi en el subsistema de Administración se gestiona la autenticación de los usuarios, teniendo en cuenta los permisos de este. Además garantiza un correcto acceso a los recursos que tiene acceso. Ante un número determinado de intentos al sistema se mostrará una captcha, la cual el usuario debe responder antes de realizar el intento.

- Evaluación del escenario Fuente de Datos.
  - ✓ Atributo de calidad: Escalabilidad.
  - ✓ Perfil: Ampliación.
  - ✓ Evaluación de la relación atributo–escenario: El gestor de bases de datos brinda la posibilidad de formar clústeres de bases de datos. Esto permite que se puedan agregar más servidores de bases de datos en caso que la concurrencia sea insuficiente o que el espacio este agotándose.

- Evaluación del escenario Acceso a Datos.
  - ✓ Atributo de calidad: Mantenibilidad.
  - ✓ Perfil: Mantenimiento.
  - ✓ Evaluación de la relación atributo–escenario: El marco de trabajo usado en la capa de acceso a datos (Hibernate) permite crear una capa de abstracción a datos que permite migrar el gestor de BD de forma transparente para la capa de negocio. Esta característica permite una arquitectura más flexible para el diseño de datos.

- Evaluación del escenario Capa de Negocio.
  - ✓ Atributo de calidad: Portabilidad.

- ✓ Perfil: Portabilidad.
  - ✓ Evaluación de la relación atributo–escenario: Los sistemas realizados en java solo necesitan de la maquina virtual de java. Esta existe para muchos sistemas operativos como Windows, la mayoría de las distribuciones GNU/Linux y Mac, por lo que el sistema debe ejecutarse sin problema alguno sobre estos sistemas operativos.

❖ Evaluación del escenario Capa de acceso a Datos.
- ✓ Atributo de calidad: Integridad.
- ✓ Perfil: Integridad.
- ✓ Evaluación de la relación atributo–escenario: El marco de trabajo Acegi garantiza la comunicación segura entre los nodos a través del protocolo HTTPS. Este cifra la información que se transmite a través de los canales de comunicación. PostgreSQL implementa mecanismo que aseguran la integridad de los datos así como mecanismos de encriptación de los usuarios y sus contraseñas.

❖ Evaluación del escenario Reutilización de un subsistema.
- ✓ Atributo de calidad: Reusabilidad.
- ✓ Perfil: Reutilización.
- ✓ Evaluación de la relación atributo–escenario: Cada uno de los subsistemas implementará sus funcionalidades de manera independiente, interactuando a través de ellos mediante fachadas. Esta estructura permite que estos puedan ser reutilizados en su conjunto.

❖ Evaluación del escenario Gestionar flujo de actividades del proceso presentación.
- ✓ Atributo de calidad: Configurabilidad.
- ✓ Perfil: Configurabilidad.
- ✓ Evaluación de la relación atributo–escenario: Cada cliente puede realizar un proceso a través de flujos distintos, para ello se utiliza el componente Gestor de procesos. Este Gestor de procesos será el encargado de controlar y guiar el proceso a través del flujo de actividades que este contenga definido. Este flujo de actividades es totalmente configurable y adaptable al cliente.

- Evaluación del escenario Gestionar flujo de actividades del proceso presentación.
  - ✓ Atributo de calidad: Modificabilidad.
  - ✓ Perfil: Adaptación.
  - ✓ Evaluación de la relación atributo–escenario: La estructura independiente de cada subsistema facilita que estos se puedan agregar o eliminar cuando se desee teniendo un costo mínimo. El modelo de los datos es también independiente por lo que no hay necesidad de transformar el modelo de datos cada vez que se adicione o elimine algún subsistema.
- Evaluación del escenario Usar información gestionada por otro sistema.
  - ✓ Atributo de calidad: interoperabilidad.
  - ✓ Perfil: Interoperabilidad.
  - ✓ Evaluación de la relación atributo–escenario: A través del paquete servicio se gestiona toda la información necesaria de otros sistemas, de los cuales el cliente requiera información. Para el caso de que se requiera compartir información se hará a través de servicios web, implementados en dicho paquete también.

3.7 Conclusiones

En este capítulo se realizó una evaluación de la arquitectura propuesta para el sistema SIGESPRO, llegando a la conclusión de que el mismo cumple con los requerimientos no funcionales que exige el sistema, satisfaciendo los atributos de calidad evaluados. La arquitectura brinda al sistema seguridad, integridad en los datos que gestiona, flexibilidad y adaptabilidad. Por lo anteriormente descrito se considera que esta arquitectura puede ser usada para SIGESPRO.

4 CONCLUSIONES GENERALES.

Teniendo en cuenta la importancia que tiene la arquitectura de software en el proceso de desarrollo de este se ha propuesto un diseño arquitectónico para SIGESPRO. Para poder cumplir con el objetivo trazado se ha hecho un estudio de los sistemas existentes actualmente que pudieran resolver las necesidades de SIGESPRO, determinando que ninguna de las analizadas cumplían a cabalidad las exigencias del PGG.

Con vista a lograr un diseño arquitectónico que cumpliese con los requerimientos de SIGESPRO se decidió utilizar una arquitectura por capas, implementando los patrones modelo-vista-controlador. Para lograr este diseño con mejor calidad se propuso el uso de los marcos de trabajo Hibernate, Spring y DOJO, todos con una probada efectividad en sistemas empresariales.

Con vista a lograr una mayor reutilización de los diferentes subsistemas se propuso que la estructura de los mismos contuvieran todas las capas. Esta estructura comprende una organización de los diferentes componentes. La vista de despliegue está confeccionada con el fin de proporcionar un mejor comportamiento del sistema ante una necesidad de mantenimiento de hardware.

Para la validación de la propuesta se utilizó el método de validación de arquitecturas ARID. Este método permitió evaluar determinados atributos de calidad, requeridos por SIGESPRO, como modificabilidad, reusabilidad e integrabilidad. En la evaluación realizada se llegó a la conclusión que el diseño arquitectónico propuesto permite el cumplimiento de los requerimientos de SIGESPRO.

5 RECOMENDACIONES

- Se recomienda desarrollar SIGESPRO a través de esta propuesta.
- Se recomienda mantener esta propuesta en constante refinamiento.
- Se recomienda una revisión más detallada de cada uno de los escenarios de la aplicación para evaluar a fondo las restricciones que se le imponen a la arquitectura.
- Se recomienda realizar un estudio de factibilidad y una evaluación de los costes de la tecnología a utilizar, por los recursos de hardware que demanda.

6 BIBLIOBRAFÍA

1. **Acosta, Fernando Perez.** Curso del sistema de ventanas X. *Curso del sistema de ventanas X*. [En línea] http://laurel.datsi.fi.upm.es/~fperez/cursoX/indice.html.
2. **Babylon. 2008.** Babylon. *Babylon.* [En línea] Babylon, 2008. http://www.babylon.com/definition/TCP_IP/Spanish.
3. **Brown, A. W. and C.Wallnau. 1998.** IEEE Software : s.n., 1998. 37-46..
4. **Fowler, Martin. 2002.** *Patterns of Enterprise Application Architecture.* s.l. : Addison Wesley, 2002. 0-321-12742-0.
5. **Garlan, David. 1994.** *An Introduction to Software Architecture.* 1994. CMU/SEI-94-TR-21.
6. **IBM. 2003.** *Ayuda de Rational Rose.* [IBM] s.l. : IBM, 2003.
7. **International Organization of Standarization. 2007.** International Organization of Standarization. *International Organization of Standarization.* [En línea] 2007. http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=45991.
8. **otros, Bachmann y. 2000.** *Technical concepts of component-based software engineering*. Carnegie Mellon University : s.n., 2000.
9. **Pressman. 2001.** *Ingenieria del software Un enfoque practico* . s.l. : Mcgraw-hill , 2001. 8448132149.
10. **Puebla, Enrique Chaviano Gómez and Yoan Arlet Carrascoso. 2008.** *Propuesta de Arquitectura Orientada a Servicios para el ERP Cubano.* Ciudad de la Habana : s.n., 2008.
11. **Sun Microsystems.** Sun Java System Portal Server 6 2004Q2 Deployment Planning Guide). *Sun Java System Portal Server 6 2004Q2 Deployment Planning Guide (chapter 1).* [En línea] Sun Microsystems. [Citado el: 20 de diciembre de 2008.] http://docs.sun.com/source/817-5321/1-portal.html .
12. **Dayley, Brad. 2006.** *Python Phrasebook.* s.l. : Sams, 2006. 0-672-32910-7.
13. **Francés, F Alvero.** *Diccionario Manual de la lengua española.* Ciudad de la Habana : Pueblo y Educación.
14. *Programación de computadoras. Introducción al paradigma OO.* **Universidad de las Ciencias Informáticas. 2007.** Ciudad Habana : s.n., 2007.

15. **Seco, José Antonio González. 2006.** www.devjoker.com. *www.devjoker.com.* [En línea] www.devjoker.com, 3 de 10 de 2006. [Citado el: 1 de 3 de 2008.] http://www.devjoker.com/asp/ver_contenidos.aspx?co_contenido=125.
16. **Souli, Juan. 2009.** cplusplus.com. *cplusplus.com.* [En línea] cplusplus.com, 9 de Abril de 2009. [Citado el: 28 de febrero de 2009.] http://www.cplusplus.com/info/description.html.
17. **ICEfaces. 2007.** ICEfaces. *ICEfaces.* [En línea] ICEfaces, 14 de noviembre de 2007. [Citado el: 15 de marzo de 2009.] http://www.icefaces.org/docs/v1_6_2/ReleaseNotes.html.
18. **Ivar Jacobson, Grady Booch, James Rumbaugh. 2000.** *El proceso Unificado del desarrollo de software.* Madrid : Addison Wesley, 2000. 84-7829-036-2.
19. **Juan Medin Piñeiro, Antonio Garcia Figueras. 2006.** *HAcia una arquitectura con JavaServer Face, Spring, Hibernate y otros frameworks.* Sevilla : s.n., 2006.
20. **Martín Santos D., Isaac A. Parra R., Saraí Gallardo V. y. 2007.** *Herramientas corporativas.* Mexico : Instituto de Investigaciones Eléctricas, 2007.
21. **Perera, Jose Raul. 2007.** *ARQUITECTURA DE SOFTWARE PARA SISTEMA GESTION DE INVENTARIOS.* Ciudad Habana : Universidad de las Ciencias Informáticas, 2007.
22. **Project Management. 2004.** *A Guide to the Project Management Body of Knowledge.* s.l. : Paperback, 2004.
23. **Roche, Jonatan Hernández.** *Un acercamiento a la Gestión de Proyectos a través de la Colaboración Internacional.* Cuba : Instituto Finlay.
24. **Sanchez, María A. Mendoza. 2004.** *Metodologías De Desarrollo De Software.* Peru : s.n., 2004.
25. **Shcmuller, Joseph. 2000.** *UML en 24 Horas.* Mexico : Pearson Educacion , 2000. 968-444-463-X.
26. **John Worsley, Joshua Drake. 2001.** *Practical PostgreSQL.* s.l. : Command Prompt, Inc., 2001.
27. **Korry Douglas, Susan Douglas. 2005.** *PostgreSQL: The comprehensive guide to building, programming, and administering.* s.l. : Sams Publishing, 2005. 0-672-32756-2.
28. **PostgreSQL. 2008.** PostgreSQL. *PostgreSQL.* [En línea] PostgreSQL, 2008. [Citado el: 17 de febrero de 2008.] http://www.postgresql.org/about/..
29. **Bosh, J. 2000.** Design & Use of software architecture . s.l. : Addison Wesley, 2000.

30. **R. Kazmar, P. Clement , M. Cleint. 2001.** *Evaluating Software architectrues. Methods and cases studies.* s.l. : Addison Wesley, 2001.
31. Wiki de PostgreSQL. (2008). *Wiki PostgreSQL*. Recuperado el 23 de 2 de 2009, de Wiki PostgreSQL: http://wiki.postgresql.org/wiki/Preguntas_Frecuentes.
32. **Babylon. 2009.** Babylon. *Babylon.* [En línea] 2009. [Citado el: 23 de marzo de 2009.] http://diccionario.babylon.com/Ente.
33. **Babylon. 2009.** Babylon. *Babylon.* [En línea] 2009. [Citado el: 23 de marzo de 2009.] http://diccionario.babylon.com/.
34. **ctisa. 2009.** Servicios informáticos. *Servicios informáticos.* [En línea] 2009. [Citado el: 25 de febrero de 2009.] http://www.ctisa.com/diccionario.asp.
35. **Sun Mycrosystems. 2009.** Sun Mycrosystems. *Sun Mycrosystems.* [En línea] 2009. [Citado el: 10 de abril de 2009.] http://java.sun.com/j2ee/overview.html.
36. **Pecos, Daniel. 2007.** PostGreSQL vs. MySQL. *PostGreSQL vs. MySQL*. [En línea] 2007. [Citado el: 11 de abril de 2009.] http://www.netpecos.org/docs/mysql_postgres/index.html.
37. **Sobrino, Alejandro. 2008.** Tuxitos. *Tuxitos.* [En línea] Tuxitos, octubre de 2008. [Citado el: 25 de febrero de 2009.] http://tuxitos.es/2008/07/28/balanceo-de-carga-entre-servidores-web-apache/.
38. **OnPedia. 2009.** OnPedia. *OnPedia.* [En línea] OnPedia, 2009. [Citado el: 20 de abril de 2009.] http://www.onpedia.com/encyclopedia/Captcha.

ANEXOS

1 Tabla de valores de crecimiento por tipo de datos soportados por PostgreSQL, según (John Worsley, 2001).

- Para datos numéricos

Tipo de Dato	Almacenado
bigint, int8	8 bytes
double precisión, float8, float	8 bytes
entero, int, int4	4 bytes
numeric, decimal	Variable
real, float4	4 bytes
smallint, int2	2 bytes
money	4 bytes
serial	4 bytes

- Para caracteres.

Tipo de Dato	Almacenado
character(*n*), char(*n*)	(4 + *n*) bytes
character varying(*n*),	Superior a (4 + *n*) bytes
text	Variable

2 Tamaño de almacenamiento de las principales entidades persistentes del sistema SIGESPRO.

Teniendo en cuenta las especificaciones de los casos de uso del sistema y la información brindada por el anexo 1 se puede llegar a la siguiente información.

- Por la creación de un proyecto se estima que tenga un crecimiento aproximado de 3414 bytes. La siguiente tabla desglosa esta información.

Información	Tipo de Dato	Tamaño (bytes)
Nombre del proyecto	varchar	54
Marco de aprobación	varchar	54
Coordinador intermedio	varchar	104
Contraparte	varchar	104
Coordinador básico	varchar	104
Contraparte	varchar	104
Duración (meses)	bigint	8
Modalidad	varchar	54
Fundamentación	text	1004
Descripción breve	text	504
Objetivos	text	504
Objetivos específicos	text	504
Metas del proyecto	varchar	104
Impacto del proyecto	varchar	104
Población beneficiada	varchar	104
Total		**3414**

- Otro de los elementos persistentes importantes en este sentido son los planes operativos, estos a su vez están compuestos por actividades y recursos humanos, los cuales aportan un crecimiento aproximado de 660 bytes cada una. A continuación esta información a detalles.

Información	Tipo de Dato	Tamaño (bytes)
Categoría	varchar	54
País	varchar	54
Cantidad	bigint	8
Tiempo	bigint	8
Honorario	double	8
Viatico	double	8
Pasaje	double	8
Otros gastos	double	8
Descripción	text	504
Total		**660**

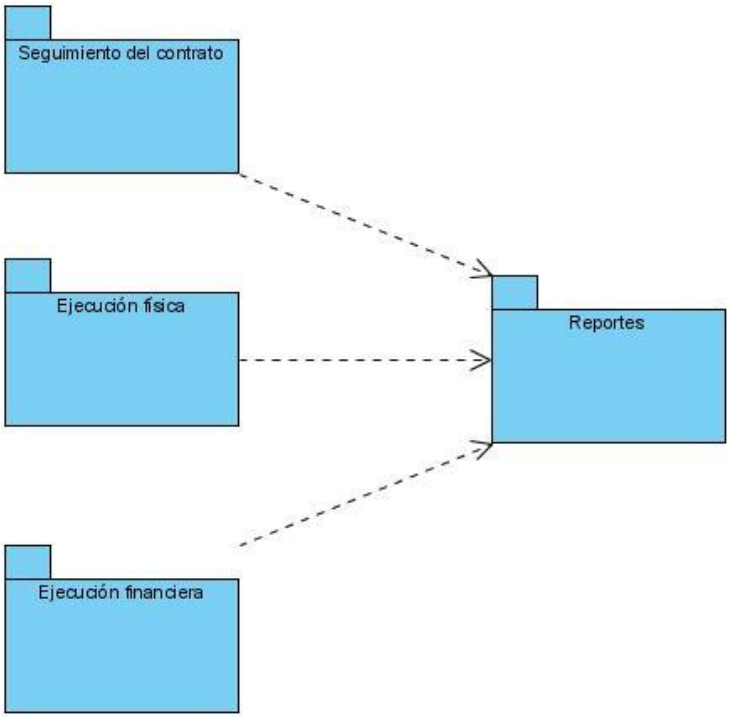
Recursos humanos.

Información	Tipo de Dato	Tamaño (bytes)
Nombre de la actividad	varchar	54
fecha de inicio	varchar	19
fecha de fin	varchar	19
Coordinador básico	varchar	104
Receptor	varchar	54
Duración	double	8
Clasificación	varchar	54
Total		**660**

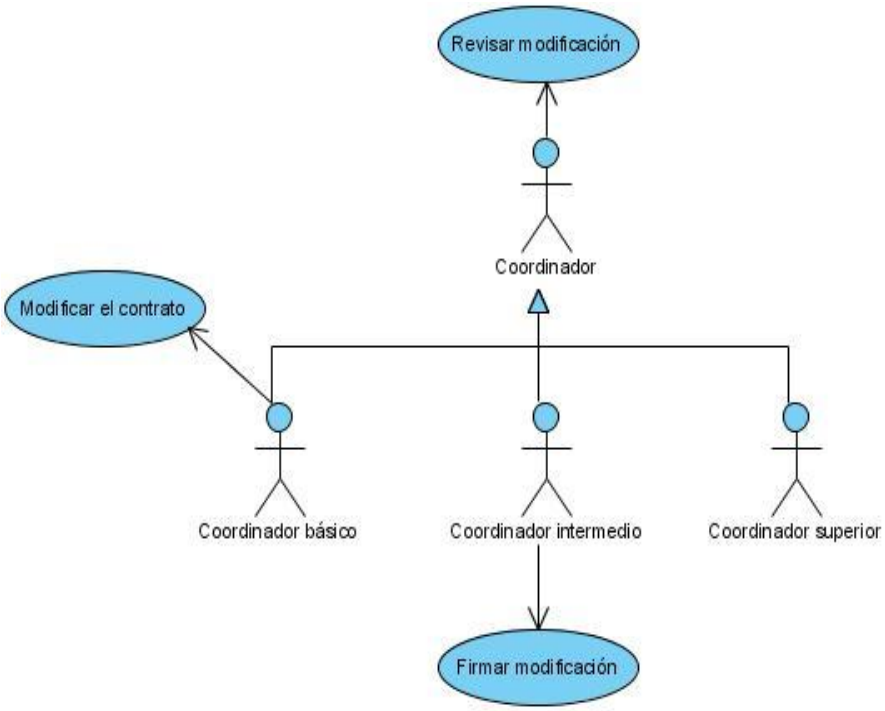
Actividad

3 Diagrama de casos de uso del sistema del subsistema Presentación.

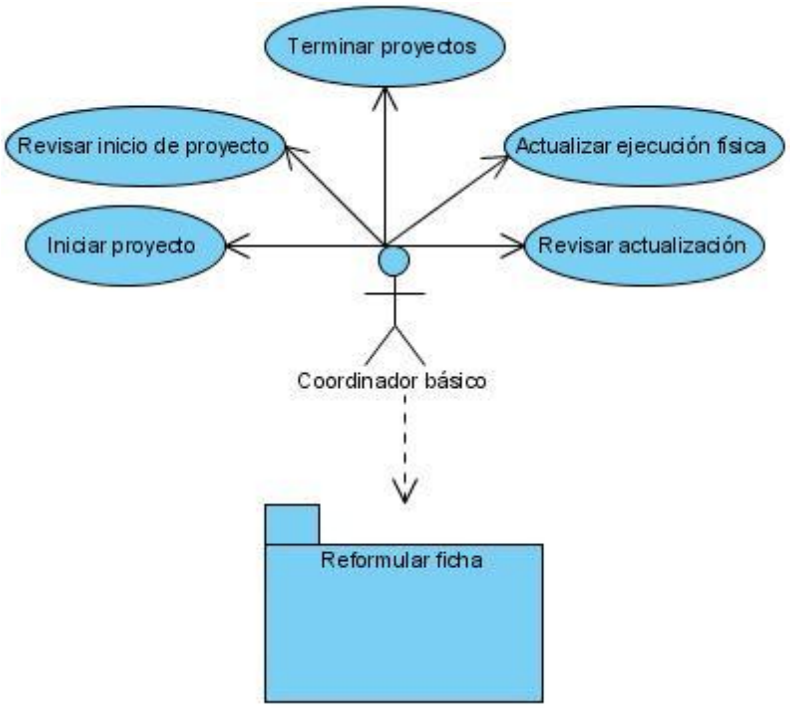
4 Diagrama de casos de uso del sistema del subsistema Seguimiento.



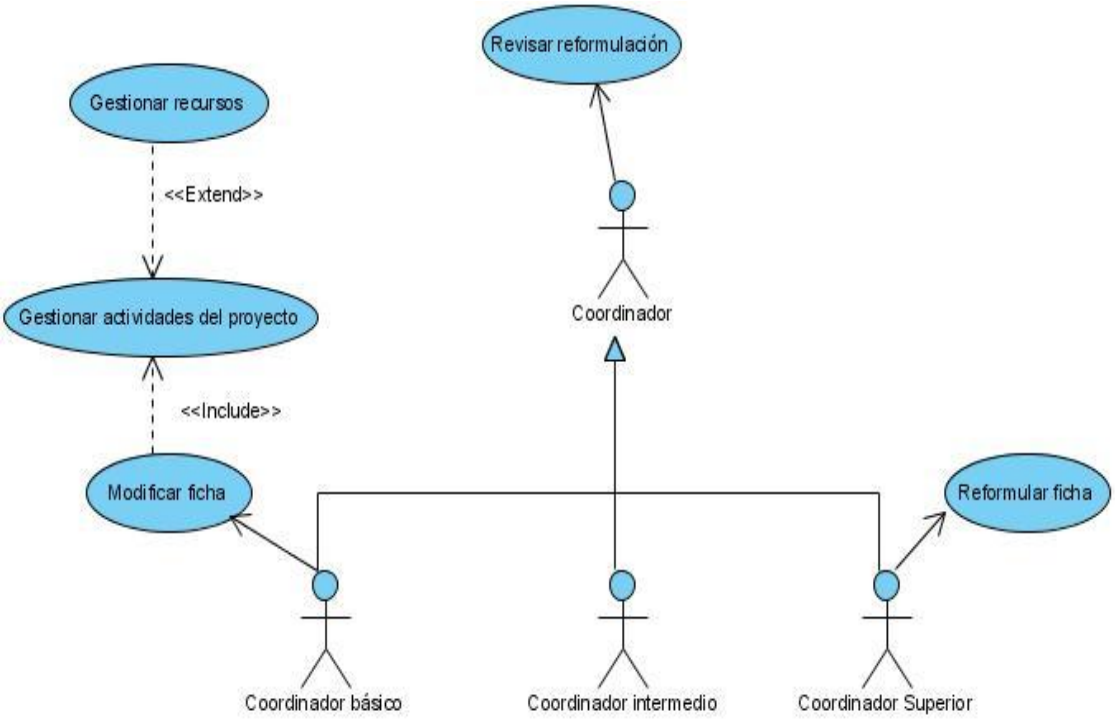
4.1 Paquete seguimiento del contrato.



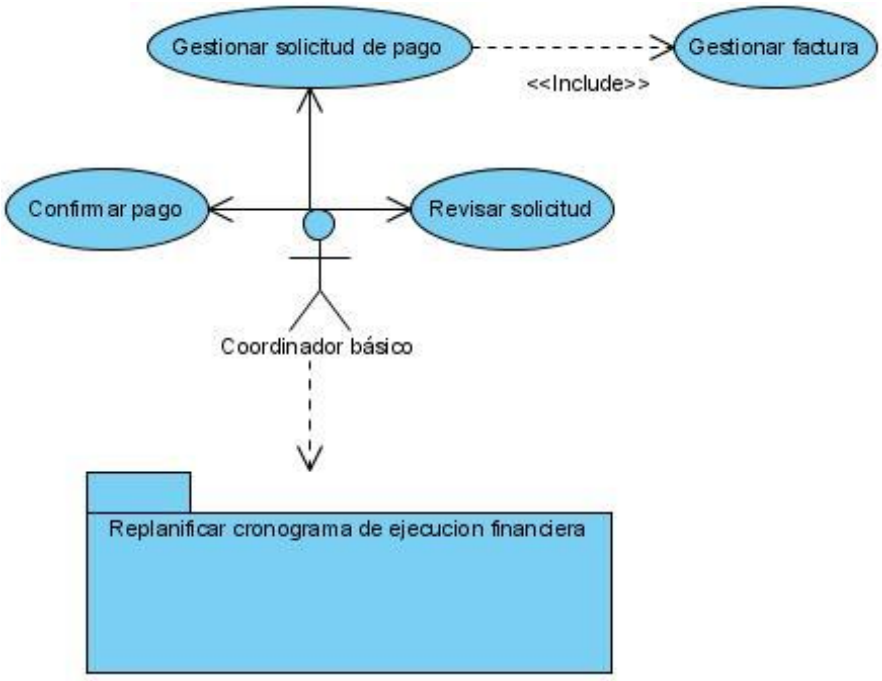
4.2 Paquete ejecución física.



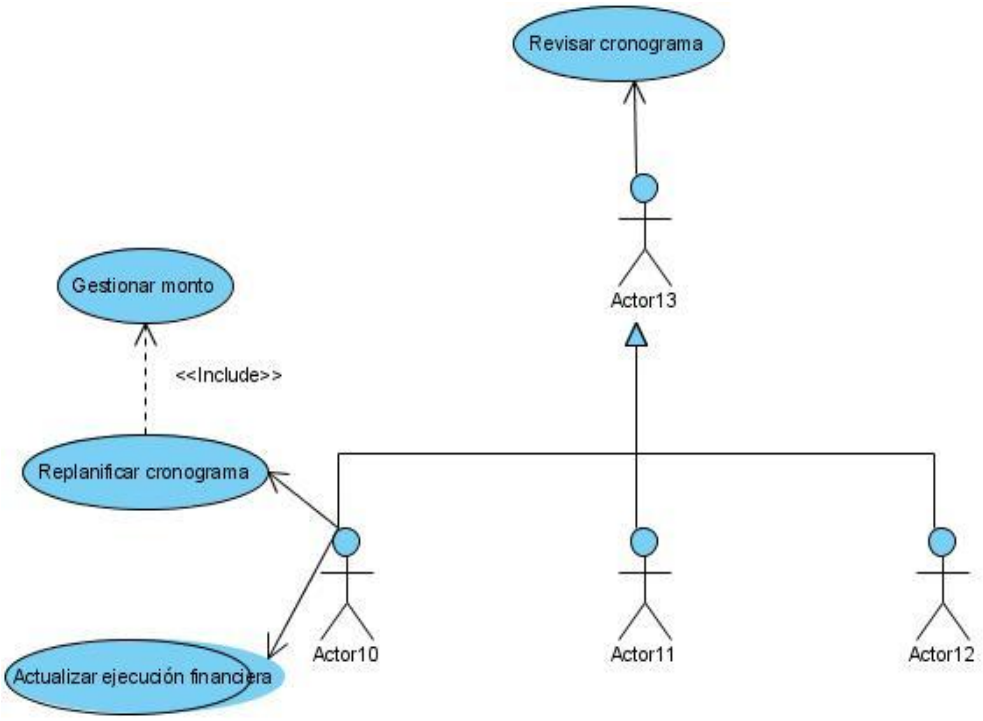
4.2.1 Paquete Reformular ficha.



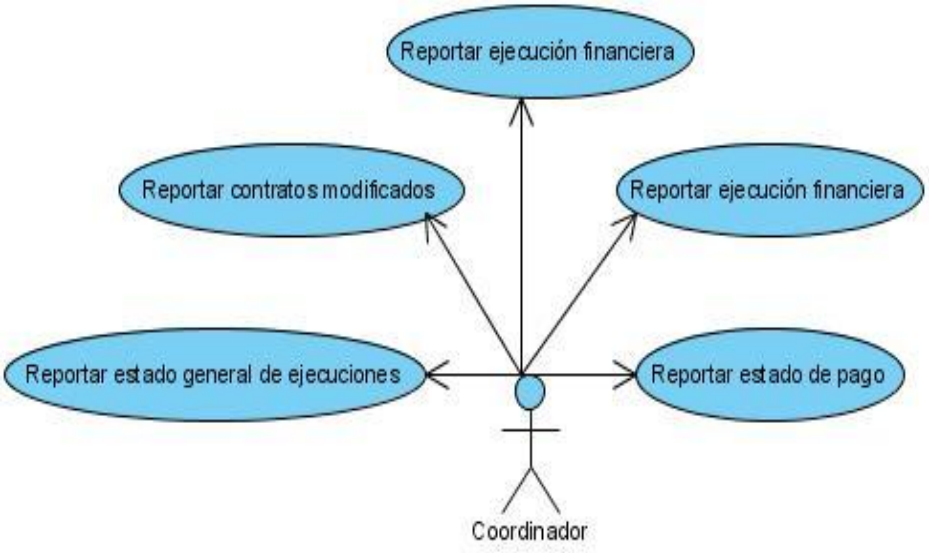
4.3 Paquete ejecución financiera.



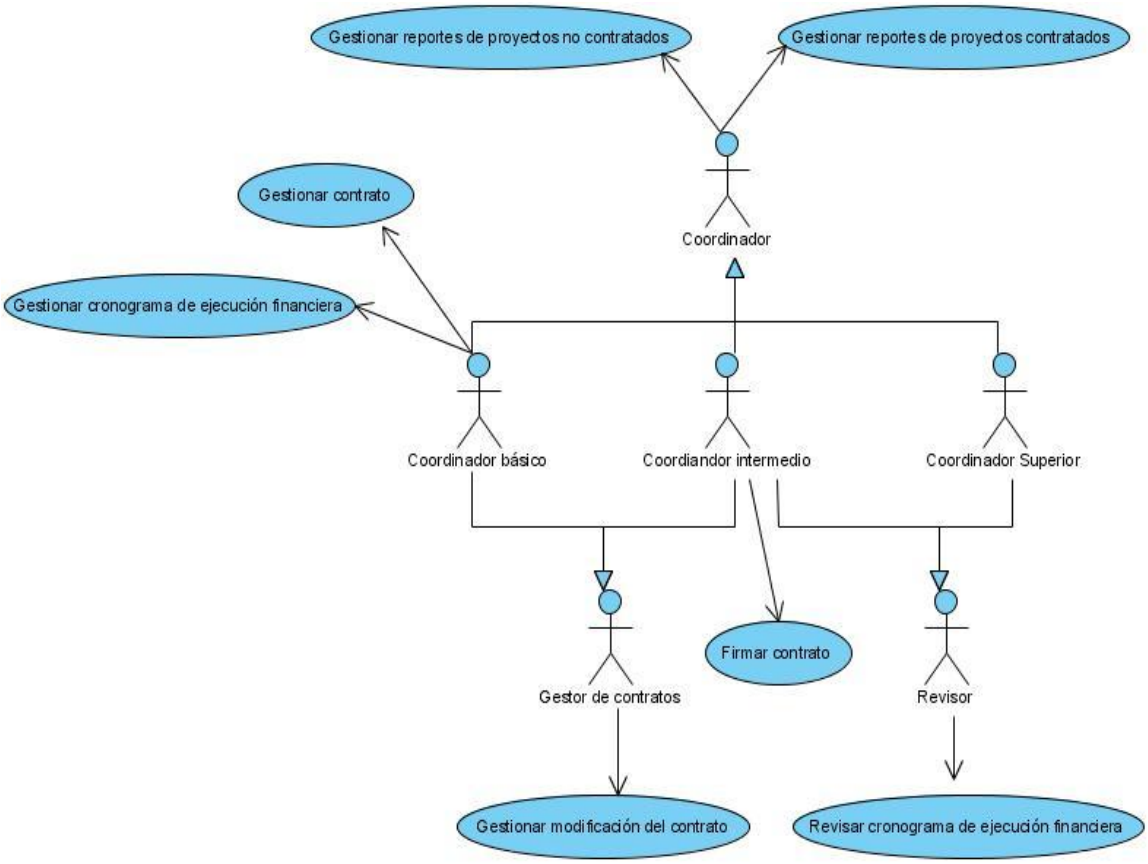
4.3.1 Paquete Re-planificar cronograma de ejecución financiera.



4.4 Paquete Reportes



5 Diagrama de casos de uso del sistema del subsistema Contratación.



GLOSARIO DE TERMINOS.

1. **Balance o balanceo de carga:** es un concepto usado en informática que se refiere a la técnica usada para compartir el trabajo a realizar entre varios procesos, ordenadores, discos u otros recursos. Está íntimamente ligado a los sistemas de multiprocesamiento, o que hacen uso de más de una unidad de procesamiento para realizar labores útiles.
2. **Ente:** Ente es todo aquello que posee ser, aunque no agote todos los rasgos del ser. Es una concreción particular del ser; en cierto modo, puede afirmarse que un ente es un ser existente de modo concreto, con unos rasgos determinados. (Babylon, 2009)
3. **Groupware:** Es el sistema que permite a los usuarios de una red local (LAN) la utilización de todos los recursos de ésta como, programas compartidos, accesos a Internet, intranet y a otras áreas, correo electrónico, firewalls, proxys, etc. (Babylon, 2009)
4. **IDE:** Se refiere a un entorno de programación que ha sido empaquetado como un programa de aplicación, es decir, consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica GUI. Los IDEs pueden ser aplicaciones por si solas o pueden ser parte de aplicaciones existentes, proveen un marco de trabajo amigable para la mayoría de los lenguajes de programación. Es posible que un mismo IDE pueda funcionar con varios lenguajes de programación. Este es el caso de Eclipse o NetBeans, que mediante pluggins se le puede añadir soporte de lenguajes adicionales.
5. **J2EE:** La Java 2 Platform, Enterprise Edition (J2EE) define el estándar para el desarrollo de aplicaciones empresariales. La plataforma J2EE simplifica las aplicaciones empresariales basándolas en normalización de componentes modulares, proporcionando un conjunto completo de servicios a esos componentes, y manejando muchos detalles de comportamiento de las aplicaciones automáticamente, sin programación compleja. (Sun Mycrosystems, 2009)
6. **Proxy:** Proxy: Software que permite a varios ordenadores acceder a Internet a través de una única conexión física. Según lo avanzado que sea, puede permitir acceder a páginas Web, FTP, correo electrónico, etc. Es frecuente que también incluyan otros servicios, como cortafuegos. (ctisa, 2009)
7. Una prueba de **desafío-respuesta** es un conjunto de preguntas (o problemas), que la persona u otra entidad que tiene que responder a fin de superar la prueba. Si la persona o entidad que proporciona una respuesta adecuada a los desafíos, entonces se considera que esta persona o entidad ha superado la prueba.