Universidad de las Ciencias Informáticas
Facultad 4

Título: Arquitectura de Software General de Auditoría

Trabajo de Diploma para optar por el título de

Ingeniero en Ciencias Informáticas

Autores: Humberto Santos Suárez.

Pedro Manuel Alás Verdecia.

Tutor: Lic. Guillermo Francisco Wood Fonseca

Co-tutor: Ing. Yaima García García.

Asesor: Ing. Iosev Pérez Rivero.

Ciudad de La Habana

Julio 2, 2008

DECLARACIÓN DE AUTORÍA

Declaramos ser autores de la presente tesis y reconocemos 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 _____.

_____ _____

Firma del Autor Firma del Autor

_____

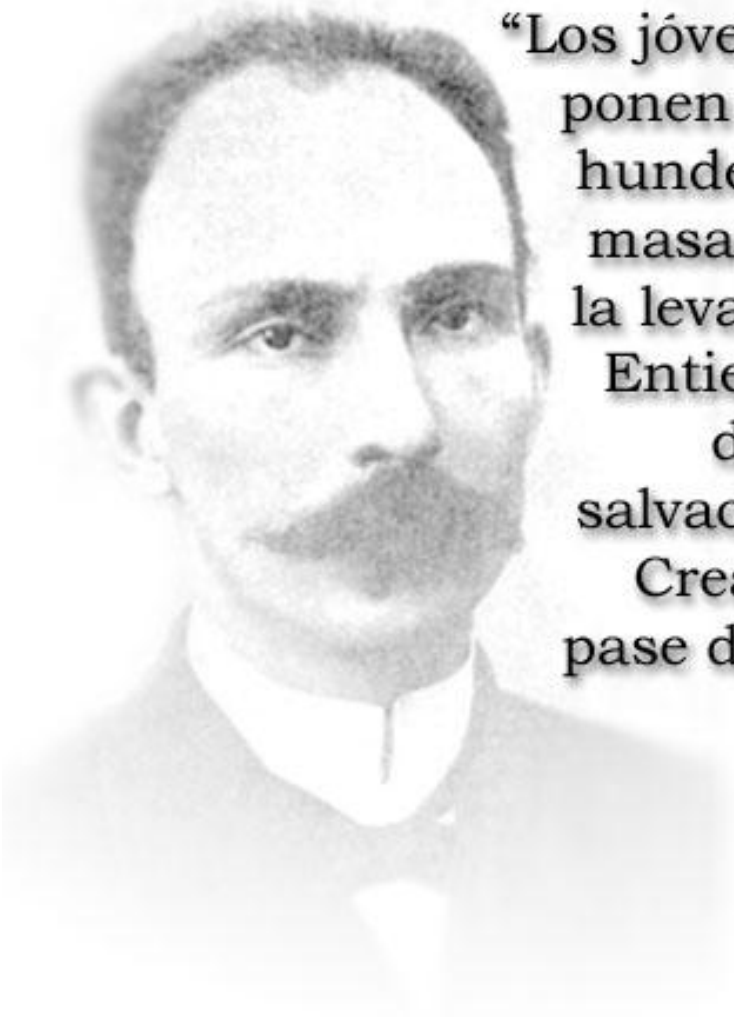
Firma del Tutor

DATOS DE CONTACTO

Guillermo Francisco Wood Fonseca, Licenciado en Ciencias de la Computación, UH, Diplomado en Economía, UH; Instructor Principal; 27 años de experiencia en el tema; 33 años de graduado en Informática.

Dirección General de Auditoría (DGA)/CIMEX
Teléfono. DGA: 203 95 97
Particular: 272 82 59

Email: gwood@cimex.com.cu
edelmir@infomed.sld.cu
guillermog@uci.cu



"Los jóvenes de América se ponen la camisa al codo, hunden las manos en la masa, y la levantan con la levadura de su sudor. Entienden que se imita demasiado, y que la salvación está en crear. Crear es la palabra de pase de esta generación".

José Martí

AGRADECIMIENTOS

A nuestros padres, por dedicar sus vidas a enseñarnos y mantenernos siempre por el camino correcto.

A Iosev Pérez, nuestro asesor, por brindar su ayuda de forma desinteresada bajo cualquier circunstancia.

A Yaima García, nuestra co-tutora, por su preocupación constante porque nuestro trabajo saliera bien.

A nuestro tutor, Guillermo Wood, por todo lo aprendido a su lado, por estar en todo momento junto a nosotros con el consejo oportuno; y sobre todo, por la confianza depositada en nuestras actitudes y aptitudes.

Al eterno Comandante Fidel, por hacer de nuestros sueños, sus sueños. Y sobre todo, por la capacidad de convertirlos en realidades.

DEDICATORIA

De Pedro Manuel

A mis padres, Enith y Domingo ejemplo y faro de mi vida.
A mi abuelita y a mi abuelote.
A mis hermanas.
A Dailin por su paciencia y amor.
A mi familia por su amor.
A todos aquellos profesores y amigos que me han enseñado y apoyado para llegar hasta aquí.

De Humberto

A mis padres, Humberto y Magalys, por su amor de siempre. Por tantas horas de desvelos, de alegrías y sufrimientos. Por haberme ayudado a llegar hasta este punto, a partir del cual, seré yo quien haga por ellos.

A mis abuelos, Leonel, Juana, Ventura y de forma muy especial a esa persona que el destino quiso arrancar de mi lado tan prematuramente, a mi abuelo Rafael; para él estas palabras: *"papi, tu niño se ha convertido en un hombre de bien".*

A mi hermana Maylin y mi prima Maibel. A mis tías, Zoraida y Mercedes, por adorarme de la manera que lo hacen.

Al viejo Juan Francisco, por siempre cuidar de mí, por ser mi guía por la vida.

A todas las personas que de una forma u otra, en el transcurso de estos 5 años en la universidad, han vivido momentos tristes y alegres a mi lado. Mencionar nombres sería correr el riesgo de dejar a alguien fuera. Por eso, es mejor darles las gracias a todos por haber contribuido en la formación de la persona que soy hoy; y asegurarles que cada uno tiene un lugar bien especial, reservado en mi corazón para siempre.

RESUMEN

A medida que las Tecnologías de la Información y las Comunicaciones (TIC) han evolucionado, han planteado nuevos desafíos a los procesos de auditoría y control de las actividades y medios de las empresas, por lo que ha sido necesario añadir a las herramientas y técnicas tradicionales de la auditoría, las Técnicas de Auditorías Asistidas por Computadoras (TAAC).

Contar con un software cubano que, haciendo uso de las TAAC pueda ser usado en los procesos de auditorías por estos profesionales cubanos, sería sin duda un paso de avance en el logro de la efectividad y eficiencia del control de los recursos de la empresa y el estado. Con el desarrollo de esta investigación se pretende proponer una arquitectura de software para desarrollar una aplicación sobre plataformas libres que dote de una poderosa herramienta a los auditores cubanos en el desempeño de sus funciones diarias.

En el trabajo se presentan una serie de vistas arquitectónicas que dan una descripción general de la arquitectura, así como distintos métodos para evaluar la misma.

PALABRAS CLAVE

Arquitectura, auditoría, TAAC

TABLA DE CONTENIDOS

INTRODUCCIÓN

A finales del siglo XX, los Sistemas Informáticos se han convertido en las herramientas más poderosas para materializar uno de los conceptos más vitales y necesarios para cualquier organización empresarial, los Sistemas de Información de la empresa.

La Informática hoy, está subsumida en la gestión integral de la empresa, y por eso las normas y estándares propiamente informáticos deben estar, por lo tanto, sometidos a los generales de la misma. A medida que las Tecnologías de la Información y las Comunicaciones (TIC) han evolucionado, han planteado nuevos desafíos a los procesos de auditoría y control de las actividades y medios de las empresas, por lo que ha sido necesario añadir a las herramientas y técnicas tradicionales de la auditoría, las Técnicas de Auditorías Asistidas por Computadoras (TAAC).

Los principales objetivos de la auditoría informática son los controles de la función informática, el análisis de la eficiencia de los sistemas informáticos, la verificación del cumplimiento de la normativa general de la empresa y el Estado en este ámbito, y la revisión de la eficaz gestión de los recursos materiales, humanos e informáticos.

El Proceso de auditoría comprende varias etapas. La primera de ellas es la planificación de la auditoría, que incluye principalmente la Exploración, el Planeamiento, la asignación de tareas a los auditores, informáticos, abogados, entre otros, que participan en el grupo que va a auditar, la asignación de recursos, tiempo y el control de todo esto y su representación gráfica para mejorar la comprensión de las autoridades de la empresa auditada.

En la Etapa de Planificación, se usan muchos software (SW en lo adelante) para auditoría: SW para elaborar el plan de auditoría, SW de auditoría para entrevistas, SW para encuestas, SW para llenar papeles de trabajo de auditoría, SW para análisis de riesgos, SW de Muestreo Estadístico que representan las ofertas de la Industria del SW a la Auditoría Externa, Interna y Gubernamental.

La Etapa de Ejecución, donde se hacen varias actividades, que utilizan varios SW de auditorías diferentes, por ejemplo: COBIT ADVISOR, PROAUDIT, RISK, entre otros. Algunos se usan desde la etapa de planificación.

Hay un momento en esta Etapa de Ejecución, del Proceso de Auditoría, en que el Auditor realiza pruebas de auditoría: sustantivas y de cumplimiento, donde él comprueba o no sus hipótesis y recoge las evidencias, las sustenta con pruebas tomadas de las Bases de Datos del auditado que después presenta a la Entidad auditada. En esta realización de pruebas, despliegan toda su fuerza SW como IDEA y el ACL.

La Etapa de Informe es donde se usan generadores de informes, graficadores y presentadores.

La auditoría puede usar SW para monitorear el comportamiento diario de los SW de banca, ERP entre otros.

Internacionalmente, es usual entre auditores usar SW General de Auditoría (SOGA), para ganar en objetividad, eficacia, eficiencia y economía en el proceso de auditoría. Además de los auditores, los contadores, informáticos, economistas, profesionales del control y la seguridad informática hacen uso de esta herramienta con los mismos objetivos en sus procesos de trabajo.

Disímil es el SW a nivel mundial que pone en práctica las TAAC y logra una excelente gestión, control e integridad de los recursos. Entre los más renombrados se encuentran ACL e IDEA, ambos pertenecientes a compañías privadas. Ambas compañías venden su SW a precios elevados que un país subdesarrollado y bloqueado como Cuba no puede permitirse comprar en cantidades necesarias. Bastaría con mencionar que una sola licencia de cada uno, tiene un valor por encima de los 1900 dólares. Si esa cifra se multiplica por alrededor de 10 000 licencias que se necesitarían en el país para que auditores, contadores, la industria del SW (analistas, calidad, auditoría informática), supervisores bancarios, entre otros, pudiesen hacer su trabajo satisfactoriamente, sin duda alguna el monto total sería alto.

En Cuba no existe un SW general de auditoría cubano, lo que limita realmente las posibilidades de estos profesionales en su trabajo diario.

Por lo antes expuesto se pretende lograr con la realización de esta investigación la definición de una arquitectura para un SW general de auditoría cubano (SOGAC). Resultará menos costoso desarrollar un software nacional que cubra las necesidades de la auditoría actual. Tomando como referencia lo antes expuesto se tiene el siguiente:

Problema a resolver: Inexistencia de una arquitectura definida para implementar en Cuba, un SW general de auditoría, que soporte los procesos de pruebas de auditoría.

Para enfocar la investigación se planteó como:

Objetivo General: Obtener una Arquitectura de SW general de auditoría, que permita tomar decisiones sobre los aspectos dinámicos y estáticos más significativos del sistema, la selección de elementos estructurales mediante los cuales se compone dicho sistema y sus colaboraciones; además que permita a los desarrolladores saber con claridad qué forma tiene el sistema que se está desarrollando.

Desglosado el mismo en los siguientes:

Objetivos específicos:

-Seleccionar y argumentar una metodología específica para el desarrollo.

-Seleccionar y argumentar un Patrón arquitectónico a aplicar

-Seleccionar, argumentar y aplicar patrones de diseño.

Objeto de Estudio: Proceso de definición de una arquitectura para un software que realice pruebas de auditorías sustantivas y de cumplimiento, a sistemas contables y financieros.

Campo de Acción: Arquitectura para SW General de Auditoría que permita realizar pruebas de cumplimiento y sustantivas a datos de tipo contable financieros.

Hipótesis: Si se construye un SW general de auditoría cubano se ahorrará al país miles de dólares que actualmente se gastan por concepto de compra de licencias de software extranjeros del tipo antes mencionados.

Tareas de la investigación

-Estudiar las técnicas y herramientas de auditoría, fundamentalmente las Técnicas de Auditoría Asistida por Computadoras (TAAC).

-Investigar y hacer un estudio sobre el funcionamiento de los SOGA más usados en el mundo.

-Determinar las funcionalidades más importantes desde el punto de vista arquitectónico.

-Investigar y Analizar el estado del arte de varias herramientas y plataformas de software.

-Definir herramientas, plataforma y tecnologías a utilizar para el desarrollo del software; que se adapten a las necesidades del país y a las características del SOGA.

-Realizar un estudio del estado del arte de varios estilos y patrones de arquitectura de software.

-Hacer un estudio de las arquitecturas existentes y evaluar hasta que punto son aplicables al sistema.

-Definir un modelo de diseño de software para el SOGAC.

-Analizar varias estrategias de prueba para la arquitectura de software.

Variable independiente:

Arquitectura

Variable dependiente:

Reusabilidad

Flexibilidad

Mantenibilidad

Métodos teóricos:

Análisis y Síntesis: para el procesamiento de la información y arribar a las conclusiones de la investigación, así como precisar las características del modelo arquitectónico propuesto.

Histórico – Lógico: Para determinar las tendencias actuales de desarrollo de los modelos y enfoques arquitectónicos.

Métodos empíricos:

Observación: Para la percepción selectiva de las restricciones y propiedades del sistema y sistémica para el control de la evolución de la arquitectura inicial.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

1.1 Introducción

1.1.1 Técnicas de Auditorías Asistidas por Computadoras.

Los objetivos de una auditoría no cambian cuando se conduce una auditoría en un ambiente de sistemas de información. Sin embargo, la aplicación de procedimientos de auditoría puede requerir que el auditor considere técnicas conocidas como técnicas de auditoría asistidas por computadora (TAAC) que usan la computadora como una herramienta de auditoría.

Las TAAC pueden mejorar la efectividad y eficiencia de los procedimientos de auditoría. Pueden también proporcionar pruebas de control efectivas y procedimientos sustantivos cuando no haya documentos de entrada o un rastro visible de auditoría, o cuando la población y tamaños de muestra sean muy grandes.

Las TAAC pueden usarse para desarrollar diversos procedimientos de auditoría, incluyendo los siguientes:

- Pruebas de detalles de transacciones y saldos, por ejemplo, el uso de software de auditoría para recalcular los intereses o la extracción de facturas por encima de un cierto valor de los registros de computadora.
- Procedimientos analíticos, por ejemplo, identificar inconsistencias o fluctuaciones importantes;
- Pruebas de controles generales, por ejemplo, pruebas de la instalación o configuración del sistema operativo o procedimientos de acceso a las bibliotecas de programas o el uso de software de comparación de códigos para verificar que la versión del programa en uso es la versión aprobada por la administración.
- Muestreo de programas para extraer datos para pruebas de auditoría.
- Pruebas de controles de aplicación, por ejemplo, pruebas del funcionamiento de un control programado;
- Rehacer cálculos realizados por los sistemas de contabilidad de la entidad.

Las TAAC son programas de computadora que el auditor usa como parte de los procedimientos de auditoría para procesar datos importantes, contenidos en los sistemas de información de una entidad. Los datos pueden ser de transacciones, sobre los que el auditor desea realizar pruebas de controles o procedimientos sustantivos, o pueden ser otros tipos de datos. Por ejemplo, los detalles de la aplicación de algunos controles generales pueden mantenerse en forma de archivos de texto u otros archivos por aplicaciones, que no sean parte del sistema contable. El auditor puede usar TAAC para revisar dichos archivos para obtener evidencia de la existencia y operación de dichos controles. Las

TAAC pueden consistir en programas de paquete, programas escritos para un propósito, programas de utilería o programas de administración del sistema. Independientemente del origen de los programas, el auditor ratifica que sean apropiados y válidos para fines de auditoría antes de usarlos.

1.1.2 Líderes mundiales en Software General de auditoría.

En la actualidad existen varios software de auditoría pero dos de ellos son líderes en el mercado mundial: ACL e IDEA.

- IDEA es una herramienta de PC basada en la Interrogación de Archivos para ser utilizada por:
- Auditores.
- Contadores.
- Investigadores.
- Personal de seguridad informática.

Analiza los datos de diversas maneras y permite la extracción, el muestreo y la manipulación de datos para identificar errores, problemas, cuestiones específicas y tendencias.

IDEA es muy eficiente en auditorías financieras, en la misma puede controlar:

- Cuentas por pagar
- Cuentas por cobrar
- Activos fijos
- Ventas
- Cobros

También se emplea para realizar pruebas específicas como pruebas de:

- Fraude
- Seguridad informática
- Dirección contable.

Las principales funcionalidades del IDEA son:

- Importar datos desde un amplio rango de tipos de archivo
- Crear diferentes vistas de los datos y reportes
- Llevar a cabo análisis específicos de los datos como:
  - Cálculo de estadísticas diversas
  - Detección de omisiones
  - Detección de duplicados
  - Sumarizaciones
  - Anticuación de fechas

• Efectuar diversidad de cálculos
• Obtener muestras usando diversas técnicas de muestreo
• Unir y comparar diferentes archivos de datos
• Crear tablas pivot para análisis multidimensional
• Generar automáticamente un historial completo documentando los análisis realizados.

IDEA cuenta con funciones para:

• Aritmética
• Textos
• Fechas
• Horas.

Estas funciones le permiten ejecutar operaciones con:

• Fechas,
• Cálculos financieros y estadísticos
• Búsquedas de textos.

Utilizando IDEA se pueden obtener varias ventajas, algunas de estas pueden ser:

• Incremento del alcance de las investigaciones llevando a cabo pruebas que no pueden ser efectuadas manualmente,
• Incremento de la cobertura (verificación de un gran número de elementos cubriendo potencialmente el 100% de las transacciones de un año o más),
• Mejor información (por ejemplo análisis adicionales o perfil de los datos),
• Permite ahorro significativo de tiempo.

ACL es un software para análisis de datos utilizados por auditores internos y externos en complejas organizaciones. Este software ayuda a monitorizar las transacciones de ERP en sus procesos. El software de auditoría financiera de ACL brinda prestaciones de análisis de datos sólidas que permiten que las organizaciones aseguren la precisión, la totalidad y la integridad de los datos transaccionales, brindando un único punto de vista de los datos de la empresa mediante una potente combinación de acceso y análisis de datos, con funciones integradas de creación de informes.

El software de ACL llega prácticamente a los datos de cualquier fuente, en cualquier sistema, mediante una interfaz de usuario consistente, ya sea que se encuentren en servidores, sistemas heredados o redes de computadoras.

Utilizando ACL es posible analizar datos más rápido y eficientemente, independientemente del departamento de informática, con una interfaz de usuario intuitiva, menús desplegables, barras de

herramientas y comandos para analizar rápidamente datos transaccionales en archivos de todos los tamaños, asegurando una cobertura del 100% y la absoluta confianza en los resultados.

También es posible:

- Producir informes claros.
- Diseñar y generar una vista previa y modificar los resultados en una forma fácil, en pantalla e identificar tendencias.
- Indicar excepciones con toda precisión y resaltar posibles áreas problemáticas.
- Localizar errores y posibles fraudes comparando y analizando archivos en función de los criterios del usuario final.
- Identificar problemas de control y garantizar el cumplimiento de los estándares.

Los procesos comunes que implementan los líderes IDEA y ACL en sus SW especializados en el análisis y auditoría de datos son las siguientes:

Importar de un amplio rango de formatos de almacenamiento de datos a un sistema propio de ficheros.

Exportar datos.

Analizar datos.

Estratificar

Determinar Antigüedad

Verificar y Ubicar Duplicados

Detectar Faltantes

Muestrear datos.

Filtrar en forma selectiva según criterios dados.

Editar Funciones para Aritmética, Texto, Fechas.

Generar Informes y estadísticas.

Exportar datos a diferentes formatos de almacenamiento.

Las necesidades de los auditores y contadores del país en estos momentos, como principales clientes, son las de poseer una aplicación de escritorio y multiplataforma, del tipo general de auditoría, para que desarrollen pruebas sustantivas y de cumplimiento. Esta herramienta se usará en los procesos de auditorías para realizar análisis a sistemas automatizados contables (SAC).

Las necesidades de los auditores cubanos coinciden con los procesos comunes que implementan los SW líderes, sin embargo, debido al precio que tienen ambos SW se hace difícil adquirir las licencias necesarias. Resulta menos costoso desarrollar un software nacional que cubra las

necesidades de la auditoría actual. Con la investigación se pretende proveer una definición de arquitectura para modelar y desarrollar un software que cumpla con las demandas que tienen los procesos de auditoría cubanos. Se cuenta con el capital humano y se tienen el mínimo de los recursos que se requieren para desarrollar un software de tal magnitud.

1.2 Arquitectura de Software (AS).

La AS tiene sus raíces en 1968, donde Edsger Dijkstra, de la Universidad Tecnológica de Eindhoven en Holanda y Premio Turing 1972, propuso que se establezca una estructuración correcta de los sistemas de software antes de lanzarse a programar, escribiendo código de cualquier manera [Dijkstra, 1983]. Dijkstra, quien sostenía que las ciencias de la computación eran una rama aplicada de las matemáticas y sugería seguir pasos formales para descomponer problemas mayores, fue uno de los introductores de la noción de sistemas operativos organizados en capas que se comunican sólo con las capas adyacentes.

En 1975, Frederick Phillips Brooks Jr, diseñador del sistema operativo OS/360 y Premio Turing 2000, utilizaba el concepto de arquitectura del sistema para designar "la especificación completa y detallada de la interfaz de usuario" y consideraba que el arquitecto es un agente del usuario, igual que lo es quien diseña su casa [Brooks Jr., 1975], también distinguía entre arquitectura e implementación; mientras aquella decía qué hacer, la implementación se ocupa de cómo. La AS siguió tomando auge con el paso de los años pero no es hasta 1992 que AS se define como disciplina de software en la publicación "Foundations for the study of software architecture" escrito por Dewayne Perry, Alexander Wolf donde planteaban [Perry & Wolf, 1992]:

"La década de 1990, creemos, será la década de la arquitectura de software…"

En el año 2000 la IEEE hace la definición "oficial" de AS en su documento IEEE 1471:

"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". [Garlan & Shaw, 1994]

Este documento de la IEEE 1471 ha sido adoptado por todo el mundo y por todas las organizaciones que tienen algo que ver con la AS. Esta definición, deja claro que la AS no se inscribe en ninguna metodología de desarrollo, sino que es en sí, una disciplina que se desarrolla durante todo el ciclo de desarrollo de software.

1.2.1 Importancia y necesidad de una arquitectura

Se necesita una arquitectura para:

• Comprender el sistema.
• Organizar el desarrollo
• Fomentar la reutilización
• Hacer evolucionar el sistema

1.2.2 Principales Corrientes Arquitectónicas

Rational Unified Process, 1999, plantea:

"Una arquitectura es el sistema de decisiones significativas sobre la organización de un sistema de software, la selección de los elementos estructurales y de sus interfaces por los cuales el sistema es compuesto, junto con su comportamiento según lo especificado en las colaboraciones entre estos elementos, la composición de estos elementos estructurales y del comportamiento en subsistemas progresivamente más grandes y el estilo arquitectónico que dirigen esta organización, los elementos y sus interfaces, sus colaboraciones y su composición"[Jacopbson, 1999].

Esta define la AS como una etapa más de la Ingeniería de Software, da una visión de la arquitectura como algo más que la construcción de herramientas o de tecnologías a usar en el desarrollo del sistema. Está enfocada a la orientación a objetos y a UML, según lo plantea la metodología de desarrollo RUP.

A continuación se expondrán las principales corrientes arquitectónicas:

• **Arquitectura estructural**, basada en un modelo estático de estilos, Lenguajes de Descripción de Arquitectura (ADLs en inglés) y vistas. Constituye la corriente fundacional y clásica de la disciplina. Los representantes de esta corriente son todos académicos, mayormente de la Universidad Carnegie Mellon en Pittsburgh: Mary Shaw, Paul Clements, David Garlan, Robert Allen, Gregory Abowd, John Ockerbloom. En toda la corriente, el diseño arquitectónico no sólo es el de más alto nivel de abstracción, sino que además no tiene por qué coincidir con la configuración explícita de las aplicaciones; rara vez se encontrarán referencias a lenguajes de programación o piezas de código.

• **Arquitectura** como una **Etapa de ingeniería y diseño orientada a objetos**. Esta corriente es una alternativa de la descrita anteriormente. Es el modelo de James Rumbaugh, Ivar Jacobson, Grady Booch, Craig Larman y otros, ligado estrechamente al mundo de UML y Rational. En esta postura, la arquitectura se restringe a las fases iniciales y preliminares del proceso y concierne a los niveles más elevados de abstracción. Importa más la abundancia y el detalle de diagramas y técnicas disponibles que la simplicidad de la visión de conjunto. La definición de arquitectura que se promueve en esta corriente tiene que ver con aspectos formales a la hora del desarrollo. Las definiciones revelan que la AS, en esta perspectiva, concierne a decisiones sobre organización, selección de elementos estructurales, comportamiento, composición y estilo arquitectónico susceptibles de ser descriptas a

través de las cinco vistas clásicas del modelo 4+1 de Kruchten [Kruchten, 1995] [Booch, Rumbaugh & Jacobson, 1999].

• **Arquitectura basada en patrones**, esta corriente se basa principalmente en la redefinición de los estilos como patrones POSA, el diseño consiste en identificar y articular patrones pre-existentes, que se definen en forma parecida a los estilos de arquitectura [Shaw, 1996]

• **Arquitectura procesual y metodologías**. Esta surge desde comienzos del siglo XXI, con centro en el SEI y con participación de algunos arquitectos de Carnegie Mellon de la primera generación y muchos nombres nuevos de la segunda: Rick Kazman, Len Bass, Paul Clements, Felix Bachmann, Fabio Peruzzi, Jeromy Carrière, Mario Barbacci y Charles Weinstock, Intenta establecer modelos de ciclo de vida y técnicas de diseño, análisis, selección de alternativas, validación, comparación, estimación de calidad y justificación económica específicas para la arquitectura de software [White & Lemus, 1997].

1.3 Patrones. Definiciones y Distinciones

Se entiende por **Patrón** una solución probada que se puede aplicar con éxito a un determinado tipo de problema que aparece repetidamente en algún campo. El establecimiento de estos patrones comunes es lo que posibilita el aprovechamiento de la experiencia acumulada en el diseño de aplicaciones.

Si se mantienen dentro del ámbito del diseño de alto nivel, donde como se ha visto se encuadra la AS, se estará hablando entonces de **Patrones Arquitectónicos**, es decir, de esquemas de organización de los sistemas de software que determinan cuál va a ser la estructura de los mismos mediante el establecimiento de su división en subsistemas, indicando las responsabilidades de cada uno de estos subsistemas y las reglas y criterios que rigen las relaciones entre ellos.

Por el contrario, si se centran en el ámbito del diseño de bajo nivel, o diseño detallado, se estará hablando entonces de los llamados **Patrones de Diseño**. A diferencia de los anteriores, el uso de un determinado patrón de diseño no afecta a la estructura general de una aplicación, sino sólo a una parte puntual de la misma.

Los patrones pueden dividirse o clasificarse en:

Patrones de Arquitectura: Relacionados a la interacción de objetos dentro o entre niveles arquitectónicos Problemas arquitectónicos, adaptabilidad a requerimientos cambiantes, desempeño, modularidad, acoplamiento, entre otros.

Patrones de Diseño: que fueron construidos dado los problemas con la claridad de diseño, multiplicación de clases, adaptabilidad a requerimientos cambiantes, proponiendo como solución: Comportamiento de factoría, Clase-Responsabilidad-Contrato (CRC).

Patrones de Análisis: Usualmente específicos de aplicación.

Patrones de Proceso o de Organización: muestran las estructuras de organización o los procesos de administración; pueden estar conformados por roles y áreas de trabajo.

Patrones de programación (o de Idioma): Patrones de Idioma: que reglan la nomenclatura en la cual se escriben, se diseñan y desarrollan los sistemas. Son patrones de bajo nivel de un lenguaje de programación específico. Describen como implementar aspectos de componentes o de las relaciones entre ellos utilizando las facilidades del lenguaje de programación dado.

Los elementos de un patrón son:

1. Nombre

• Define un vocabulario de diseño.

• Facilita la abstracción.

2. Problema

• Describe cuando aplicar el patrón.

• Conjunto de fuerzas: objetivas y restricciones.

• Pre-requisitos.

3. Solución

• Elementos que constituyen el diseño (plantilla).

• Forma canónica para resolver fuerzas.

4. Consecuencias

• Resultados.

• Extensiones.

• Consensos.

1.4 Estilos Arquitectónicos

La clave del trabajo arquitectónico tiene que ver con la correcta elección del estilo arquitectónico.

¿Qué es un estilo arquitectónico?

En el caso de los “estilos arquitectónicos” de SW son arquitecturas de SW comunes, marcos de referencias arquitectónicas, formas comunes o clases de sistemas.

Los estilos de arquitectura se definen como las 4C [Perry & Wolf, 1992]:

• Componentes (Elementos).

• Conectores.

• Configuraciones.

• Restricciones (Constraints en inglés).

A la hora de definir un estilo arquitectónico es necesario tener en cuenta el tipo de aplicación, ya que puede imponer restricciones que acotan la interacción de los componentes, además se tiene en cuenta el patrón de organización general. Algunos de los principales estilos arquitectónicos que se usan en la actualidad están divididos por Clases de Estilos las que engloban una serie de estilos arquitectónicos específicos:

1.4.1 Estilos de Flujo de datos:

Esta familia de estilos enfatiza la reutilización y la modificabilidad. Es apropiada para sistemas que implementan transformaciones de datos en pasos sucesivos.

• **Tuberías y Filtros** (estilo arquitectónico específico): Una tubería es una popular arquitectura que conecta componentes computacionales a través de conectores, de modo que el procesamiento de datos se ejecuta como un flujo. Los datos se transportan a través de las tuberías entre los filtros, transformando gradualmente las entradas en salidas. Este estilo se percibe como una serie de transformaciones sobre sucesivas piezas de los datos de entrada.

1.4.2 Estilos centrados en datos:

Esta familia de estilos enfatiza la integridad de los datos. Se estima apropiada para sistemas que se fundan en acceso y actualización de datos en estructuras de almacenamiento.

• **Arquitecturas de Pizarra o repositorio:** Esta arquitectura está compuesta por dos componentes principales: una estructura de datos que representa el estado actual y una colección de componentes independientes que operan sobre él. Estos sistemas se han usado en aplicaciones que requieren complejas interpretaciones de proceso de señales (reconocimiento de patrones, reconocimiento de habla).

1.4.3 Estilos de Llamada y Retorno:

Esta familia de estilos enfatiza la modificabilidad y la escalabilidad. Son los estilos más generalizados en sistemas en gran escala.

• **Arquitectura en Capas:** En este estilo arquitectónico cada capa proporciona servicios a la capa superior y se sirve de las prestaciones que le brinda la inferior, al dividir un sistema en capas, cada capa puede tratarse de forma independiente, sin tener que conocer los detalles de las demás. La división de un sistema en capas facilita el diseño modular, en la que cada capa encapsula un aspecto concreto del sistema y permite además la construcción de sistemas débilmente acoplados, lo que significa que si se minimiza las dependencias entre capas, resulta más fácil sustituir la implementación de una capa sin afectar al resto del sistema.

• **Arquitectura Orientada a Objetos:** Los componentes de este estilo son los objetos, o más bien instancias de los tipos de datos abstractos. En la caracterización clásica de David Garlan y Mary Shaw [Garlan & Shaw, 1994], los objetos representan una clase de componentes que ellos llaman managers, debido a que son responsables de preservar la integridad de su propia representación. Un rasgo importante de este aspecto es que la representación interna de un objeto no es accesible desde otros objetos.

• **Arquitectura basada en Componentes**: Los sistemas de SW basados en componentes se basan en principios definidos por una ingeniería de SW específica. Los componentes son las unidades de modelado, diseño e implementación, las interfaces están separadas de las implementaciones, y conjuntamente con sus interacciones son el centro de incumbencias en el diseño arquitectónico. Los componentes soportan algún régimen de introspección, de modo que su funcionalidad y propiedades puedan ser descubiertas y utilizadas en tiempo de ejecución.

1.4.4 Estilo de Código Móvil:

Esta familia de estilos enfatiza la portabilidad. Ejemplos de la misma son los intérpretes, los sistemas basados en reglas y los procesadores de lenguaje de comando.

• **Arquitectura de Máquinas Virtuales:** Esta arquitectura se conoce como intérpretes basados en tablas o sistemas basados en reglas. Estos sistemas se representan mediante un seudo-programa a interpretar y una máquina de interpretación. Estas variedades incluyen un extenso espectro que está comprendido desde los llamados lenguajes de alto nivel hasta los paradigmas declarativos no secuenciales de programación.

1.4.5 Estilos Peer To Peer:

Esta familia se conoce también como componentes independientes, enfatiza la modificabilidad por medio de la separación de las diversas partes que intervienen en la computación. Consiste por lo general en procesos independientes o entidades que se comunican a través de mensajes.

• **Arquitecturas Basadas en Eventos:** Estas se han llamado también arquitectura de invocación implícita, se vinculan con sistemas basados en publicación-suscripción. La idea dominante en la invocación implícita es que, en lugar de invocar un procedimiento en forma directa, un componente puede anunciar mediante difusión uno o más eventos.

• **Arquitecturas Orientadas a Servicios (SOA en inglés):** Esta construye toda la topología de la aplicación como una topología de interfaces, implementaciones y llamados a interfaces; es una

relación entre servicios y consumidores de servicios, ambos lo suficientemente amplios como para representar una función de negocio completa.

• **Arquitecturas Basadas en Recursos:** Esta define recursos identificables y métodos para acceder y manipular el estado de esos recursos. El caso de referencia es nada menos que la World Wide Web, donde los URLs identifican los recursos y HTTP es el protocolo de acceso. El argumento central es que HTTP mismo, con su conjunto mínimo de métodos y su semántica simplísima, es suficientemente general para modelar cualquier dominio de aplicación.

1.5 Patrones de Arquitectura

Al contrario de los estilos arquitectónicos los patrones son muchos y muy variados y es casi imposible revisar todos los patrones que existen a la hora de hacer una determinada aplicación, por eso se recomienda el uso de los patrones que estén predeterminados en cada uno de los estilos que se seleccionen para la arquitectura.

Los siguientes son algunos de los patrones más usados, divididos en las principales categorías de patrones arquitectónicos [Gamma, 2002]:

- **Domain Logic Patterns** (Lógica de dominio)

- Domain Model (Modelo de dominio): Modelo del objeto del dominio que incorpora comportamiento y datos.
- Service Layer (Capa de servicio): Define el límite de un uso con una capa de servicios que establezca un sistema de operaciones disponibles y coordine la respuesta del uso en cada operación.

  Este patrón se puede utilizar con el fin de englobar la funcionalidad de la lógica de negocio solo en una capa de la aplicación.

- **Data Source Architectural Patterns (Patrones Arquitectónicos de Fuente de Datos)**

- Active Record (Registro Activo):

  Un objeto que envuelve una fila en una tabla o una consulta de la base de datos, encapsula el acceso de base de datos, y agrega lógica del dominio en los datos.
- Data Mapper (Mapeo de Datos):

  La capa Mapper (Mapeo) mueve los datos, convirtiendo los objetos en datos a insertar en las tablas.

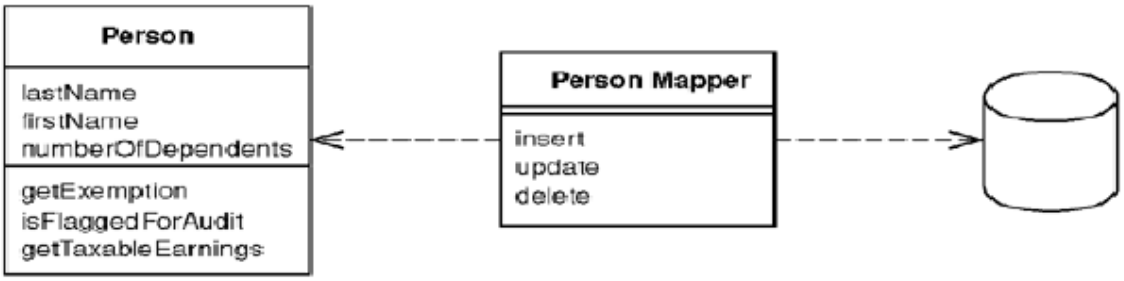


Fig. 1- Capa de Mapeo[Palos, 2006]

Presentation Patterns (Patrones de Presentación)

Model View Controller (Modelo - Vista - Controlador):

Divide la interacción con la interfaz de usuario en tres elementos diferentes

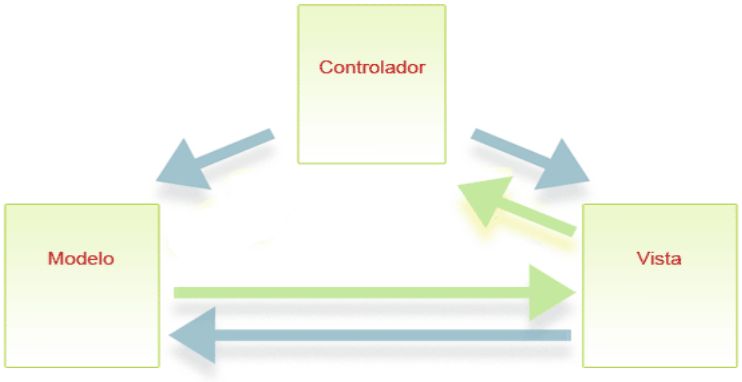


Fig. 2- Modelo Vista Controlador

Patrones J2EE [Sun Microsystems, 2005]

Data Access Object (Objetos de Acceso a Datos):

El acceso a los datos varía dependiendo de la fuente de los datos. Tener acceso al almacenaje persistente, por ejemplo a una base de datos, varía grandemente dependiendo del tipo de almacenaje (bases de datos orientadas al objeto, ficheros planos).

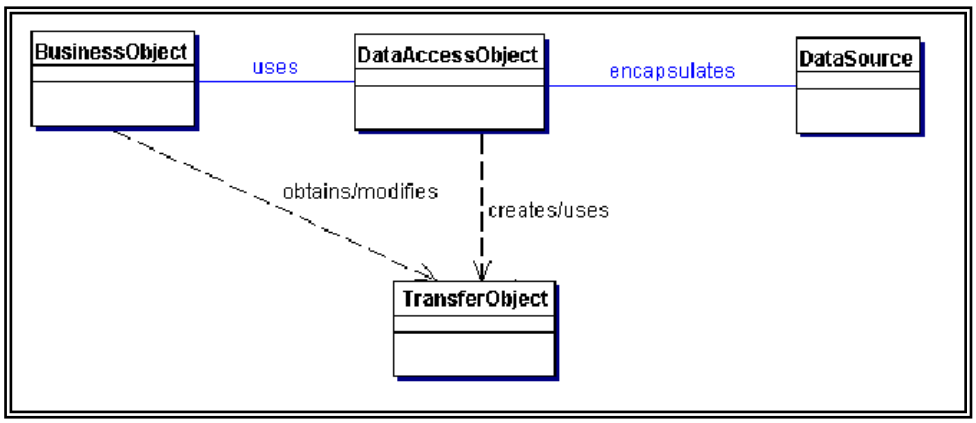


Fig. 3- Patrón Acceso a Datos [Palos, 2006]

1.6 Lenguajes de Descripción Arquitectónica (ADLs en inglés)

En la década de 1990 y en lo que va del siglo XXI, se han materializado diversas propuestas para describir y razonar en términos de arquitectura de SW: muchas de ellas han asumido la forma de lenguajes de descripción arquitectónicos o ADLs, estos ocupan una parte importante del trabajo arquitectónico desde la fundación de la disciplina. Se trata de un conjunto de propuestas de variado nivel de rigurosidad, casi todas ellas de extracción académica, que fueron surgiendo desde comienzos de la década de 1990 hasta la actualidad con el proyecto de unificación de los lenguajes de modelado bajo la forma de UML.

Los ADLs difieren sustancialmente de UML, que al menos en su versión 1.x se estima inadecuado en su capacidad para expresar conectores en particular y en su modelo semántico en general para las clases de descripción y análisis que se requieren. Los ADLs permiten modelar una arquitectura mucho antes que se lleve a cabo la programación de las aplicaciones que la componen, analizar su adecuación, determinar sus puntos críticos y eventualmente simular su comportamiento.

Entre las características a considerar de estos lenguajes se pueden citar las siguientes:

• **Composición:** Permiten la representación del sistema como composición de una serie de partes.

• **Configuración:** La descripción de la arquitectura es independiente de la de los componentes que formen parte del sistema.

• **Abstracción:** Describen los roles o papeles abstractos que juegan los componentes dentro de la arquitectura.

• **Flexibilidad:** Permiten la definición de nuevas formas de interacción entre componentes.

• **Reutilización:** Permiten la reutilización tanto de los componentes como de la propia arquitectura.

• **Heterogeneidad:** Permiten combinar descripciones heterogéneas.

• **Análisis:** Permiten diversas formas de análisis de la arquitectura y de los sistemas desarrollados a partir de ella.

Entre los principales ADLs por ser los más usados en la actualidad están:

• ACME (1995)

• Armani (1998)

• Jacal (1997)

• CHAM (1990)
• Aesop (1994)
• ArTek (1994)

1.7 Metodologías de Desarrollo de Software

1.7.1 Rational Unified Process

El proceso unificado de desarrollo (RUP), es en la actualidad uno de los más usados por las empresas de software y es validado continuamente para el perfeccionamiento de su uso. Está concebido para que desde el inicio del proceso, se establezca una definición acertada del proyecto, haciendo innecesarias las reconstrucciones parciales posteriores. Además está dirigido a la programación orientada a objetos que permite obtener sistemas escalables en el tiempo que no necesitarán grandes inversiones de recursos en sus modificaciones posteriores.

Las características principales de este proceso son:

- Guiado por casos de Uso.
- Iterativo e incremental.
- Centrado en la arquitectura.

A través de un proyecto guiado por RUP, los requerimientos funcionales son expresados en la forma de casos de uso, que guían la realización de una arquitectura ejecutable de la aplicación. Además el proceso focaliza el esfuerzo del equipo en construir los elementos críticos estructuralmente y del comportamiento (llamados Elementos Arquitecturales) antes de construir elementos menos importantes.

Este proceso se basa en el modelo espiral que organiza iteraciones por etapas y fases para obtener una estructura más sólida, clara y ajustable a las necesidades particulares de cada organización, facilitando además la administración del proyecto. Cada iteración se considera un sub proyecto que no sólo genera documentación, sino también productos de software, permitiendo con esto que el usuario tenga puntos de verificación y control más rápidos, y que se realice un proceso continuo de pruebas y de integración desde las primeras iteraciones.

Presenta la particularidad de que, en cada ciclo de iteración, se hace exigente el uso de artefactos, siendo por este motivo, una de las metodologías más importantes para alcanzar un grado de certificación en el desarrollo de software y como un enfoque basado en modelos, utiliza bien definido para tal fin el lenguaje de representación visual UML.

1.7.2 eXtreme Programming (XP)

XP es una de las metodologías de desarrollo de software más exitosas en la actualidad utilizada para proyectos a corto plazo. La metodología consiste en una programación rápida o extrema, cuya popularidad es tener como parte del equipo, al usuario final, pues es uno de los requisitos para llegar al éxito del proyecto.

La metodología se basa en:

Pruebas unitarias: se basa en las pruebas realizadas a los principales procesos, de tal manera que se puedan hacer pruebas de las fallas que 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, lo cual consiste en que dos desarrolladores participen en un proyecto en una mismo estación de trabajo. Cada miembro lleva a cabo la acción que el otro no está haciendo en ese momento.

Lo fundamental en este tipo de metodología es:

La comunicación entre los usuarios 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.

1.7.3 Comparación entre RUP y XP

- ✓ XP es orientado al cliente y de iteraciones cortas y rápidas. También hay que decir que debido al carácter general de RUP, se consideran todos los demás proceso de desarrollo como casos particulares de este.
- ✓ RUP está pensado para proyectos y equipos grandes, en cuanto a tamaño y duración. XP se implementa mejor para proyectos cortos y equipos pequeños.
- ✓ RUP es más escalable que XP, ya que a mayor tamaño de código y/o equipo, mayor es la necesidad de cierta organización.

Después de analizadas las características de estas dos metodologías de desarrollo, y por adecuarse a las necesidades y magnitudes del proyecto que se quiere desarrollar se escogió a RUP como la metodología de desarrollo por todo lo antes explicado.

1.8 El Lenguaje Unificado de Modelado

El Lenguaje Unificado de Modelado (UML) es una notación para especificar, visualizar y documentar sistemas de SW desde la perspectiva de la orientación a objetos. Su objetivo es representar el conocimiento acerca de los sistemas que se pretenden construir y las decisiones tomadas durante su desarrollo, tanto lo referido a su estructura estática como a su comportamiento dinámico. UML postula un proceso de desarrollo **iterativo**, **incremental**, **guiado por los casos de uso** y **centrado en la arquitectura** [Booch et al., 1999]. La representación en UML de un sistema de SW consta de cinco vistas o modelos parciales separados, aunque relacionados entre sí, denominadas vista de casos de uso, de diseño, de implementación, de procesos y de despliegue. Cada uno de estos modelos representa el sistema por medio de diversos diagramas.

Aunque no existe de forma explícita una vista arquitectónica, estas cinco vistas pretenden describir, en su conjunto, la arquitectura del sistema [Kruchten, 1995]. En favor de UML se puede señalar que es un lenguaje gráfico con sintaxis y semántica bastante bien definidas. La sintaxis de la notación gráfica se especifica mediante su correspondencia con los elementos del modelo semántico subyacente [Rumbaugh et al., 1999], cuya semántica se define de manera semi-formal por medio de un meta-modelo, textos descriptivos y restricciones.

En cuanto a su capacidad para describir la arquitectura, se debe señalar que UML maneja conceptos utilizados también por la Arquitectura del SW, como son los de interfaz, componente o conexión, y que los mecanismos de extensión disponibles pueden utilizarse para definir otros conceptos no contemplados o para establecer restricciones que definan de forma más precisa la semántica de estos conceptos.

1.9 Herramientas CASE:

1.9.1 Rational Rose Enterprise Suite:

Esta herramienta es una solución integrada y completa para todo el ciclo de vida del desarrollo de SW. Acelera el desarrollo mediante el modelado visual, la generación de código y las capacidades de ingeniería inversa. La suite de Rational ofrece varios productos, destacándose los siguientes:

1.9.2 Rational Rose:

Es una herramienta de modelación visual para el proceso de modelación del negocio, análisis de requerimientos y diseño de arquitectura de componentes. Proporciona mecanismos para realizar la denominada Ingeniería Inversa, es decir, a partir del código de un programa, se puede obtener información sobre su diseño.

1.9.3 Rational Requisite Pro:

Mantiene a todo el equipo de desarrollo actualizado a través del proceso de desarrollo de aplicaciones haciendo que los requerimientos se puedan escribir, comunicar y cambiar fácilmente. Facilita que los miembros del equipo colaboren con los requerimientos del proyecto.

1.9.4 Rational ClearQuest:

Es un sistema poderoso y altamente flexible para seguimiento de defectos y cambios que captura y maneja todo tipo de solicitud de cambio a lo largo del ciclo de vida del desarrollo.

1.9.5 Rational SoDA:

Automatiza la producción de documentación para todo el proceso de desarrollo de SW, reduciendo dramáticamente el tiempo y el costo de documentar el SW. Brinda a sus usuarios una interfaz única para reportar requerimientos, diseño, pruebas, y estado de defectos.

1.9.6 Rational ClearCase:

Es una herramienta de administración de configuración de SW, simplifica el proceso de cambio, ofrece una gestión fiable, ampliable y flexible de los activos de SW para equipos de desarrollo de gran tamaño y tamaño medio.

1.10 JAVA

Las características principales que nos ofrece Java respecto a cualquier otro lenguaje de programación, son:

Es ***Simple:***

Java ofrece toda la funcionalidad de un lenguaje potente, pero sin las características menos usadas y más confusas de éstos. C++ es un lenguaje que adolece de falta de seguridad, pero C y C++ son lenguajes más difundidos, por ello Java se diseñó para ser parecido a C++ y así facilitar un rápido y fácil aprendizaje.

Java elimina muchas de las características de otros lenguajes como C++, para mantener reducidas las especificaciones del lenguaje y añadir características muy útiles como el reciclador de memoria dinámica (*garbage collector en inglés*). No es necesario preocuparse de liberar memoria, el reciclador se encarga de ello y como es un *thread (hilo)* de baja prioridad, cuando entra en acción, permite liberar bloques de memoria muy grandes, lo que reduce la fragmentación de la memoria.

Java reduce en un 50% los errores más comunes de programación con lenguajes como C y C++ al eliminar muchas de las características de éstos, entre las que destacan:

- aritmética de punteros
- no existen referencias
- registros (struct)

- definición de tipos (typedef)
- macros (#define)
- necesidad de liberar memoria (free)

Aunque, en realidad, lo que hace es eliminar las palabras reservadas (struct, typedef), ya que las clases son algo parecido. Además, el intérprete completo de Java que hay en este momento es muy pequeño, solamente ocupa 215 Kb de RAM.

Es ***Orientado a Objetos***:

Java implementa la tecnología básica de C++ con algunas mejoras y elimina algunas cosas para mantener el objetivo de la simplicidad del lenguaje. Java trabaja con sus datos como objetos y con interfaces a esos objetos. Soporta tres características propias del paradigma de la orientación a objetos: encapsulamiento, herencia y polimorfismo. Las plantillas de objetos son llamadas, como en C++, *clases* y sus copias, *instancias.* Estas instancias, como en C++, necesitan ser construidas y destruidas en espacios de memoria. Java incorpora funcionalidades inexistentes en C++ como por ejemplo, la resolución dinámica de métodos. Esta característica deriva del lenguaje Objective C, propietario del sistema operativo Next.

En C++ se suele trabajar con librerías dinámicas (DLLs) que obligan a recompilar la aplicación cuando se retocan las funciones que se encuentran en su interior. Este inconveniente es resuelto por Java mediante una interfaz específica llamada RTTI (*RunTime Type Identification*) que define la interacción entre objetos excluyendo variables de instancias o implementación de métodos. Las clases en Java tienen una representación en el run-time que permite a los programadores interrogar por el tipo de clase y enlazar dinámicamente la clase con el resultado de la búsqueda.

Es ***Distribuido***:

Java se ha construido con extensas capacidades de interconexión TCP/IP. Existen librerías de rutinas para acceder e interactuar con protocolos como *http* y *ftp*. Esto permite a los programadores acceder a la información a través de la red con tanta facilidad como a los ficheros locales.

La verdad es que Java en sí no es distribuido, sino que proporciona las librerías y herramientas para que los programas puedan ser distribuidos, es decir, que se corran en varias máquinas, interactuando.

Es ***Robusto***:

Java realiza verificaciones en busca de problemas tanto en tiempo de compilación como en tiempo de ejecución. La comprobación de tipos en Java ayuda a detectar errores, lo antes posible, en el ciclo de desarrollo. Java obliga a la declaración explícita de métodos, reduciendo así las posibilidades de error. Maneja la memoria para eliminar las preocupaciones por parte del programador

de la liberación o corrupción de memoria. También implementa los *arreglos auténticos*, en vez de listas enlazadas de punteros, con comprobación de límites, para evitar la posibilidad de sobrescribir o corromper memoria resultado de punteros que señalan a zonas equivocadas. Estas características reducen drásticamente el tiempo de desarrollo de aplicaciones en Java.

Además, para asegurar el funcionamiento de la aplicación, realiza una verificación de los *byte-codes*, que son el resultado de la compilación de un programa Java. Es un código de máquina virtual que es interpretado por el intérprete Java. No es el código máquina directamente entendible por el hardware, pero ya ha pasado todas las fases del compilador: análisis de instrucciones, orden de operadores, entre otras, y ya tiene generada la pila de ejecución de órdenes.

Java proporciona, pues:

- Comprobación de punteros
- Comprobación de límites de arreglos
- Excepciones
- Verificación de byte-codes

Es de ***Arquitectura Neutral***:

Para establecer Java como parte integral de la red, el compilador Java compila su código a un fichero objeto de formato independiente de la arquitectura de la máquina en que se ejecutará. Cualquier máquina que tenga el sistema de ejecución (*run-time*) puede ejecutar ese código objeto, sin importar en modo alguno la máquina en que ha sido generado. Actualmente existen sistemas run-time para Solaris 2.x, SunOs 4.1.x, Windows 95, Windows NT, Linux, Irix, Aix, Mac, Apple y probablemente haya grupos de desarrollo trabajando en la portabilidad a otras plataformas.

Es ***Seguro:***

La seguridad en Java tiene dos facetas. En el lenguaje, características como los punteros o el *casting* implícito que hacen los compiladores de C y C++ se eliminan para prevenir el acceso ilegal a la memoria. Cuando se usa Java para crear un navegador, se combinan las características del lenguaje con protecciones de sentido común aplicadas al propio navegador.

El lenguaje C, por ejemplo, tiene lagunas de seguridad importantes, como son los *errores de alineación.* Los programadores de C utilizan punteros en conjunción con operaciones aritméticas. Esto le permite al programador que un puntero referencie a un lugar conocido de la memoria y pueda sumar (o restar) algún valor, para referirse a otro lugar de la memoria. Si otros programadores conocen las estructuras de datos pueden extraer información confidencial del sistema. Con un lenguaje como C, se pueden tomar números enteros aleatorios y convertirlos en punteros para luego acceder a la memoria:

El código Java pasa muchas pruebas antes de ejecutarse en una máquina. El código se pasa a través de un verificador de byte-codes que comprueba el formato de los fragmentos de código y aplica un probador de teoremas para detectar fragmentos de código ilegal -código que falsea punteros, viola derechos de acceso sobre objetos o intenta cambiar el tipo o clase de un objeto-.

El Cargador de Clases también ayuda a Java a mantener su seguridad, separando el espacio de nombres del sistema de ficheros local, del de los recursos procedentes de la red. Esto limita cualquier aplicación del tipo *Caballo de Troya*, ya que las clases se buscan primero entre las locales y luego entre las procedentes del exterior.

En resumen, las aplicaciones de Java resultan extremadamente seguras, ya que no acceden a zonas delicadas de memoria o de sistema, con lo cual evitan la interacción de ciertos virus. Java no posee una semántica específica para modificar la pila de programa, la memoria libre o utilizar objetos y métodos de un programa sin los privilegios del kernel del sistema operativo. Además, para evitar modificaciones por parte de los crackers de la red, implementa un método ultra seguro de autentificación por clave pública. El Cargador de Clases puede verificar una firma digital antes de realizar una instancia de un objeto. Por tanto, ningún objeto se crea y almacena en memoria, sin que se validen los privilegios de acceso. Es decir, la seguridad se integra en el momento de compilación, con el nivel de detalle y de privilegio que sea necesario.

Es ***Portable***:

Más allá de la portabilidad básica por ser de arquitectura independiente, Java implementa otros estándares de portabilidad para facilitar el desarrollo. Los enteros son siempre *enteros* y además, enteros de 32 bits en complemento a 2. Además, Java construye sus interfaces de usuario a través de un sistema abstracto de ventanas de forma que las ventanas puedan ser implantadas en entornos Unix, Pc o Mac.

Es ***Interpretado*** :

El intérprete Java (sistema run-time) puede ejecutar directamente el código objeto. Enlazar un programa, normalmente, consume menos recursos que compilarlo, por lo que los desarrolladores con Java pasarán más tiempo desarrollando y menos esperando por el ordenador.

Java para conseguir ser un lenguaje independiente del sistema operativo y del procesador que incorpore la máquina utilizada, es tanto interpretado como compilado. Y esto no es ningún contrasentido, me explico, el código fuente escrito con cualquier editor se compila generando el byte-code. Este código intermedio es de muy bajo nivel, pero sin alcanzar las instrucciones de máquina, propias de cada plataforma y no tiene nada que ver con el p-code de Visual Basic. El byte-code corresponde al 80% de las instrucciones de la aplicación. Ese mismo código es el que se puede

ejecutar sobre cualquier plataforma. Para ello hace falta el run-time, que sí es completamente dependiente de la máquina y del sistema operativo, que interpreta dinámicamente el byte-code y añade el 20% de instrucciones que faltaban para su ejecución. Con este sistema es fácil crear aplicaciones multiplataforma, pero para ejecutarlas es necesario que exista el run-time correspondiente al sistema operativo utilizado.

Es ***Multi-Hilos (Multi-Threaded en inglés)***:

Al ser multi-hilos, Java permite muchas actividades simultáneas en un programa. Los hilos (a veces llamados, procesos ligeros), son básicamente pequeños procesos o piezas independientes de un gran proceso. Al estar los hilos construidos en el lenguaje, son más fáciles de usar y más robustos que sus homólogos en C o C++.

El beneficio de ser multi-hilos consiste en un mejor rendimiento interactivo y mejor comportamiento en tiempo real. Aunque el comportamiento en tiempo real está limitado a las capacidades del sistema operativo subyacente (Unix, Windows, etc.), aún supera a los entornos de flujo único de programa tanto en facilidad de desarrollo como en rendimiento.

Es ***Dinámico***:

Java se beneficia todo lo posible de la tecnología orientada a objetos. Java no intenta conectar todos los módulos que comprenden una aplicación hasta el tiempo de ejecución. Las librerías nuevas o actualizadas no paralizarán las aplicaciones actuales (siempre que mantengan el API anterior).

Java también simplifica el uso de protocolos nuevos o actualizados. Si su sistema ejecuta una aplicación Java sobre la red y encuentra una pieza de la aplicación que no sabe manejar, tal como se ha explicado en párrafos anteriores, Java es capaz de traer automáticamente cualquiera de esas piezas que el sistema necesita para funcionar.

1.11 IDEs que usan Java.

Entre los IDE (en inglés integrated development environment) de código abierto que explotan las bondades de Java y son usados y reconocidos a escala mundial, para el desarrollo sobre plataformas libres, se encuentran las versiones de Eclipse y NetBeans.

Ambas comunidades de desarrollo, contemplan en su haber un conjunto de versiones con herramientas de una amplia gama de facilidades, enfocadas en las necesidades de los desarrolladores, para que puedan escoger según el tipo de proyecto que llevan a cabo. Entre las ventajas que más se aprovechan a nivel mundial se encuentran:

-Código abierto.

-Portabilidad en sistemas operativos como: Windows, Linux y Mac OS.X.

-Editor de Java validado: basados en Java SE, Java EE y Java ME.

-Integración con otros lenguajes de programación como C, C++, Ruby y Java Script.

-Completamiento de código y editor XML.

-Soporte directo y constante.

-Mecanismo integrado de actualización online.

-Despliegue y administración del SW a través de su ciclo de vida.

-Instalación simple.

1.11.1 NetBeans IDE 6.0

La integración de todos los productos NetBeans en un solo IDE se ha logrado en NetBeans 6.0. En este IDE se encuentran las herramientas que permiten crear aplicaciones profesionales Desktop, Enterprise, Web y aplicaciones móviles. Incluye una completa reestructuración de la infraestructura del editor, soporte para lenguajes adicionales, una fácil instalación y configuración del IDE donde encuentras lo que necesitas. Entre los rasgos y mejoras que lo caracterizan se encuentran:

1. Código abierto y libre.
2. Poderoso constructor de interfaz gráfica de usuario (en inglés grafic user interface, GUI).
3. Soporte para los estándares de Java y plataformas: Java ME, Java EE y Java SE.
4. Herramientas de rendimiento y eliminación de errores: asistencia experta para optimizar la velocidad y memoria de sus aplicaciones.
5. Poderoso editor Ruby y soporte: completamiento de código y depurado.
6. Soporte de SOA.
7. Plataforma extensible: capacidad para añadir nuevos rasgos y configuraciones o mantener una existente.
8. Adaptar con facilidad proyectos fuera del IDE de desarrollo.
9. Incluye otros lenguajes de programación como C++, C, JavaScript, Ruby, entre otros.
10. Capacidad de generación de documentación a partir del código.

El enfoque de NetBeans IDE 6.0 es la productividad del desarrollador con un editor más inteligente y rápido. Incluye soporte total para las nuevas tecnologías de sobremesa, como Beans Binding y el entorno Swing Application Framework integrado.

Otras características que esta versión incluye son: mejor edición del código, capacidades de navegación e inspección, historia local, soporte integrado para Subversion, y mayores capacidades de personalización integradas en la distribución estándar.

Este IDE reconoce los directorios de las herramientas de control de versiones automáticamente. Un menú permite actualizar, chequear, comparar y eliminar ficheros, además de subir al directorio del proyecto las nuevas modificaciones.

Permite elegir entre tres tipos de herramientas de control de versiones: CVS, Subversion o Mercurial.

1.12 Herramientas de Control de Versiones.

1.12.1 Subversion

Subversion es un sistema de control de versiones de código fuente abierto y libre. Subversion maneja ficheros y directorios a través del tiempo. Hay un árbol de ficheros en un repositorio central. Esto le permite recuperar versiones antiguas de sus datos, o examinar el historial de cambios de los mismos. Subversion puede acceder al repositorio a través de redes, lo que le permite ser usado por personas que se encuentran en distintos ordenadores. A cierto nivel, la capacidad para que varias personas puedan modificar y administrar el mismo conjunto de datos desde sus respectivas ubicaciones fomenta la colaboración. Se puede progresar mas rápidamente sin un único conducto por el cual deban pasar todas las modificaciones; si se ha hecho un cambio incorrecto a los datos, simplemente se deshace. Subversion es un sistema general que puede ser usado para administrar cualquier conjunto de ficheros.

Subversion proporciona:

Versionado de directorios: implementa un sistema de ficheros versionado “virtual” que sigue los cambios sobre árboles de directorios completos a través del tiempo. Ambos, ficheros y directorios, se encuentran bajo el control de versiones.

Verdadero historial de versiones: puede añadir, borrar, copiar, y renombrar ficheros y directorios. Y cada fichero nuevo añadido comienza con un historial nuevo, limpio y completamente suyo.

Envíos atómicos: una colección cualquiera de modificaciones o bien entra por completo al repositorio, o bien no lo hace en absoluto. Esto permite a los desarrolladores construir y enviar los cambios como fragmentos lógicos e impide que ocurran problemas cuando sólo una parte de los cambios enviados lo hace con éxito.

Versionado de metadatos: cada fichero y directorio tiene un conjunto de propiedades, claves y sus valores, asociados a él. Usted puede crear y almacenar cualquier par arbitrario de clave/valor que desee. Las propiedades son versionadas a través del tiempo, al igual que el contenido de los ficheros.

Manipulación consistente de datos: expresa las diferencias del fichero usando un algoritmo de diferenciación binario, que funciona idénticamente con ficheros de texto (legibles para humanos) y ficheros binarios (ilegibles para humanos). Ambos tipos de ficheros son almacenados igualmente comprimidos en el repositorio, y las diferencias son transmitidas en ambas direcciones a través de la red.

Ramificación y etiquetado eficientes: el coste de ramificación y etiquetado no necesita ser proporcional al tamaño del proyecto. Crea ramas y etiquetas simplemente copiando el proyecto, usando un mecanismo similar al enlace duro. De este modo estas operaciones toman solamente una cantidad de tiempo pequeña y constante.

La herramienta además es compatible con los siguientes protocolos: http://; https://; svn://; svn+ssh://.

1.12.2 TortoiseSVN

Es un cliente gratuito de código abierto para el sistema de control de versiones Subversion. Maneja ficheros y directorios a lo largo del tiempo. Los ficheros se almacenan en un repositorio central. El repositorio es prácticamente lo mismo que un servidor de ficheros ordinario, salvo que recuerda todos los cambios que se hayan hecho a sus ficheros y directorios. Esto permite que pueda recuperar versiones antiguas de sus ficheros y examinar la historia de cuándo y cómo cambiaron sus datos, y quién hizo el cambio. Es fácil de usar, permite ver el estado de los archivos desde el explorador de Windows y permite también crear gráficos de todas las revisiones asignadas.

1.13 Herramientas de Gestión de Proyecto

1.13.1 DotProject

DotProject fue creado en el año 2000, con el fin de construir una herramienta para la Gestión de Proyectos. Está construido por aplicaciones de código abierto y es mantenida por un pequeño, pero dedicado grupo de voluntarios. Es una aplicación basada en web, multiusuario, soporta varios lenguajes y es software libre.

Esta programada en PHP, y utiliza MySQL como base de datos (aunque otros motores como Postgres también pueden ser utilizados). La plataforma recomendada para utilizar dotProject se denomina LAMP (Linux + Apache + MySQL + PHP). De todas formas, existen versiones para instalar dotProject bajo otros sistemas operativos, tales como Microsoft Windows (NT,2000,XP) y MacOS.

Es una herramienta orientada a la Gestión de Proyectos. Para eso se orienta a la administración de recursos para desarrollar un producto, cuya producción requiera de un conjunto de actividades o tareas que se desarrollen entre ellas en forma paralela o independiente.

La aplicación consta de un conjunto de entidades ordenadas jerárquicamente las cuales permiten brindar la funcionalidad del producto.

A continuación se mencionan las entidades más importantes de dotProject:

* Compañías: Son las entidades que agrupan proyectos, actividades y usuarios.

* Departamentos: Son áreas dentro de las compañías, que permiten agrupar usuarios en dicho nivel.

*Usuarios/Contactos: dotProject tiene usuarios los cuales son capaces de autentificarse en dotProject y trabajar dentro del esquema de permisos que posea el rol de dicho usuario. Los contactos son usuarios especiales que, asignados a un determinado proyecto pueden recibir por ejemplo: correo, actualizaciones y noticias pero no necesariamente deben tener acceso al sistema. Los usuarios y contactos pertenecen a una compañía.

* Proyectos: Es la entidad que contiene el grupo de tareas necesarias para desarrollar un determinado producto.

* Actividades: son las tareas asignadas dentro de un proyecto. Son los componentes sobre los cuales se controla: la duración, dependencias, recursos asignados y progreso. Las actividades deben de pertenecer a un único proyecto.

* Diagramas de Gantt: Permite ver en forma gráfica las actividades ordenadas jerárquicamente, mostrando las dependencias y solapamientos de las mismas.

* Tickets: para administrar todos los problemas relacionados a un proyecto.

* Archivos: Permite almacenar archivos dentro de un proyecto permitiendo un versionado básico de los mismos.

* Foros: Permite la creación de foros de discusión dentro de cada proyecto para distribuir información y discutir temas relativos al proyecto del foro.

* Administración del Sistema: Contiene la actividades relacionadas a la administración de usuarios, roles y configuración del sistema.

* Recursos: Permite asignar recursos no humanos (oficinas, equipamiento, etc) a un proyecto.

1.13.2 OpenProject

Es un programa de escritorio para la gestión de proyectos: gratuito, código abierto, con versiones para Linux, Unix, Mac y Windows; compatible con ficheros MS Project y con todas las

funcionalidades que ofrece Project (como aplicación de escritorio stand-alone). Sin duda el rival más serio de Microsoft Project.

¿Por qué escoger DotProject?

Se escogió DotProject como herramienta para la gestión de proyecto, pues es la que la universidad tiene oficializada para su uso en todos los proyectos de software de la UCI.

1.14 Frameworks

En el desarrollo de software, Marco de trabajo o Framework en su término en inglés, es una estructura de soporte definida en la cual otro proyecto de software puede ser organizado y desarrollado. Típicamente, un framework puede incluir soporte de programas, bibliotecas y un lenguaje interpretado, entre otros software para ayudar a desarrollar y unir los diferentes componentes de un proyecto. 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.

El interés en reutilizar software ha sido cambiado de la reutilización de componentes simples a diseño de sistemas enteros o estructuras de aplicaciones. Un sistema software que pudiera ser reutilizado en este nivel para la creación de aplicaciones completas es llamado framework. Los frameworks están basados en la idea que deberían permitir la producción fácil de un conjunto de sistemas específicos pero similares, dentro de un cierto dominio comenzando desde una estructura genérica.

Brevemente, los frameworks son arquitecturas genéricas integradas por un extensible conjunto de componentes. Además, los frameworks pueden contener subframeworks los cuales representen subconjuntos de componentes de un sistema más grande. [KAISER 2005]

1.15 Sistema Gestor de Base de Datos (SGBD)

Un Sistema Gestor de Base de Datos (SGBD) es un conjunto de programas que permite crear y mantener una Base de Datos, asegurando su integridad, confidencialidad y seguridad. Las principales funciones que debe cumplir un SGBD son la creación y mantenimiento de la base de datos, el control de accesos, la manipulación de datos de acuerdo con las necesidades del usuario, el cumplimiento de las normas de tratamiento de datos, evitar redundancias e inconsistencias.

Clasificación de los SGBD

Según modelos conceptuales.

• Relacional

• Redes

• Jerárquico

• Orientado a Objetos

• Otros

Según el número de usuarios.

• Monousuario.

• Multiusuario.

Según el número de sitios.

• Centralizado.

• Distribuido.

Según el uso de Software.

• Homogéneo.

• Heterogéneo.

Entre los SGBD más reconocidos se haya PostgresSQL, Oracle, Microsoft SQL Server, y MySQL.

1.15.1 Sistemas de Base de Datos Embebida

Se entiende por bases de datos embebidas, o empotradas, aquellas que no inician un servicio en la máquina independiente de la aplicación, pudiéndose enlazar directamente al código fuente o bien utilizarse en forma de librería.

Normalmente las bases de datos empotradas comparten una serie de características comunes: su pequeño tamaño, la huella pequeña (small footprint, en inglés) en términos de tamaño de código añadido a la aplicación y recursos que consumen. La característica fundamental que distingue a estas base de datos embebidas, es que requieren de poca o ninguna administración por parte del usuario final.

Hoy en día existen en el mercado una gran variedad de bases de datos relacionales diseñadas específicamente para su uso de manera embebida, como por ejemplo:

- Sybase SQL Anywhere.
- InterSystems cache.
- PervasiveSQL.
- Microsoft Jet Engine.

Otros lo que han hecho es modificar bases de datos existentes para crear variantes que funcionen de manera embebida. Algunos ejemplos de estos incluyen:

- IBM DB2 Everyplace
- Oracle 10g
- Microsoft SQL Server Desktop Engine

1.15.1.1 HSQLDB

HSQLDB es un motor de base de datos relacionales de SQL escrito en JAVA. Ofrece un motor de base de datos pequeño (menos de 100 Kb en una versión para applets). Es rápido y permite tener tablas en memoria y en disco. Está diseñado para funcionar como base de datos embebida o como servidor, según se desee.

Adicionalmente incluye algunas herramientas, como, un pequeño servidor web, consultas en memoria y otras herramientas para la administración. [HSQLDB Development Group, 2008].

El producto actualmente es utilizado como base de datos y motor de persistencia en muchos proyectos de Open Source Software. La suite ofimática OpenOffice.org lo incluye desde su versión 2.0 para dar soporte a la aplicación Base. La versión actual de HSQLDB (1.8.0.10) es sumamente estable, fidedigna y compacta.

El software es de código abierto y la única restricción que imponen sus desarrolladores originales (HSQLDB Development Group) es que no se use su nombre para distribuir las versiones del software modificadas por terceros.

Características de HSQLDB

HSQLDB puede correr de forma general en dos modos:

- In-process mode: Este modo es comúnmente llamado: de aplicación. En este caso es cuando funciona como base de datos embebida como parte de la aplicación dentro de la misma Máquina Virtual de Java.
- Server Mode: Es el modo de servidor. En este modo puede configurarse como servidor de BD o como servidor web.

Cada base de datos dentro de HSQLDB tiene de 2 a 5 archivos, que poseen el mismo nombre, pero diferentes extensiones. A continuación se explica la función de cada uno de esos archivos:

- **Nombre_BD.properties:** contiene las diferentes configuraciones de la base de datos.
- **Nombre_BD.script :** contiene la definición de las tablas y otros objetos de la BD.
- **Nombre_BD.log :** contiene los cambios recientes hechos sobre la BD.
- **Nombre_BD.data :** contiene los datos de las tablas en memoria Caché.

- **Nombre_BD.bakcup:** contiene una copia de seguridad del último estado consistente de los datos en la BD.

HSQLDB soporta el dialecto SQL definido en el Estándar de SQL del 92, 99, y 2003.

El software posee tres modos de tablas persistentes:

Tablas Memoria (Memory tables, en inglés): Son el tipo de tabla que se crea por defecto cuando se usa el comando CREATE TABLE. Sus datos están completamente escritos en memoria, pero cualquier cambio en su estructura o contenido son escritos en el fichero Nombre_BD.script. La próxima vez que se abra la aplicación se leerá el fichero Nombre_BD.script y la tabla será recreada nuevamente en memoria.

Tablas Cached (CACHED tables, en inglés): son creadas con el comando CREATE CACHED TABLE. Solo una parte de la tabla, o su indexado es cargada en memoria. Evitando así que se ocupen varios megabytes de memoria RAM.

Tablas Texto (TEXT tables, en inglés): Este formato de tablas usa un CVS (Separador de valores por coma, Comma Separate Value en inglés) como su fuente de datos. Se puede especificar un fichero CVS existente, así como usar los datos de otra base de datos o programa, para llenar los datos de la tabla. También puedes definir un fichero vacío que será llenado con los datos por el motor de la base de datos. HSQLDB tiene una alta integración con el framework de persistencia de datos, Hibernate.

1.15.1.2 SQLite

SQLite es un librería de software escrita en el lenguaje de programación C, que trabaja como base de datos embebida, por lo cual en lugar de funcionar como un proceso independiente, este opera de manera simbiótica dentro de la aplicación. Lo que quiere decir que su código está entrelazado, o incrustado como parte del programa que lo alberga. [SQLite, 2007]

SQLite es diferente a todos los demás motores de búsqueda, pues su principal objetivo es ser simple.

- Simple de administrar.
- Simple de operar.
- Simple de incrustar en largos programas.
- Simple de mantener y personalizar.

La librería implementa la mayor parte del estándar SQL-92, incluyendo transacciones de base de datos atómicas, consistencia de base de datos, aislamiento, y durabilidad (ACID), triggers y la mayor parte de las consultas complejas.

La librería puede ser usada desde programas C/C++, Perl, Pyton, Delphi y PHP. SQLite tiene como desventajas que no brinda un conjunto de funcionalidades que muchos usuarios consideran muy útiles, como por ejemplo:

- Alta concurrencia.
- Funciones incorporadas.
- Procedimientos almacenados.
- Extensiones de XML y/o JAVA.

1.15.1.3 Comparando HSQLDB y SQLite.

- HSQLDB está completamente escrito en Java y puede ser usado desde aplicaciones Java. Cosa que no sucede con SQLite, que, además de estar escrito en C, actualmente no hay ninguna librería que le permite correr desde aplicaciones Java.
- SQLite no se puede permitir manipular extensiones de XML, a diferencia de HSQLDB.
- Para SQLite es imposible manipular procedimientos almacenados, funcionalidad imprescindible para muchos desarrolladores, y que HSQLDB maneja de una forma cómoda.
- HSQLDB puede correr en modo servidor, funcionalidad que puede ser aprovechada en un futuro para desarrollar una versión de la aplicación en modo cliente servidor.
- HSQLDB tiene una alta integración con el framework de persistencia de datos, Hibernate, framework que posteriormente se propondrá usar para el desarrollo de la aplicación.

Por todo lo antes expuesto se puede llegar a la conclusión de que HSQLDB es la mejor opción para almacenar los datos de la aplicación que se propone desarrollar.

1.16 Conectividad a bases de datos

1.16.1 Conectividad abierta de la base de datos (ODBC, en sus siglas en inglés)

Open Database Connectivity (ODBC) es un estándar de acceso a bases de datos desarrollado por Microsoft Corporation, el objetivo de ODBC es hacer posible el acceder a cualquier dato desde cualquier aplicación, sin importar qué Sistema Gestor de Bases de Datos (DBMS, por sus siglas en inglés) almacene los datos, ODBC logra esto al insertar una capa intermedia llamada manejador de Bases de Datos, entre la aplicación y el DBMS, el propósito de esta capa es traducir las consultas de datos de la aplicación en comandos que el DBMS entienda. Para que esto funcione, la aplicación como

el DBMS, deben ser compatibles con ODBC: esto es, que la aplicación debe ser capaz de producir comandos ODBC y el DBMS debe ser capaz de responder a ellos.

1.16.2 Conectividad de Java a bases de datos (JDBC, de sus siglas en inglés)

JDBC es el acrónimo de Java Database Connectivity, un API que permite la ejecución de operaciones sobre bases de datos desde el lenguaje de programación Java independientemente del sistema operativo donde se ejecute o de la base de datos a la cual se accede utilizando el dialecto SQL del modelo de base de datos que se utilice.

El API JDBC se presenta como una colección de interfaces Java y métodos de gestión de manejadores de conexión hacia cada modelo específico de base de datos. Un manejador de conexiones hacia un modelo de base de datos en particular, es un conjunto de clases que implementan interfaces Java y que utilizan los métodos de registro para declarar los tipos de localizadores a base de datos (URL) que pueden manejar. Para utilizar una base de datos particular, el usuario ejecuta su programa junto con la librería de conexión apropiada al modelo de su base de datos, y accede a ella estableciendo una conexión, para ello provee el localizador de la base de datos y los parámetros de conexión específicos.

1.17 Generadores de Reportes

En cualquier actividad de auditoría es indispensable y obligatorio generar reportes de las anomalías, o resultados finales del procedimiento. Por eso, una aplicación que servirá de herramienta en un proceso de auditoría a un sistema contable y financiero deberá tener la capacidad de generar reportes en diferentes formatos. Por todo esto, a continuación se analizarán una serie de herramientas de generación de reportes, candidatas, cada una de ellas, a ser seleccionada como la que se utilizará en el desarrollo de la aplicación.

1.17.1 Crystal Reports

Crystal Reports está diseñado para trabajar con una base de datos y ayudar en el análisis y la interpretación de la información importante. Facilita la creación de informes simples y dispone también de funciones poderosas necesarias para generar informes complejos o especializados. Pertenece a la categoría de software privado y una licencia de Crystal Reports Professional cuesta 499 dólares.

Esta herramienta puede importar datos de una amplia gama de ficheros:

- ACCES (mdb),
- dBASE (5.0, III y IV)
- EXCEL (xls, .xlw)

- HTML
- XML
- PARADOX (3.x, 4.x, y 5.x)
- LOTUS (wk1, wk3, wk4)
- BTRIEVE (ddf)

Crystal Reports puede tomar también los datos desde bases de datos directamente mediante: SQL/ODBC o ADO. Además se puede importar un objeto con formato y basado en texto desde un archivo existente. Otra ventaja que ofrece es la manipulación de los datos y el manejo de los campos, se podrán insertar campos de fórmula, campos de enunciado, campos de parámetro, se puede además insertar campos (de un mismo tipo) en un objeto de texto. Los campos que son insertados en un objeto de texto, son acortados automáticamente (no tienen espacios en blanco al principio, ni al final).

1.17.2 Agata Reports

Es una aplicación liberada bajo la licencia GNU GPL, está escrita en PHP GTK y se soporta tanto en Windows como Linux.

Agata Reports es un conjunto de herramientas para sacar lo más sustantivo de las bases de datos. Permite extraer datos de:

- PostgreSQL
- MySQL
- Oracle
- DB2
- MS-SQL
- Informix
- Internase
- Sybase
- Frontbase

La información, para ser visualizada permite exportarla a:

- PostScript
- Texto
- HTML
- XML

- PDF
- CSV.

Esta es una herramienta potente y especializada en la importación desde las bases de datos

1.17.3 JasperReports

Jasper Reports es una herramienta gratuita y open source que se compone de un conjunto de librerías Java para facilitar la generación de informes. Es capaz de extraer la información desde diferentes bases de datos instaladas, o ficheros (Excel y Acces).

Uno de los puntos fuertes de JasperReports es el apoyo creciente de la comunidad, se espera su desarrollo posterior a diferentes versiones más potentes.

Cada reporte que se genera tiene definido su diseño en un archivo XML, por convención su extensión es .jrxml, o sea la herramienta crea una extensión de ficheros propia. Es capaz de generar, compilar, modificar y exportar el fichero. Puede exportar a las siguientes extensiones:

- PDF
- XML
- HTML
- CSV
- XLS

Lo más interesante de esta herramienta es la creación de ficheros con extensión propia y la capacidad de exportar a diferentes ambientes.

JasperReports tiene un fuerte soporte para la internacionalización provisto por la plataforma Java. Soporta todos los lenguajes mundiales por la codificación Unicode (UTF-8). Así como otras codificaciones nativas como ISO-8859-1 (Latin 1) y para los lenguajes del Oeste Europeo.

Los datos para llenar los reportes pueden ser provistos como parámetros de otra aplicación; o desde alguna fuente de datos (data source, en inglés) definida dentro de JasperReport. La herramienta tiene incluida, el driver JDBC que conoce como extraer los datos de Sistemas de Base de Datos Relacionales, Java Beans (EJB, Hibernate) y XML.

¿Por qué usar JasperReport?

JasperReports, además de ser una herramienta gratuita, tiene la ventaja de estar hecha completamente en Java, lo que significa una mayor compatibilidad con el IDE con el cual se desarrollar la aplicación. Otro motivo por el cual debe ser JasperReports la herramienta para generar los reportes viene dado por su compatibilidad con el framework de persistencia de datos Hibernate que posteriormente se propondrá en la arquitectura. Además de que el único requerimiento que necesita

JasperReports para su funcionamiento es la Máquina Virtual de Java corriendo, requisito que coincide con la del IDE de desarrollo.

1.18 Conclusiones.

Para el desarrollo de este Software General de Auditoría Cubano se propone una aplicación de escritorio, mono-usuario. A desarrollar sobre lenguaje Java usando el IDE Netbeans 6.0 y siguiendo la metodología de desarrollo RUP.

La arquitectura a proponer se basa en el estilo N-Capas específicamente contará con 3 capas: Interfaz de Usuario, Lógica de Negocios y Acceso a datos. Para la comunicación entre capas se usarán interfaces, permitiendo un desarrollo paralelo e independiente.

Para lograr la persistencia de los datos se utilizará un gestor de base de datos embebido, lo que mejorará considerablemente los tiempos de respuesta de la aplicación.

CAPÍTULO 2: REPRESENTACION DE LA ARQUITECTURA DEL SISTEMA

2.1 Introducción.

Durante el desarrollo de este capitulo se hará un análisis de dos de las tareas fundamentales que debe realizar el arquitecto del proyecto; estas son:

- ✓ Definición de la Línea Base de la Arquitectura.
- ✓ Elaboración del documento Descripción de la Arquitectura.

2.2 Línea Base de la Arquitectura

La Línea Base de la Arquitectura es un esqueleto del sistema con pocos músculos de software, pues no es más que la versión interna del sistema al final de la fase de elaboración, en fin tiene las versiones de todos los modelos que un sistema terminado contiene al final de la fase de construcción. Incluye el mismo esqueleto de subsistemas, componentes y nodos que un sistema definitivo, pero no existe toda la musculatura. La definición de la línea base de la arquitectura es uno de los dos aspectos más importante que debe desarrollar el rol de arquitecto.

En la misma se exponen los estilos arquitectónicos de la aplicación, así como los principales elementos de la arquitectura. Se describen los principales patrones de arquitectura utilizados y las tecnologías y herramientas de software que se utilizarán en el sistema a desarrollar.

La Línea Base de la Arquitectura describe la estructura del sistema en un alto nivel de abstracción. Describe detalladamente el organigrama de la arquitectura según los estilos arquitectónicos que se utilizarán.

2.2.2 Frameworks de Desarrollo

Otro de los niveles de abstracción que se mencionó en el capitulo anterior fueron los frameworks, que no son más que arquitecturas definidas para un determinado dominio de aplicación que contiene un conjunto de componentes implementados y sus interfaces bien definidas, estos componentes se pueden utilizar, redefinir y crear nuevos componentes.

A partir de la estructura del sistema propuesto, en capas, se propone la utilización de tres frameworks distribuidos en cada una de esas capas.

- ✓ Capa Presentación: Framewok Swing.
- ✓ Capa Negocio: Framework Spring.
- ✓ Capa Acceso a Datos: Framework Hibernate.

2.2.2.1 Framework Swing

Características [Gomez, 2006]

- Amplia variedad de componentes: En general las clases que comiencen por "J" son componentes que se pueden añadir a la aplicación. Por ejemplo: JButton.

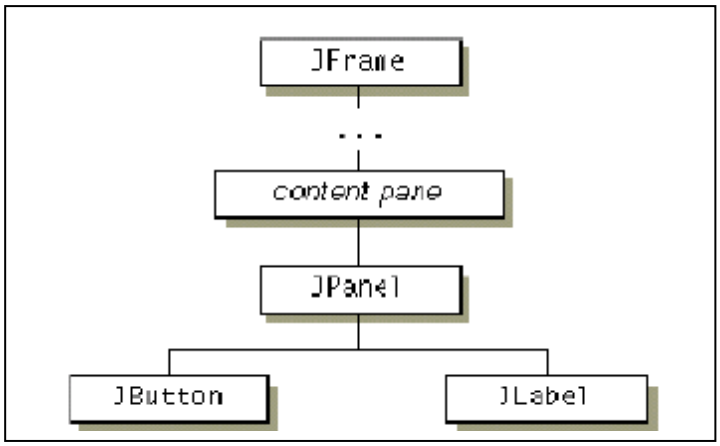


Fig. 4- Jerarquía de clases en Swing

- Aspecto modificable (look and feel): Se puede personalizar el aspecto de las interfaces o utilizar varios aspectos que existen por defecto (Metal Max, Basic Motif, Window Win32).
- Arquitectura **Modelo-Vista-Controlador**: Esta arquitectura da lugar a todo un enfoque de desarrollo muy arraigado en los entornos gráficos de usuario realizados con técnicas orientadas a objetos. Cada componente tiene asociado una clase de modelo de datos y una interfaz que utiliza. Se puede crear un modelo de datos personalizado para cada componente, con sólo heredar de la clase Model.
- Gestión mejorada de la entrada del usuario: Se pueden gestionar combinaciones de teclas en un objeto KeyStroke y registrarlo como componente. El evento se activará cuando se pulse dicha combinación si está siendo utilizado el componente, la ventana en que se encuentra o algún hijo del componente.
- Objetos de acción (action objects): Estos objetos cuando están activados (enabled) controlan las acciones de varios objetos componentes de la interfaz. Son hijos de ActionListener.
- Contenedores anidados: Cualquier componente puede estar anidado en otro. Por ejemplo, un gráfico se puede anidar en una lista.

- Escritorios virtuales: Se pueden crear escritorios virtuales o "interfaz de múltiples documentos" mediante las clases JDesktopPane y JInternalFrame.

- Bordes complejos: Los componentes pueden presentar nuevos tipos de bordes. Además el usuario puede crear tipos de bordes personalizados.
- Diálogos personalizados: Se pueden crear multitud de formas de mensajes y opciones de diálogo con el usuario, mediante la clase JOptionPane.
- Clases para diálogos habituales: Se puede utilizar JFileChooser para elegir un fichero, y JColorChooser para elegir un color.
- Componentes para tablas y árboles de datos: Mediante las clases JTable y JTree.
- Potentes manipuladores de texto: Además de campos y áreas de texto, se presentan campos de sintaxis oculta JPassword, y texto con múltiples fuentes JTextPanel. Además hay paquetes para utilizar ficheros en formato HTML o RTF.
- Capacidad para "deshacer": En gran variedad de situaciones se pueden deshacer las modificaciones que se realizaron.
- Soporte a la accesibilidad: Se facilita la generación de interfaces que ayuden a la accesibilidad de discapacitados, por ejemplo en Braille.

Todas las clases componentes de Swing (clases hijas de JComponent), son hijas de la clase Component de AWT.

Modelo del Framework

El Framework Swing tiene el siguiente modelo de paquetes donde contiene los modelos diseño de componentes del Framework

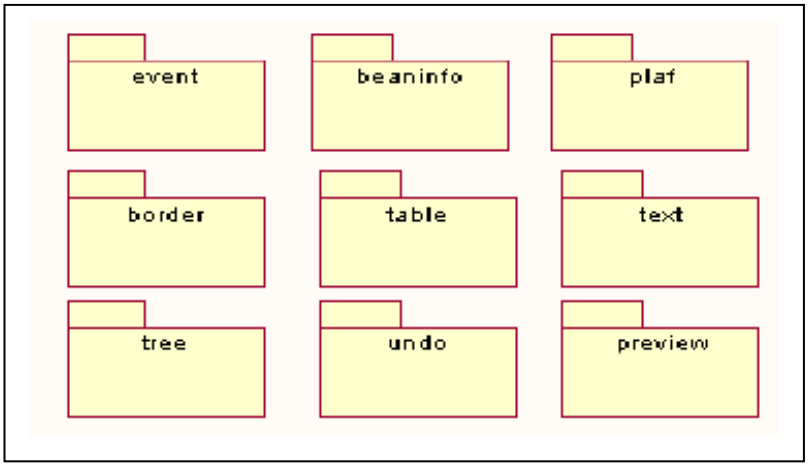


Fig. 5- Paquetes del Framework Swing

- Paquete **beanInfo:** Contiene clases que manejan los Bean.
- Paquete **event:** Contiene un conjunto de interfaces que permiten el control de los eventos sobre los componentes visuales.

- Paquete **plaf:** Contiene los paquetes que definen las clases con los estilos en que se muestran los componentes visuales.
- Paquete **border:** Contiene las clases que manejan los estilos de Bordes de los componentes.
- Paquete **table:** Contiene el conjunto de clases que permiten crear tablas, estilos de tablas y sus formas.
- Paquete **text:** Estos dos paquetes encierran las clases necesarias para trabajar los documentos en formato RTF o HTML.
- Paquete **tree:** Conjunto de clases e interfaces que permiten el tratamiento del componente árbol.
- Paquete **undo:** Conjunto de clases e interfaces que permiten la funcionalidad de revertir las operaciones.
- Paquete **preview:** Contiene un paquete que contiene las clases necesarias para el tratamiento de ficheros en los distintos tipos de Sistemas Operativos.

Ejemplo de algunas de las clases más importantes:

Todas las clases componentes de Swing (clases hijas de JComponent), son hijas de la clase Component de AWT.

ButtonGroup: Muestra una lista de elementos (JRadioButton) con sólo uno seleccionable. Cada elemento tiene un círculo, que en caso del elemento seleccionado contendrá un "punto".

JToggleButton: Es como un botón normal, pero al ser pinchado por el usuario queda activado.

JProgressBar: Representa una barra de estado de progreso, mediante la que habitualmente se muestra el desarrollo de un proceso en desarrollo (ejemplo: la instalación de una aplicación).

JTabbedPane: Es una ventana con solapas (la que utiliza Windows). Este componente había sido muy solicitado.

JApplet:: Aunque ya existía una clase Applet en AWT, esta nueva versión es necesaria para crear applets Java que utilicen interfaces Swing.

2.2.2.2 Framework Spring.

Spring es un framework que facilita la creación de diversas aplicaciones. Diseñado en módulos, con funcionalidades específicas y consistentes con otros módulos, facilita el desarrollo de nuevas funcionalidades y hace que la curva de aprendizaje sea favorable para el desarrollador, al ser fácil de asimilar. [CRAIG & RYAN, 2005]

El objetivo central de Spring es permitir que objetos de negocio y de acceso a datos sean reusables, no atados a servicios J2EE específicos. Estos objetos pueden ser reutilizados tanto en entornos J2EE (web o EJB), aplicaciones stand-alone y entornos de pruebas.

Filosofía de Spring Framework:

- Proveer un Framework no invasivo.
- Siempre que se pueda reusar código.
- Plug & Play de componentes.

Dentro de las ventajas que ofrece Spring, se encuentran que facilita la manipulación de los objetos, se usen EJBs o no, reduce la proliferación de Singletons (Patrón de Diseño Singleton), elimina la necesidad de usar distintos y variados tipos de ficheros de configuración, mejora la práctica de programación, permite el uso o no de EJBs, realizando el mismo tipo de funciones sin ellos.

Spring proporciona:

- Una potente configuración de la aplicación basada en JavaBeans, aplicando los principios de Inversión de Control (IoC). Esto hace que la configuración de aplicaciones sea rápida y sencilla. Ya no es necesario tener singletons ni ficheros de configuración, una aproximación consistente y elegante. Esta factoría de beans puede ser usada en cualquier entorno, desde el contexto de la aplicación.
- Una capa genérica de abstracción para la gestión de transacciones, permitiendo gestores de transacción enchufables (pluggables), y haciendo sencilla la demarcación de transacciones sin tratarlas a bajo nivel. Se incluyen estrategias genéricas y un único JDBC DataSource. En contraste con EJB (Enterprise Java Bean), el soporte de transacciones de Spring no está atado a entornos J2EE.
- Una capa de abstracción JDBC que ofrece una significativa jerarquía de excepciones (evitando la necesidad de obtener de SQLException los códigos que cada gestor de base de datos asigna a los errores), simplifica el manejo de errores, y reduce considerablemente la cantidad de código necesario.
- La funcionalidad AOP (Programación Orientada a Aspectos), está totalmente integrada en el framework Spring. Se puede aplicar AOP a cualquier objeto gestionado por Spring, añadiendo aspectos como gestión de transacciones declarativa.
- Un framework MVC (Model-View-Controller), construido sobre el núcleo de Spring. Este framework es altamente configurable vía interfaces. De cualquier manera una capa modelo realizada con Spring puede ser fácilmente utilizada con una capa de Presentación basada en MVC (Modelo Vista Controlador) como es el caso de Swing.

¿Por qué usar Spring?

Spring es un framework que tiene el objetivo de facilitar la construcción de aplicaciones java. A diferencia de otros frameworks como Struts, Spring se puede utilizar en cualquier tipo de aplicación, web o desktop como la que se propone. Spring es un framework ligero por el mínimo impacto que tiene en las aplicaciones.

Constituye una alternativa a estos, como es el caso del manejo de los objetos y la gestión de transacciones, disminuyendo el tiempo y el esfuerzo en el desarrollo de las aplicaciones. Spring tiene integrado el framework Acegi, que provee un mecanismo de seguridad potente y fácil de configurar permitiendo implementar las políticas de seguridad de manera transparente al código de la aplicación.

Está pensado para manejar los objetos del negocio de la aplicación con su técnica de inyección de instancias, todo, desde un fichero de configuración, manteniendo el código limpio y disminuyendo el acoplamiento. Contiene un módulo para programar orientado a aspectos que permite implementar funciones que de hacerlas orientado a objetos serían largas e intrusivas en el código de la aplicación. Posee una capa de abstracción para el acceso a datos con JDBC que facilita la implementación en esta capa. Cuando se utilizan otros frameworks de desarrollo tanto en la capa de presentación, en este caso Swing, como en la capa de acceso a datos, Spring funciona como un framework integrador que facilita la comunicación entres ellas.

2.2.2.3 Framework JUnit

Es un framework que permite realizar la ejecución de clases Java de manera controlada, para poder evaluar si el funcionamiento de cada uno de los métodos de la clase se comporta como se espera. Es decir, en función de algún valor de entrada se evalúa el valor de retorno esperado; si la clase cumple con la especificación, entonces JUnit devolverá que el método de la clase pasó exitosamente la prueba; en caso de que el valor esperado sea diferente al que regresó el método durante la ejecución, JUnit devolverá un fallo en el método correspondiente.

JUnit es también un medio de controlar las pruebas de regresión, necesarias cuando una parte del código ha sido modificado y se desea ver que el nuevo código cumple con los requerimientos anteriores y que no se ha alterado su funcionalidad después de la nueva modificación. El propio framework incluye formas de ver los resultados (runners) que pueden ser en modo texto, gráfico (AWT o Swing).

2.2.2.4 Framework Hibernate 3.2 [Bauer, 2005]

En los entornos de desarrollo actuales, trabajar con software orientado a objetos y bases de datos relacionales puede hacerle invertir mucho tiempo a los desarrolladores. Hibernate realiza el mapeo entre el mundo orientado a objetos de las aplicaciones y el mundo entidad-relacion de las bases de datos.

El centro de la arquitectura de Hibernate lo constituyen una serie de interfaces que realizan el grueso de las funcionalidades del framework, dentro de ellas están:

Session: Es la más usada en las aplicaciones, es la manejadora de la persistencia: a través de ella se podrá guardar y cargar objetos de la base de datos.

SessionFactory: Mediante ella se obtiene la Session.

Configuration: Es usada para configurar Hibernate y para especificar la ruta de los documentos de mapeo.

Transaction: Se utiliza para el control de las transacciones.

Query y Criteria Permiten la ejecución de consultas a la Base de Datos.

Además de estas interfaces Hibernate brinda otros aspectos muy importantes y que serán de gran utilidad en las aplicaciones, por ejemplo:

La interface **Type**, es un elemento fundamental y muy poderoso, permite el mapeo de tipos "java" a columnas en bases de datos incluso a varias columnas, incluye tipos como Calendar, byte[] y permite además; definir los propios tipo de datos (Persona, Dirección, Nombre).

Las interfaces **Callback** gestionan el ciclo de vida de los objetos (Lifecycle, Validatable). Un dialecto orientado a objetos (HQL) fácil de manejar y familiar a SQL que aunque no es un lenguaje de manipulación de datos como este, puesto que es usado solamente para extraer datos no para borrar, insertar o actualizar, permite realizar complejas consultas.

Hibernate puede ser configurado y ejecutado en cualquier aplicación java y entorno desarrollo.

Define dos tipos de configuración:

- Entorno manejado, entornos que proveen reservas de recursos como conexiones a base de datos (connections pool), transacciones y seguridad declarativa, en este caso se encuentran los servidores de aplicación como JBoss, BEA, WebLogic, entre otros.
- Entorno no manejado, es el caso de los contenedores de servlet como Tomcat, las aplicaciones de escritorio también son incluidas aquí; este tipo de entorno no provee transacciones automáticas ni manejos de recursos, la propia aplicación realiza el manejo de las conexiones a base de datos. En los entornos no manejados Hibernate se ocupa de las transacciones y las conexiones JDBC o lo deja para que el desarrollador se ocupe de esto, en el caso de los entornos manejados Hibernate se integra con el

contenedor para ocuparse de los DataSources y de las transacciones. Salvo estas diferencias lo demás es común para cualquier entorno.

En los entornos no manejados Hibernate se ocupa de las transacciones y las conexiones JDBC o lo deja para que el desarrollador se ocupe de esto, en el caso de los entornos manejados Hibernate se integra con el contenedor para ocuparse de los DataSources y de las transacciones. Salvo estas diferencias lo demás es común para cualquier entorno.

Otros aspectos que son esenciales en el desarrollo de aplicaciones con Hibernate lo constituyen los ficheros de mapeo y configuración. Para cada clase persistente se le hace corresponder un fichero de mapeo que es el lugar donde se le especifica al framework como va a persistir dicha clase en la base de datos pero además se trata todo lo referente a las relaciones de esta clase con otras incluyendo el tratamiento de herencia y polimorfismo. El archivo de configuración es el que le dice a Hibernate todo lo referente a la conexión que se puede alcanzar vía JNDI en el caso de los entornos manejados o cargando la conexión con el driver correspondiente al gestor haciendo uso si se quiere de alguna herramienta que maneje la reserva de conexiones todo esto en el mismo archivo.

Ventajas:

- Abstrae del manejo del código SQL, permitiendo ganar en tiempo de desarrollo.
- No necesita ningún entorno especial para correr sea servidor aplicaciones, contenedores.
- Las clases que persisten no tienen que implementar ninguna interfaz especial ni extender ninguna clase, por lo que son completamente trasparentes haciendo posible su reutilización en distintas partes de la aplicación con el objetivo de transportar los datos, y haciéndola desacoplada.
- Permite manejar las transacciones de una manera eficaz.
- Toda la configuración de la conexión y el mapeo se realiza en ficheros externos, haciendo transparente el uso de distintos gestores.
- Permite manejo de almacén de conexiones (Connections Pool).
- Es un framework open source y gratis.

Desventajas

- No es un estándar J2EE.
- Es un framework complejo. Puede costar un poco de esfuerzo dominar cada una de las propiedades que caracterizan al mismo.

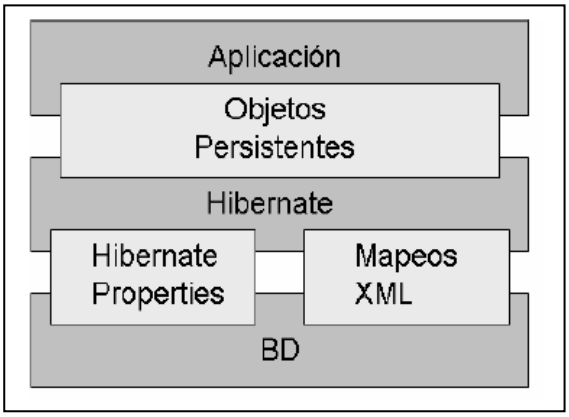


Fig. 6- Hibernate Framework

¿Por qué usar Hibernate?

En aplicaciones complejas el uso del API JDBC puede traer consigo interminables líneas de código SQL para realizar todo el trabajo de acceso a datos. Una de las alternativas a esta problemática son los EJB de entidad. Esta variante tiene como inconveniente el hecho de que son complejos de implementar, se ejecutan dentro de un contenedor EJB y por tanto en un servidor de aplicaciones. Es por eso que la propuesta se inclina al uso de un framework de persistencia como Hibernate.

Hibernate crea un puente entre el mundo orientado a objetos de las aplicaciones y en el entorno relacional de las base de datos., abstrayendo al desarrollador del código JDBC y de las consultas SQL. Es un framework ORM (Object Relational Mapping), que se basa en ficheros XML que mapean los objetos con tablas en la base de datos. Hibernate constituye un mecanismo persistente transparente pues las clases persistentes desconocen que tiene esa propiedad porque no tienen que implementar ninguna interfaz especial, ni extender de ninguna clase, y pueden ser utilizadas en cualquier capa de la aplicación.

Hibernate provee un lenguaje de consultas orientado a objetos que facilita la ejecución de estas; crea una capa de abstracción que permite migrar el gestor de base de datos sin cambiar una sola línea de código.

2.2.3 Organigrama de la arquitectura

Como se había visto en el capitulo anterior, 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. Los estilos arquitectónicos son el nivel de abstracción mayor para estructurar el sistema, la elección del mismo está dada por el tipo de aplicación que se vaya a desarrollar.

Teniendo en cuenta todo lo antes expuesto, a continuación se presenta la arquitectura del sistema a partir del estilo Arquitectura en Capas

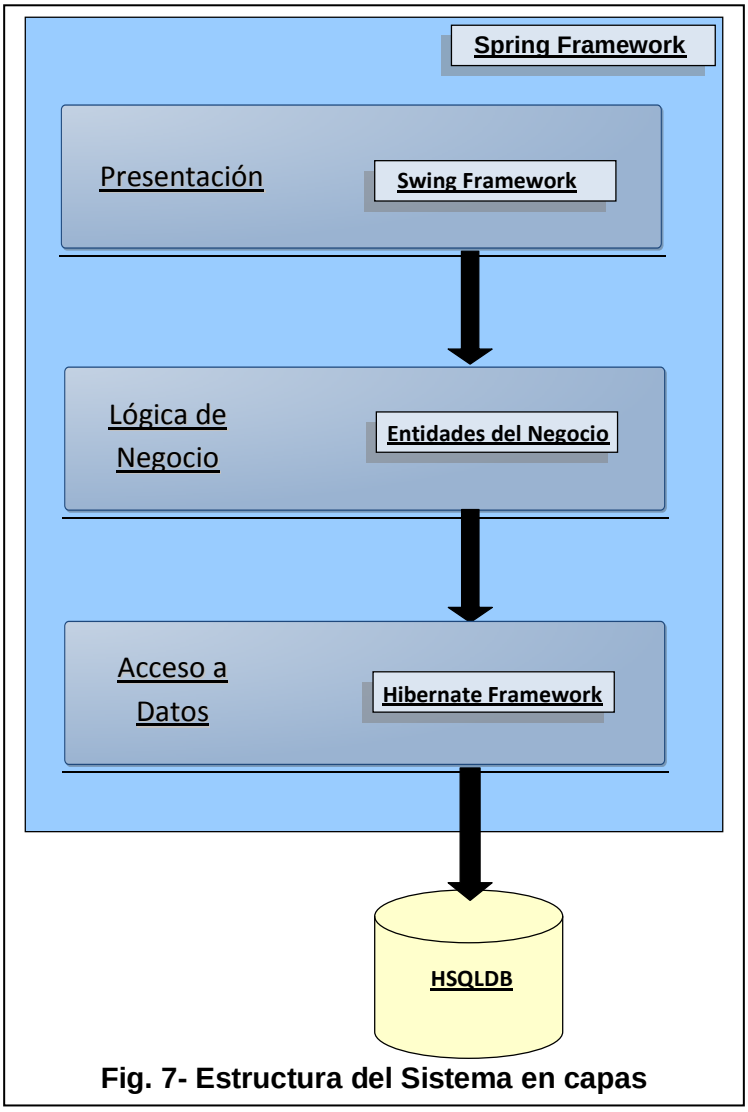


Fig. 7- Estructura del Sistema en capas

Los componentes SW en el modelo de 4 capas:

Capa de Presentación

Interfaz de usuario:

En esta capa se encuentra al Framework de Aplicación Swing en inglés (Swing Application Framework). Swing usa el Modelo Vista Controlador (MVC) como diseño fundamental detrás de cada uno de sus componentes. Esencialmente MVC divide los componentes de la interfaz gráfica en tres elementos. Cada uno de esos elementos juega un papel fundamental dentro del comportamiento de cada componente.

Modelo: El modelo abarca el estado de los datos en cada componente. Existen diferentes modelos para distintos tipos de componentes. Es importante resaltar que el modelo de datos siempre existe independientemente de la representación visual de los componentes.

Vista: La vista se refiere a cómo se ven los componentes en la pantalla. Las vistas no son responsables de modificar los datos o incluso de obtener los datos; estas simplemente sirven para mostrar los datos del modelo que han sido suministrados por un controlador.

Controlador: El controlador dicta cómo los componentes interactúan con los eventos. El controlador decide cómo cada componente reaccionará ante cada evento

Lógica del Negocio

En esta capa se agrupan todas las clases y procesos del negocio. Cada módulo identificado define su propia lógica de negocio para dar cumplimiento a los requisitos funcionales. Con la ayuda de Spring Framework se logra manipular todas las interfaces de objetos del negocio, sabiendo en un momento determinado cual de las implementaciones de esas interfaces debe utilizarse.

Acceso a Datos

El término de Objeto de Acceso a Datos o en inglés, Data Access Object (DAO) (DEEPAK ALUR 2003), es ampliamente usado en el desarrollo de software. Los DAOs encapsulan la persistencia de los objetos del negocio, proveen la persistencia de los objetos transitorios y las actualizaciones de los objetos existentes en la BD.

Las implementaciones de los DAOs estarán disponibles para los objetos (típicamente para los objetos de negocio) haciendo uso de la inyección de dependencias con los objetos de negocio y las instancias de los DAOs, configurada en el contenedor de inversión de control de Spring Framework.

En esta capa entra a jugar un papel fundamental el framework Hibernate con su mapeo en XML de todos los DAOs para lograr hacerlos persistir en la base de datos.

Conectores / Configuraciones

Los conectores son las formas de comunicación entre los distintos componentes o elementos definidos en el estilo arquitectónico, no es más que la forma en que está representada la información que fluye entre estos. En la aplicación la comunicación entre capas fluye a través de fachadas; cada capa proporciona servicios a la capa inmediatamente superior y se sirve de las prestaciones que le brinda la inmediatamente inferior y las interacciones entre estas ocurren generalmente por invocación de métodos.

Cada modulo tiene en su estructura interna la distribución de las capas lógicas que se definen de forma general para la arquitectura del sistema. Por eso a la hora de comunicarse de una capa a otra se utilizan las fachadas (clases interfaces) definidas para tal función.

Por ejemplo en la capa de presentación se hace una petición de mostrar una tabla determinada, persistente en el sistema de ficheros. La interfaz definida en la capa de presentación hace la petición a la fachada del negocio. Esta a su vez, le pide a la capa de acceso a datos la fachada del DAO específico que se encargará de, a través de Hibernate, obtener el objeto solicitado de la base de datos Entonces esté regresará por las mismas vías que se hizo la petición hasta llegar a la capa de presentación y ser mostrado al usuario.

Restricciones

La principal restricción que se le impone a los componentes de este estilo arquitectónico es que los componentes que pertenecen a las capas superiores sólo pueden hacer uso de los componentes de sus capas inmediatas inferiores. Para poder hacer uso de un modelo es necesario hacer por lo menos una primera petición a la capa inmediata inferior y así poder usar los componentes de menor nivel. La posibilidad de que un sistema requiera consideraciones de rendimiento puede requerir acoplamientos entre capas de alto y bajo nivel, no admitiendo un buen mapeo de la estructura jerárquica. En ocasiones es muy difícil encontrar el nivel de abstracción concreto, cuando por ejemplo muchos protocolos de comunicación agrupan diversas capas, trayendo consigo que proliferen los drivers y servicios monolíticos.

Ventajas

La característica del estilo N-Capas: cada capa facilita servicios a la capa inmediata superior; facilita la modularidad del sistema; mejora considerablemente el soporte del mismo y la localización de errores. Hacer uso de este estilo permite una mayor flexibilidad, se pueden añadir nuevos módulos para añadir funcionalidad, además, soporta un diseño basado en niveles de abstracción crecientes, ventaja que pueden aprovechar los implementadores para dividir un problema complejo en una secuencia de pasos incrementales. Además existe la posibilidad de definir interfaces de capa estándar, a partir de la cual se pueden construir extensiones o prestaciones específicas, reutilizando las estructuras definidas y creciendo en funcionalidad el sistema.

2.2.4 Convenciones o estándares de código y recursos.

Con el fin de lograr un lenguaje común y uniforme en toda la aplicación se especifican convenciones de nombres y estándares de código para los distintos recursos las aplicaciones.

Se definen convenciones de nombres para las distintas clases java dependiendo de las funciones que tengan cada una de estas en la aplicación:

- Clases de la capa de Acceso a Datos:
  - Las interfaces que representan las operaciones sobre los objetos de acceso a datos, correspondientes al patrón de diseño Data Access Object terminan con la palabra "DAO". Ejemplo: estratificacionDAO.
  - Las implementaciones reales de las interfaces DAO comienza con el nombre de la interfaz correspondiente y terminan con la palabra "Impl". Ejemplo: estratificacionDAOImpl.
  - Las implementaciones falsas de las interfaces DAO comienzan con el nombre de la interfaz correspondiente y terminan con la palabra "Mock". Ejemplo: estratificacionDAOMock.
  - Las clases utilizadas para realizar pruebas a los interfaces DAO comienzan con el nombre de la interfaz correspondiente y terminan con la palabra "Test". estratificacionDAOTest.
- Clases de la capa de Negocio:
  - Las interfaces que representan las fachadas que a agrupan las funcionalidades del negocio de los módulos (se basa en el patrón de diseño Facade) terminarán con la palabra "Facade". Ejemplo: ecuacionFacade.
  - Las implementaciones de las fachadas comenzarán con el nombre de la interfaz correspondiente y terminaran con la palabra "Impl". Ejemplo: ecuacionFacadeImpl.
  - Las interfaces que representan un conjunto de funcionalidades de la lógica de negocio de un módulo terminarán en "Manager". Ejemplo: estratificacionManager.
  - Las implementaciones que se le realicen la interfaz de un Manager deben terminar con el nombre de la interface más "Impl". Ejemplo: estratificacionManagerImpl.
  - Las clases de prueba de los Manager terminarán con el nombre de la interfaz a la que prueban, seguido de "Test". Ejemplo estratificacionManagerTest.
- Clases de la capa de Presentación:
  - ✓ Capa Swing:

- Las clases que manejan el flujo de la capa de presentación, es decir, los controladores, terminarán con la palabra “Controller”.
- Las clases utilizadas para hacer pruebas de unidad a los Controladores comenzarán con el nombre del controlador correspondiente y terminará con la palabra “Test”.

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

Los AppContext tendrán la siguiente estructura:

[soga]-[módulo]-[capa lógica]-context-[tipo].xml

- [módulo]: indican las unidades organizacionales utilizadas en el proyecto de forma de árbol, pueden ser tantas como la complejidad del mismo lo requiera según las estructuras definidas para cada nivel de complejidad.
- [capa lógica]: representa la capa lógica que maneja del Application-Context de acuerdo al tipo de beans que se manejan en él. Se establecen las siguientes clasificaciones explicada a continuación:
  - [accesoDatos]: Contiene los beans con las funcionales referentes a la capa de acceso a datos.
  - [presentacion]: Encapsula todo los beans de la capa de presentación. Utilizando Spring MVC aquí se configuran componentes como Controladores, Handler Mappings, View Resolvers y todos los utilizados para conformar la lógica de esta capa
  - [negocio]: Se refiere a las operaciones de negocio de la aplicación, en este XML se definen las fachadas y otros objetos que representen la capa de servicios.
- [tipo]: Se refiere al tipo de función que cumplirá este AppContext en la aplicación. Ejemplo: cuando se utiliza para configurar los beans falsos sería “mock”, en caso de ser un AppContext de negocio sería “negocio”.

A la hora nombrar los casos de uso identificados en cada módulo se debe tener en cuenta la siguiente nomenclatura:

CU_Siglas del modulo_ Nombre del caso de uso.

Donde, las siglas de cada módulo serán las siguientes:

Modulo de Muestreo: MM

Modulo de Importación / Exportación: MIE

Modulo de Análisis: MA

Modulo Editor de Ecuaciones: MEQ

Modulo de SOGAC: MSOGAC.

2.2.5 Patrones de Diseño.

Los patrones de diseño son la base para la búsqueda de soluciones a problemas comunes en el desarrollo de software y otros ámbitos referentes al diseño de interacción o interfaces.

Un patrón de diseño es una solución a un problema de diseño. Para que una solución sea considerada un patrón debe poseer ciertas características. Una de ellas es que debe haber comprobado su efectividad resolviendo problemas similares en ocasiones anteriores. Otra es que debe ser reusable, lo que significa que es aplicable a diferentes problemas de diseño en distintas circunstancias.

2.2.5.1 Patrones de Creación

Factory Method [GoF95]

Un ejemplo de utilización es para crear diferentes productos, entonces existen diferentes tipos de productos que pueden heredar de una Interface con operaciones comunes o una clase abstracta si además los productos tienen lógica en común. Existe la clase Factory que implementa la clase IFactory con un método CrearProducto (discriminador), donde el discriminador es uno o varios parámetros que identifican a un producto específico, este método es llamado por las clases que necesiten los productos.

Uso en la aplicación: Para crear los diferentes tipos de vistas que serán usadas más tarde como prototipos.

Abstract Factory [GoF95]

Proporciona una interfaz para crear **familias** de objetos relacionados o dependientes sin especificar su clase concreta.

Cuando usarlo:

- Cuando el sistema debe ser independiente de como sus productos se crean, componen y representan.
- Cuando el sistema debe configurarse con una familia de productos de entre múltiples posibles.

- Cuando se quiere dar énfasis a la restricción de que una familia de productos ha sido diseñada para que actúen todos juntos.
- Cuando se quiere proporcionar una librería de clases de productos, de los que sólo se desea revelar su interfaz, no su implementación.

Por ejemplo si existen diferentes tipos de componentes visuales (ventanas, botones,) y diferentes formas gráficas de representarlas (Gnome, X-Windows), la solución sería crear interfaces para ventanas y botones, las cuales implementarían las versiones Gnome y X-Windows de esos componentes. También se crearía una Interface IFabricaElementos con métodos para crear ventanas y botones, esta interface sería implementada por las fábricas de cada sistema gráfico diferente para construir ventanas y botones, y serían llamadas por el cliente.

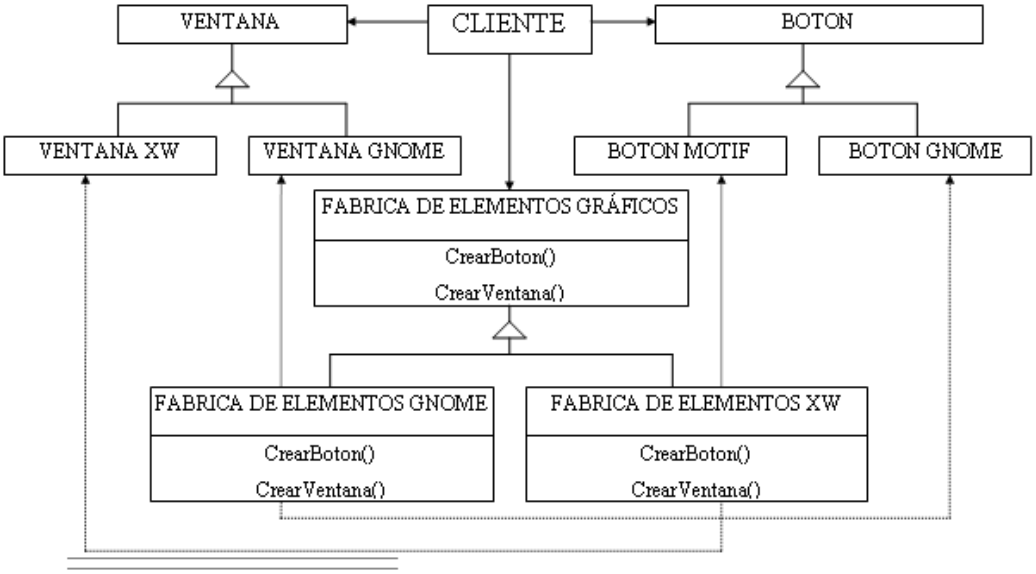


Fig. 8- Funcionamiento del patrón Abstract Factory

Builder [GoF95]

Permite a un objeto cliente construir un objeto complejo especificando solamente su tipo y contenido. El cliente es privado de los detalles de construcción del objeto. Separa la construcción de un objeto complejo de su representación, para que el mismo proceso de construcción permita crear varias representaciones.

Es usado para permitir la creación de una variedad de objetos complejos desde un objeto fuente (Producto), el objeto fuente se compone de una variedad de partes que contribuyen individualmente a la creación de cada objeto complejo.

Uso en la aplicación: Para construir los objetos a exportar en los diferentes tipos de exportación a partir de la información guardada en la BD.

Prototype [GoF95]

Con el prototype se definen valores por defecto para la utilización por primera vez del objeto, se accede a él por medio de la clonación del objeto prototipo hacia un nuevo objeto que podrá ser modificado.

Uso en la aplicación: Se podrían crear prototipos de vistas del software. Una vista es la disposición y visibilidad en la pantalla de diferentes ventanas que en conjunto facilitan la ejecución de un proceso.

Singleton [GoF95]

Garantiza que solamente se crea una instancia de la clase y provee un punto de acceso global a él. Todos los objetos que utilizan una instancia de esa clase usan la misma instancia.

2.2.5.2 Patrones Estructurales

Adapter [GoF95]

Una clase Adapter implementa un interfaz que conoce a sus clientes y proporciona acceso a una instancia de una clase que no conoce a sus clientes, es decir convierte la interfaz de una clase en una interfaz que el cliente espera. Un objeto Adapter proporciona la funcionalidad prometida por un interfaz sin tener que conocer que clase es utilizada para implementar ese interfaz. Permite trabajar juntas a dos clases con interfaces incompatibles.

Uso en la aplicación: Se usa en la interfaz de usuario, en java se manejan los eventos de esta forma.

Iterator [GoF95]

Este patrón se utiliza para realizar los recorridos, de forma ascendente y descendente.

Uso en la aplicación: Se usa a la hora de mostrar en las tablas rangos de valores, accediendo a estos ascendente y descendentemente.

Bridge [GoF95]

Es útil cuando hay una jerarquía de abstracciones y la correspondiente jerarquía de implementaciones. Más que combinar las abstracciones e implementaciones en muchas clases distintas, el patrón Bridge implementa las abstracciones e implementaciones como clases independientes que se pueden combinar dinámicamente. Es decir, desacopla una abstracción de su implementación y les permite variar independientemente.

Ejemplo: La implementación de Drivers, como JDBC de java, este se abstrae en la realización de consultas SQL sin preocuparse de la implementación específica del driver que se conectará a la base de datos deseada.

Uso en la aplicación: Se usa en el módulo de Importación/Exportación.

Facade [GoF95]

Fachada: Simplifica los accesos a un conjunto de objetos relacionados proporcionando un objeto que todos los objetos de fuera del conjunto utilizan para comunicarse con el conjunto. Define una interface de más alto nivel que permite usar el sistema más fácil.

Uso en la aplicación: Este patrón tiene un gran uso dentro del sistema. La comunicación entre los diferentes módulos de la aplicación es a través de fachadas.

Observer [GoF95]

Permite captar dinámicamente las dependencias entre objetos, de tal forma que un objeto notificará a los objetos dependientes de él cuando cambia su estado, siendo actualizados automáticamente.

Uso en la aplicación: Se puede usar en el monitoreo de la conservación de la integridad de los datos.

Strategy [GoF95]

Se utiliza para encapsular algoritmos relacionados en clases que son subclases de una superclase común. Esto permite la selección y utilización de un algoritmo que varia según el objeto utilizado.

Uso en la aplicación: Se usa en el módulo de Muestreo, al definir un método genérico para los muestreos, que va a ser implementado en los diferentes tipos de muestreos.

2.3 Descripción de la Arquitectura de Software propuesta.

El binomio RUP/UML será la metodología y lenguaje usados para el proceso de desarrollo del Software General de Auditoría Cubano. RUP es una metodología que recoge lo mejor de cada una de las analizadas y traza una mejor y completa línea de trabajo, es un proceso de desarrollo de Software que proporciona una guía en el orden de las actividades de un equipo, dirige las tareas individuales de los desarrolladores, especifica que productos deberían ser desarrollados y ofrece criterios para monitorear y medir los productos y actividades de un proyecto así como usar casos de uso en forma efectiva, facilita tener una interacción continua y clara con el cliente.

RUP es una metodología que se caracteriza por desarrollar un proceso iterativo e incremental, guiado por los casos de uso y centrado en la arquitectura.

Además, tiene a UML como lenguaje de representación visual, el cual logra representar la arquitectura conformada por diferentes visiones del sistema, constituye un modelo de cómo está estructurado dicho sistema, sirviendo de comunicación entre las personas involucradas en el desarrollo y ayudando a realizar diversos análisis que orienten el proceso de toma de decisiones.

2.3.1 Definición de Arquitectura en RUP

La arquitectura sigue el framework "4+1" vista, este framework define cuatro vistas para la arquitectura en conjunto con los escenarios, y es presentado en la siguiente figura.

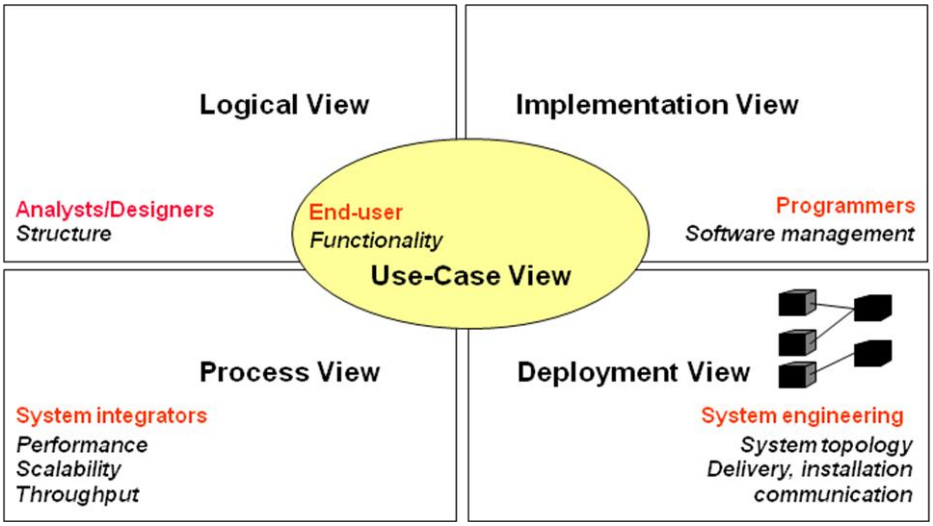


Fig. 9- Framework 4+1 vista.

Framework 4+1 vistas	Arquitectura
Use – Case View	Vista de Caso de Uso, Restricciones, Calidad
Logical View	Vista Lógica
Process View	Vista de Procesos
Deployment View	Vista de Despliegue
Implementation View	Vista de Implementación

2.3.3 Metas y restricciones arquitectónicas

Las metas y restricciones del sistema en cuanto a seguridad, portabilidad y reusabilidad, significativas para la arquitectura son:

2.3.3.1 Requerimientos de Hardware

Estaciones de Trabajo

1- Tener periféricos, Mouse Teclado

2- Tarjeta de Red.

3- 512 MB de Memoria RAM

2.3.3.2 Requerimientos de Software

Estaciones de Trabajo

1- Sistema Operativo: Windows 98 o superior, Linux, Unix.

2- Java Virtual Machine (Máquina Virtual de Java) versión 1.6 o superior.

2.3.3.3 Usabilidad.

- El sistema podrá ser usado por cualquier persona que posea conocimientos básicos en el manejo de la computadora y de auditoría, contabilidad o sean profesionales en sistemas y finanzas.
- El software tendrá siempre la posibilidad de ayuda disponible, lo que le permitirá un avance considerable en la explotación de la aplicación en todas sus funcionalidades.

2.3.3.4 Rendimiento.

- Tiempos de respuestas y velocidad de procesamiento de la información rápidos, no mayor a los 5 segundos por 50 000 registros en las importaciones de ficheros, en las exportaciones de archivos y la generación de reportes.

2.3.3.5 Soporte.

- Se requiere la instalación de la máquina virtual de Java.

2.3.3.6 Portabilidad.

- El sistema puede funcionar en cualquier plataforma Windows, Linux o Unix.

2.3.3.7 Seguridad

- Protección contra acciones no autorizadas o que puedan afectar la integridad de los datos.
- Verificación sobre acciones irreversibles (eliminaciones).
- Una vez importados los datos, los accesos a los mismos serán de solo lectura.

2.3.3.8 Confiabilidad

- La herramienta de implementación a utilizar tiene soporte para recuperación ante fallos y errores.

2.3.3.9 Implantación

- Entregar toda la documentación asociada al proyecto.
- Organizar el adiestramiento de los usuarios.

2.3.3.10 Estructura del equipo de Desarrollo.

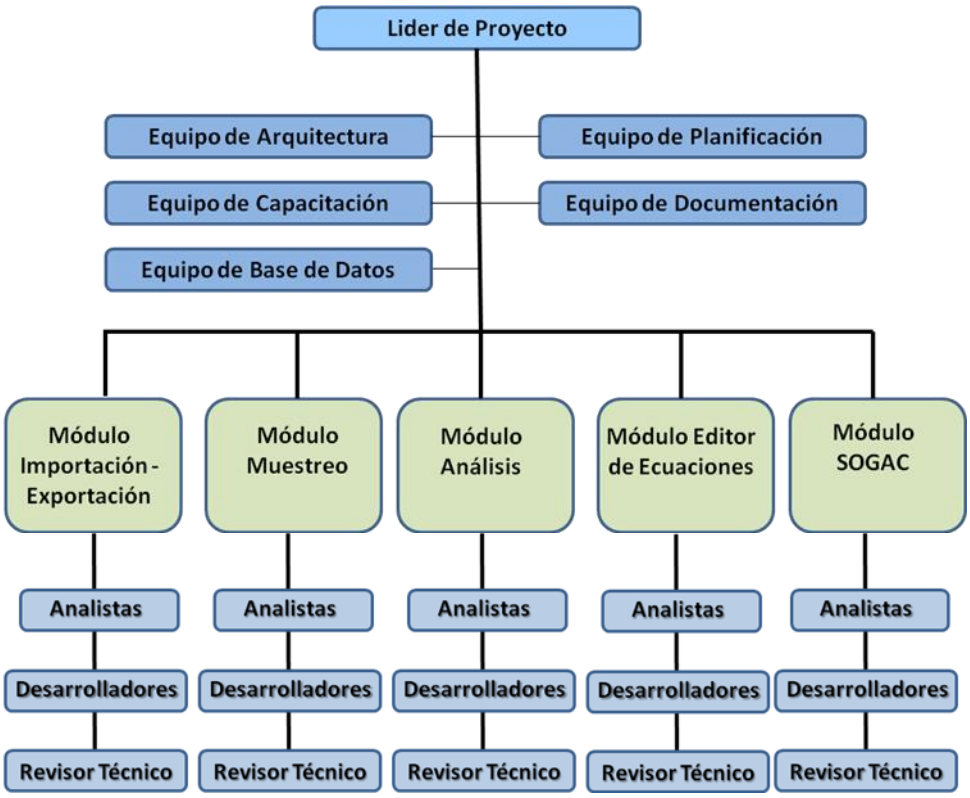


Fig. 10- Estructura del Equipo de Desarrollo

El equipo de desarrollo está dividido en los principales módulo identificados para el desarrollo del sistema, compuesto por una generalización de los roles propuestos por la metodología de desarrollo Rational Unified Process (RUP): Analistas, Desarrolladores y revisor técnico.

Configuración de los puestos de trabajo por Roles:

Rol Analista:

- PC con mouse y teclado.
- Instalación Suite de Rational 2003, para la modelación del sistema
- Paquete Office para la documentación del sistema.

Rol Desarrollador

- PC con mouse, teclado, 1 GB de memoria RAM
- Instalación de la Máquina Virtual de Java (JDK 1.6)
- NetBeans 6.0.

Rol Revisor Técnico:

- PC con mouse y teclado

- Instalación Suite de Rational 2003, para modelar los Casos de Prueba.
- Paquete Office para la documentación de las pruebas del sistema.

2.3.4 Vista de Casos de Uso del Sistema por módulos.

Módulo de Importación / Exportación de Ficheros:

Este módulo es uno de los más importantes del software, pues en él se importan los datos desde diferentes fuentes y formatos, a la BD que tendrá la plataforma donde se realizarán las pruebas sustantivas y de cumplimiento por parte del auditor o personal especializado.

Los casos de uso significativos para la arquitectura dentro de este módulo se enumeran y explican a continuación:

1- CU_MIE_Importar_Datos_Formato_Predefinido: Este caso de uso se inicia cuando el auditor desea importar a la plataforma del software un archivo que está en uno de los siguientes formatos predefinidos de archivos, conocidos por la aplicación: *.xls, *.dbf, *.mdb, *.pdf, *.xml y el ODBC. El caso de uso termina cuando se importa correctamente el archivo y se visualiza en pantalla el fichero importado.

2- CU_MIE_Importar_Datos_No_Formateados: Este caso de uso comienza cuando el auditor desea importar datos que están en código ASCII o EBCDIC sin formato, y que hay que formatear, para lograr la correcta importación a la plataforma del sistema. El caso de uso concluye cuando se importa correctamente el archivo y se visualiza en pantalla el fichero importado.

3- CU_MEQ_Gestionar_Ecuacion: El caso de uso comienza cuando en uno de los casos de uso descritos anteriormente el auditor necesita de un criterio específico para hacer la importación. El criterio lo conforman una serie de restricciones que se pueden aplicar sobre el total de los datos posibles a importar para finalmente quedarse con los datos deseados. Todas estas posibles restricciones tienen un orden y formato que queda plasmado en una ecuación. El caso de uso concluye cuando queda editada y validada la ecuación para su puesta en práctica posteriormente sobre el volumen de datos a importar.

4- CU_MIE_Exportar_Datos_Formatos_Conocidos: El caso de uso se inicia cuando el auditor desea exportar la pruebas realizadas sobre los datos a un formatos de archivos conocidos, como por ejemplo: *.pdf, *.xls y *.doc. El caso de uso termina cuando se realiza correctamente la exportación y se guarda en un lugar determinado por el auditor, el archivo exportado.

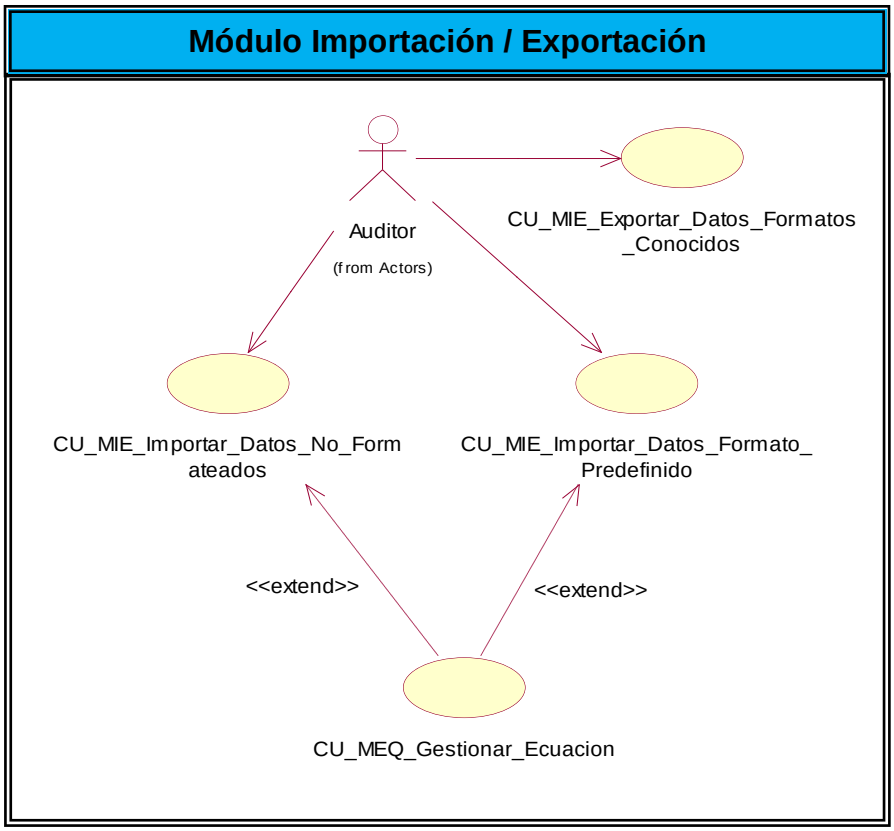


Fig. 11- Vista Casos de Uso Modulo Importación/Exportación

Módulo de Muestreo:

En este módulo se encuentran agrupados varios tipos de muestreos que son posibles aplicarles a los datos para saber si los mismos han sido alterados.

1- CU_MM_Muestreo_Sistematico: El caso de uso comienza cuando al auditor selecciona la opción realizar un muestreo sistemático, el auditor debe determinar que tipo de muestreo sistemático necesita hacer, si es según el número de registro entonces deberá entrar datos como el número de registros a seleccionar, si es según el intervalo de selección deberá seleccionar el intervalo de selección y en ambos casos, definirá el numero de registro inicial y final lo cuál determina en qué intervalo se realizará el muestreo. Además se podrá escoger los campos que se quieren ver en el resultado del muestreo. El caso de uso concluye cuando se obtiene como resultado una tabla con los campos escogidos y el número de registros según el muestreo sistemático.

2- CU_MM_Muestreo_Aleatorio: El caso de uso comienza cuando al auditor selecciona la opción de realizar un muestreo aleatorio, entrando al sistema datos como el número de

registro a seleccionar, el número de registro inicial desde donde se quiere empezar con el muestreo así como el final, además de poder escoger los campos que se quiere ver en el resultado del muestreo. El caso de uso concluye cuando se obtiene como resultado una tabla con los campos escogidos y el número de registros seleccionados.

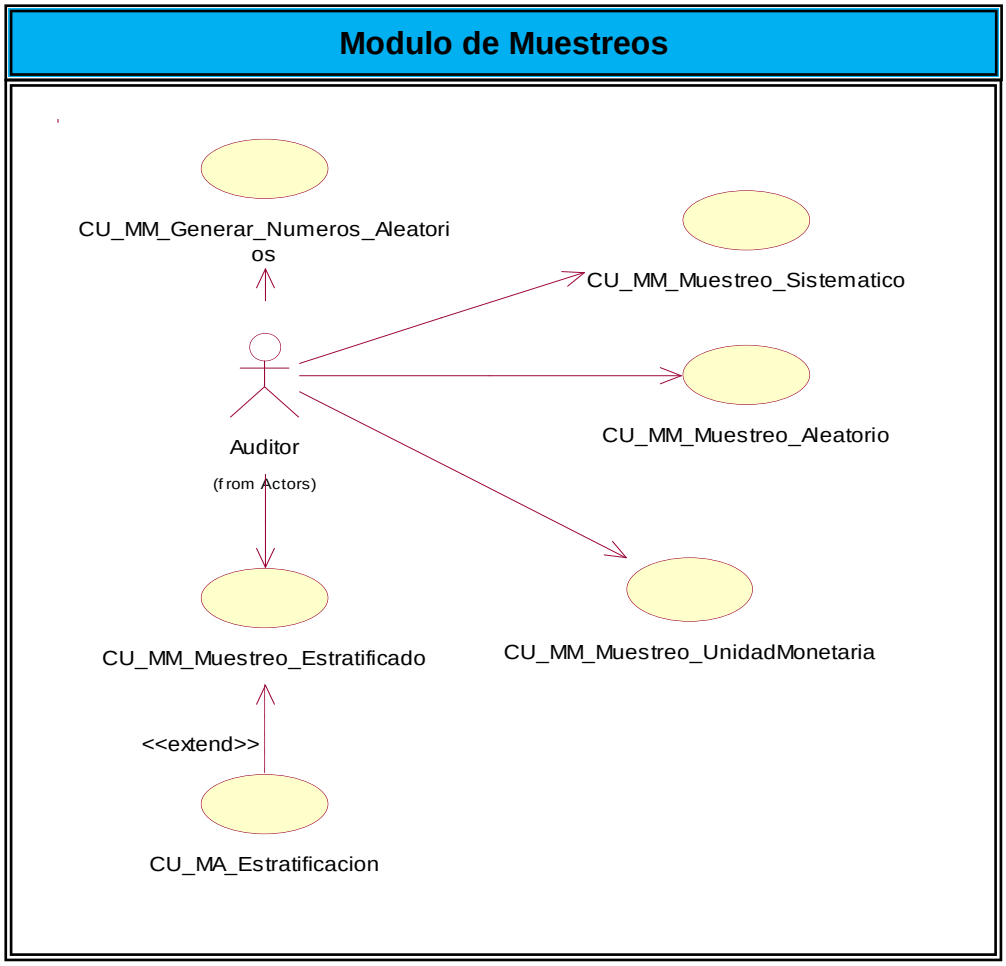
3- CU_MM_Muestreo_UnidadMonetaria: El caso de uso comienza cuando al auditor selecciona la opción realizar el muestreo de unidad monetaria a la tabla seleccionada, especificando si el tipo de extracción que se va a realizar es por intervalo fijo o por selección de celdas, además debe especificar otros datos, como son, el campo numérico al que se le va a hacer la muestra, el intervalo de selección, el punto de inicio aleatorio o semilla de números aleatorios y los valores positivos, negativos y absolutos con su total, y el número de registro que cada cual posee. El caso de uso termina cuando se muestra en pantalla una tabla con los resultados arrojados después de aplicar el muestreo.

4- CU_MM_Muestreo_Estratificado: El caso de uso comienza cuando al auditor selecciona la opción realizar un muestreo estratificado. Si la tabla activa no ha sido estratificada, entonces pasará al CU_MA_Estratificación. Una vez estratificada, el auditor debe especificar si en el muestreo él determinará la cantidad de registros a seleccionar en cada estrato, si se va a hacer de forma proporcional u óptimo, en cada uno de los casos debe entrar diferentes datos para poder realizarse el muestreo. El caso de uso termina cuando se obtiene como resultado una tabla con los registros seleccionados por el muestreo.

5- CU_MM_Generar_Numeros_Aleatorios: El caso de uso comienza cuando el auditor selecciona la opción, Generar Números aleatorios del menú Muestro. Teniendo que entrar datos tales como:

- cantidad de números aleatorios a generar,
- el rango en que se van a encontrar estos números, y
- la semilla aleatoria a utilizar.

El caso de uso concluye cuando se obtiene y muestra una tabla con todos los números aleatorios generados por el sistema.



Módulo del Editor de Ecuaciones

En este módulo se crean, modifican y/o guardan ecuaciones que pueden estar compuestas por un amplio conjunto de funciones definidas que se pueden aplicar sobre los datos de una tabla. Dicho módulo es un intérprete, este no es más que un programa que toma el código fuente (la ecuación), lo analiza y lo ejecuta directamente. Es uno de los módulos más importantes dentro del sistema, por su alta integración con los demás.

1- CU_MEQ_Gestionar_Ecuación: Este caso de uso puede ser iniciado por el auditor desde diferentes puntos del sistema en los módulos SOGAC, de Análisis e Importación / Exportación. En todos los casos el objetivo es el mismo; confeccionar o modificar una ecuación que servirá para especificar restricciones o criterio a aplicar en posibles acciones a realizar sobre la tabla activa. Esta ecuación puede ser guardada para ser cargada después en otra ocasión, y así terminaría el caso de uso. Otro posible fin del caso de uso sería cuando se comprueba que la expresión es válida y se regresa al

punto del sistema desde el cual fue llamado esté caso de uso, al mismo tiempo que devuelve los datos que cumplen con las restricciones especificadas en la ecuación.

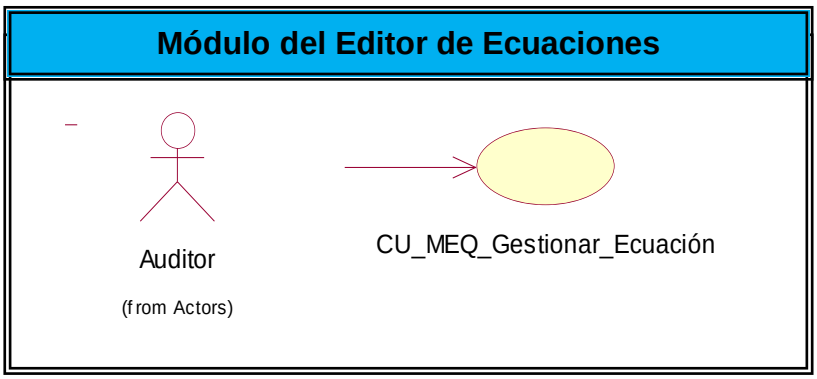


Fig. 13- Vista Casos de Uso Modulo Editor de Ecuaciones

Módulo de Análisis

Este módulo agrupa un conjunto de funcionalidades a aplicar por el auditor sobre los datos. Cada vez que se aplique una de estas funcionalidades a la tabla activa, lo que se obtenga será guardado como un Resultado para esa tabla. Los casos de usos significativos para la arquitectura de este módulo se describen a continuación.

1- CU_MA_Estratificación: El caso de uso se puede iniciar desde dos escenarios diferentes. El primero, es cuando el auditor da click en la opción Estratificación del menú Análisis. Entonces el auditor debe especificar:

-que campo de la tabla activa desea estratificar,

-cuales de los campos numéricos presentes en la tabla activa desea totalizar e incluir en la estratificación.

-el límite inferior y superior para cada estrato.

El auditor tiene la posibilidad de crear un criterio en el editor de ecuaciones el cual deberán cumplir los datos durante la estratificación. En este punto se invocaría el caso de uso CU_MEQ_Gestionar_Ecuación. El caso de uso termina cuando se obtiene una vista de la tabla teniendo en cuenta las restricciones impuestas en la estratificación.

El segundo escenario tiene lugar cuando en el Módulo de Muestreo el auditor da click en la opción Muestreo Aleatorio Estratificado. En ese momento se suceden todos los pasos descritos en el primer escenario que se explicó para este caso de uso.

2- CU_MA_Sumarización: El caso de uso se inicia cuando el auditor da click en la Opción Sumarización del menú Análisis y se abre una ventana en la cual debe especificar los siguientes datos:

- Campo(s) a Sumarizar.
- Cuales de los campos numéricos presentes en la tabla activa desea totalizar e incluir en los resultados de la sumarización.

  Si el auditor marcó la opción de totalizar algún campo numérico, entonces tendrá disponible un conjunto de Estadísticas más. Entre ellas se encuentran: Máximo y Mínimo, Promedio y Varianza. El caso de uso concluye cuando se muestra una tabla con los resultados obtenidos después de cumplir con las condiciones especificadas en la sumarización.

3- CU_MA_Clave_Dup_Detección: El caso de uso se inicia cuando el auditor desea detectar si existe alguna clave determinada que esté duplicada. En la ventana debe especificar lo siguiente:

- El campo que desea analizar.
- Si desea mostrar la Salida de los registros Duplicados o la Salida de los registros sin Duplicar.

  El auditor tiene la posibilidad de crear un criterio en el editor de ecuaciones el cual deberán cumplir los datos durante la Detección de clave duplicada. En este punto se invocaría el caso de uso CU_MEQ_Gestionar_Ecuación. El caso de uso concluye cuando se muestra una tabla con los resultados obtenidos después de cumplir con las condiciones especificadas en la sumarización.

4- CU_MA_Ley_Benford: El caso de uso comienza cuando el auditor necesita realizar un análisis a un campo de una tabla determinada a través de la Ley de Benford especificando los siguientes datos:

  - a qué campo de la tabla se le realizará el análisis,
  - si se desea incluir los valores positivos o negativos,
  - a los dígitos a los que se le aplicará el análisis sean estos el primer dígito, los dos primeros, los tres primeros o al segundo dígito,
  - además se especificará el nombre con los que se guardará cada resultado.

5- CU_MA_Detección_Omisiones: El caso de uso se inicia cuando el auditor da click en el menú Análisis y selecciona la opción Detección de Omisiones. En la venta que aparece el auditor deberá escoger el campo sobre el cual desea realizar la detección. Si el campo seleccionado es numérico, deberá especificar, además, si desea realizar la detección sobre todos los registros de la tabla o sobre un rango determinado que debe seleccionar, así como el valor del incremento de omisión.

El auditor tiene la posibilidad de crear un criterio en el editor de ecuaciones el cual deberán cumplir los datos durante la detección de omisiones. En este punto se invocaría el caso de uso CU_MEQ_Gestionar_Ecuación. El caso de uso concluye cuando se muestra una tabla con los resultados del proceso de detección.

6- CU_MA_Extraccion_Directa: El caso de uso comienza cuando el auditor da click en la opción realizar Extracción Directa del menú Analisis. En la ventana que aparece el auditor debe especificar los siguientes datos:

- Registros a extraer. Tiene dos opciones, la primera es extraer todos los registros; y la segunda es extraer un rango de registros que debe definir.

El auditor puede definir nuevos campos que tendrá la tabla resultante y/o seleccionar cuales de los campos actuales de la tabla desea que se muestren en el resultado. El auditor tiene la posibilidad de crear un criterio en el editor de ecuaciones el cual deberán cumplir los datos durante la extracción directa. En este punto se invocaría el caso de uso CU_MEQ_Gestionar_Ecuación.

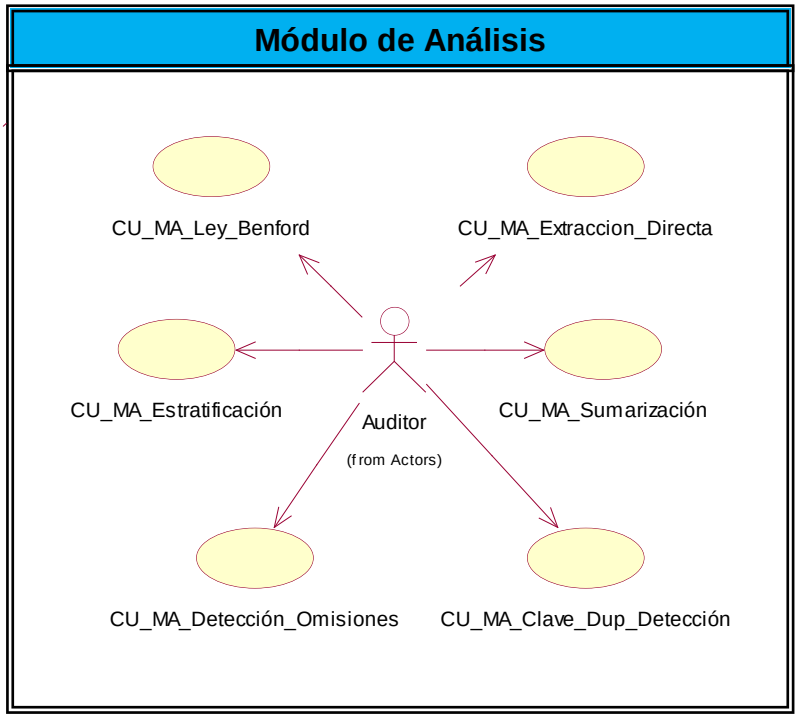


Fig. 14- Vista Casos de Uso Modulo de Análisis

Módulo SOGAC

Este módulo tiene como función principal la integración del resto de los módulos, además de un conjunto de funciones básicas para el intercambio de información entre la aplicación y el cliente. Su estructura está definida y basada en interfaces.

Los casos de uso significativos para la arquitectura son:

1- CU_SOGAC_Gestionar_Propiedades: El caso de uso comienza cuando el auditor selecciona una tabla en la ventana del explorador de archivos. El sistema debe mostrar activas, en la ventana de propiedades, las siguientes opciones dentro del marco Base de Datos: Datos, Historial, Estadísticas de campo, Total de Control y Criterio; además de mostrar el marco comentarios con la opción: Agregar comentario. Todas estas propiedades son específicas para cada base de datos, y constituyen escenarios del caso de uso. La selección de la opción Criterio implica la llamada al caso de uso CU_MEQ_Gestionar_Ecuación. El caso de uso termina cuando se selecciona otra tabla dentro del explorador de archivos.

2- CU_SOGAC_Gestionar_Vistas: El caso de uso comienza cuando el auditor selecciona en el menú Ver, la opción Vistas. El sistema muestra las siguientes opciones: Restaurar; Abrir Vista, Guardar Vista. El caso de uso concluye cuando se han guardado, restaurado o abierto la vista seleccionada por el auditor.

3- CU_SOGAC_Manejo_de_Campos: El caso de uso comienza cuando el auditor selecciona en el menú Ver, la opción Campos. El sistema muestra las siguientes opciones: Símbolo de Moneda, Separador de Miles y Configuración de Columna. Cada uno constituye un escenario, el caso de uso concluye al crearse una nueva Vista de la Base de Datos activa (Tabla Activa).

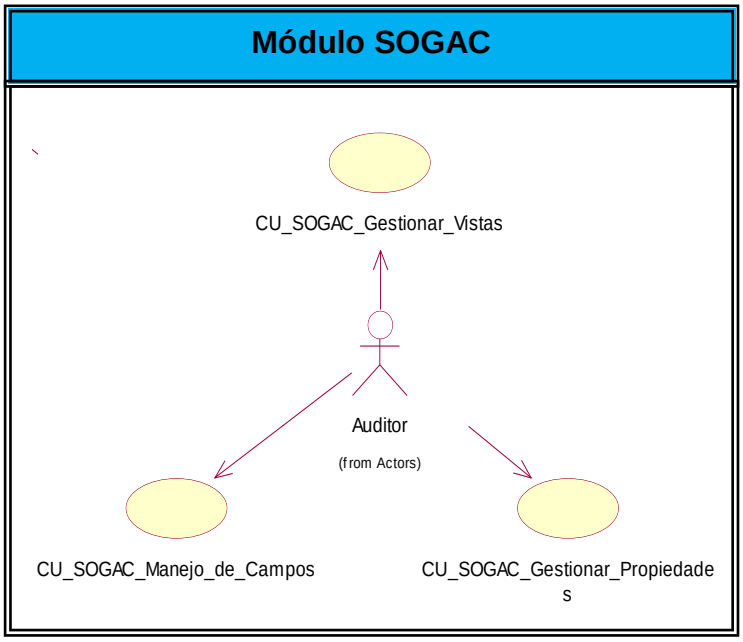


Fig. 15- Vista Casos de Uso Modulo SOGAC

2.3.5 Vista Lógica

En la descripción de la arquitectura, la vista lógica describe las clases más importantes que formarán parte del ciclo de desarrollo. Se describe los paquetes más abstractos del sistema y las relaciones que entre ellos existen ya sea de dependencia o de uso.

En la siguiente figura muestra la división del sistema en módulos, con el objetivo de promover la reusabilidad y facilitar el mantenimiento del sistema ante cambios en los procesos de negocio, garantizando que solo se modifique el módulo al que pertenece la funcionalidad afectada por el cambio.

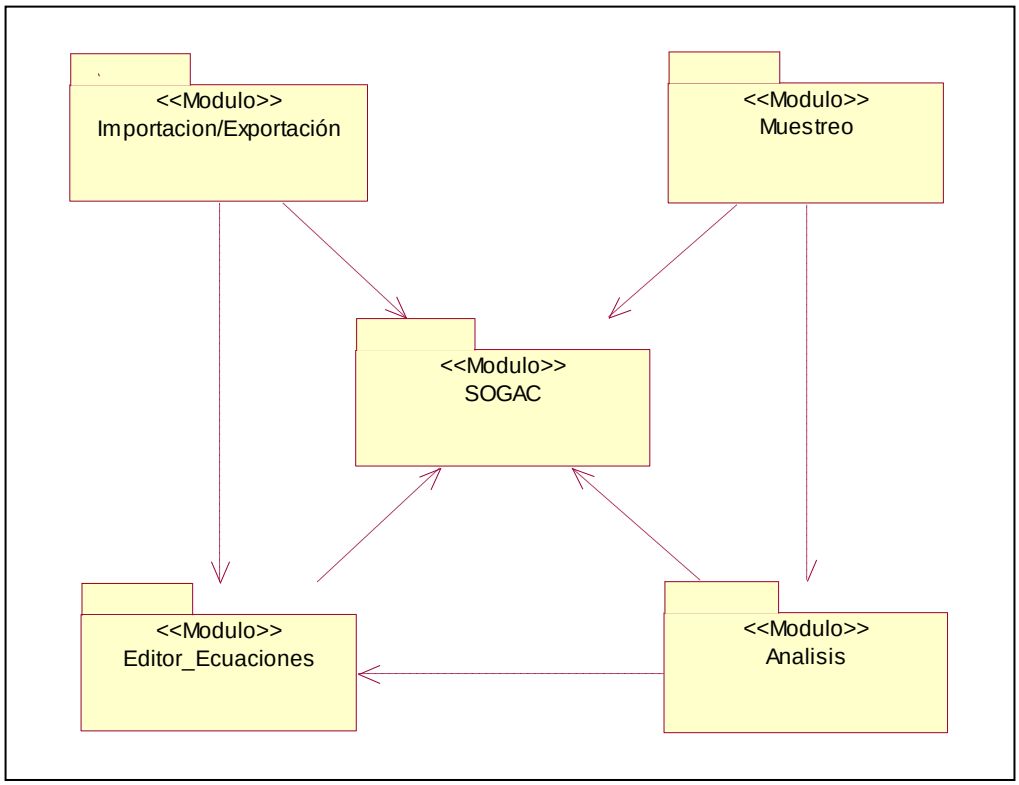


Fig.16- Representación de los Módulos del Sistema

Los principales módulos del sistema son:

Importación / Exportación: Este módulo es uno de los más importantes del software, pues en él se importan los datos desde diferentes fuentes y formatos, a la BD de la plataforma donde se realizarán las pruebas sustantivas y de cumplimiento por parte del auditor o personal especializado.

Muestreo: En este módulo se encuentran agrupados varios tipos de muestreos que son posibles aplicarles a los datos para saber si los mismos han sido alterados.

Análisis: Este módulo agrupa un conjunto de funcionalidades a aplicar por el auditor sobre los datos. Cada vez que se aplique una de estas funcionalidades a la tabla activa, lo que se obtenga será guardado como un Resultado para esa tabla

Editor de Ecuaciones: En este módulo se crean, modifican y/o guardan ecuaciones que pueden estar compuestas por un amplio conjunto de funciones definidas que se pueden aplicar sobre los datos de una tabla. Dicho módulo es un intérprete, este no es más que un programa que toma el código fuente (la ecuación), lo analiza y lo ejecuta directamente. Es uno de los módulos más importantes dentro del sistema, por su alta integración con los demás.

SOGAC: Este módulo tiene como función principal la integración del resto de los módulos, además de un conjunto de funciones básicas para el intercambio de información entre la aplicación y el cliente. Su estructura está definida y basada en interfaces.

La representación del sistema en capaz o subsistemas es la siguiente:

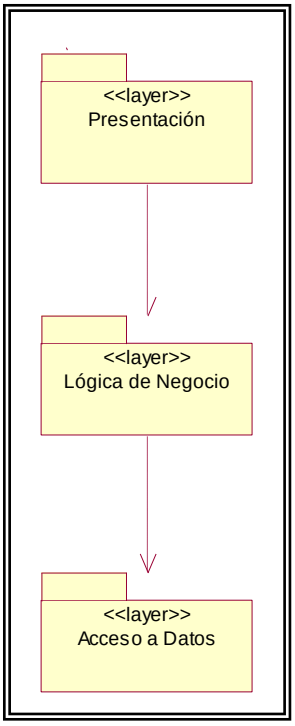


Fig. 17- Representación del Sistema en Capas

- **Presentación**: Clases y componentes del framework Swing.
- **Lógica de Negocio**: Conjunto de clases encargadas de gestionar todos los procesos de la lógica del negocio que con la ayuda de Spring Framework se logra una correcta manipulación y selección de la implementación de la interfaz a usar.

- **Acceso a Datos**: Conjunto de clases e Interfaces encargadas de manipular los datos con ayuda del framework Hibernate.

Un ejemplo de cómo fluye la comunicación entre los módulos a través de las capas lógicas de la aplicación mediante interfaces se representa en la siguiente figura:

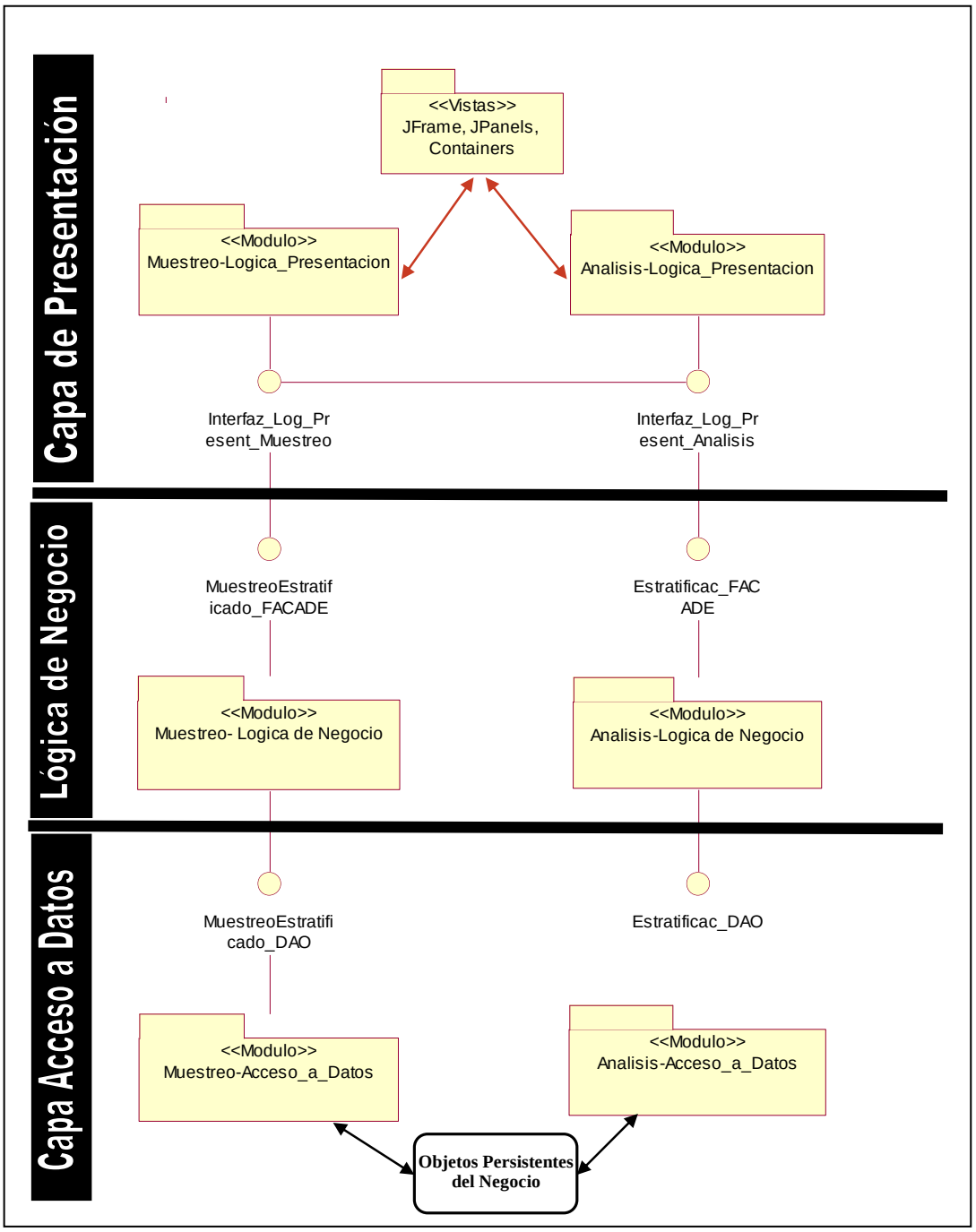


Fig. 18- Comunicación entre módulos y capas

Un concepto tratado en la figura, es “objetos persistentes del negocio (entidades)”. El mismo representa el modelo de objetos o conceptos reales, tales como, el resultado de una estratificación, una nueva tabla importada, el historial de acciones sobre determinada tabla.

Mecanismos de colaboración

Los mecanismos de colaboración son estrategias propuestas para establecer la forma de comunicación entre los componentes del software. Según las estructuras de código planteadas en la sección anterior, que dividen al sistema en estructuras: módulos, se proponen las estrategias de comunicación o colaboración entre ellos.

Si se detecta que dos módulos deben utilizar una misma funcionalidad, pueden existir varios escenarios para determinar la estrategia por la que se relacionarán. Si varios módulos de un subsistema comparten una misma funcionalidad y ninguno de ellos es responsable de implementarla entonces se está en presencia de una funcionalidad común entre ellos. Si uno de ellos es responsable o especialista en implementar la funcionalidad entonces la funcionalidad la realiza el módulo especialista, exporta la misma a los demás módulos y los mismos la consumirán. A continuación se explican los procedimientos a seguir en cada uno de los casos descritos.

Funcionalidad común.

En este caso donde los módulos comparten funcionalidades y no son de ninguno exclusivamente entonces se estará en presencia de funcionalidades comunes entre ellos, esta funcionalidad se colocará en el módulo común del subsistema (“comun”)

Funcionalidad especializada.

En este escenario, varios módulos o subsistemas, necesitan utilizar una funcionalidad sobre la cual sólo uno de ellos es especialista y ese es el que la implementará. En este caso los módulos no pueden consumir directamente la funcionalidad dado que la visibilidad entre módulos del árbol de dependencias solo les permite ver a los componentes de sus padres que por lo general son los módulos comunes.

El módulo especialista en la funcionalidad exportará la funcionalidad a una fachada de servicio para que los demás que la necesiten la puedan consumir.

El mecanismo por el que los módulos consumirán el servicio o la funcionalidad expuesta será utilizar la inyección directa mediante la clase base.ExternalBean.

Estructura de paquetes.

Organización de la aplicación por módulos.

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

Estructura de un módulo

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

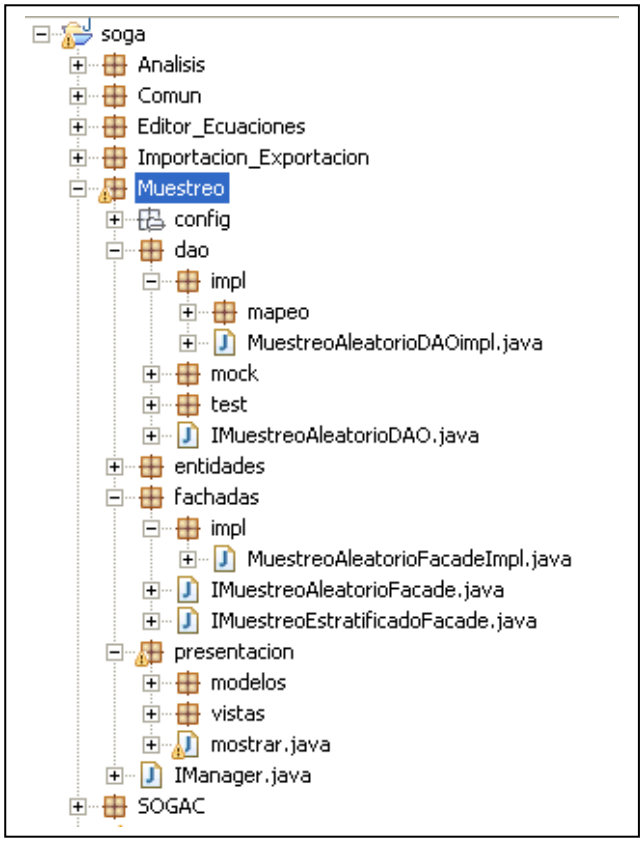


Fig. 19- Estructura de paquetes

configuration: Se localizarán todos los archivos que tienen que ver con la configuración del módulo, los "*Application-Context*" de Spring Framework, los archivos utilizados para la internacionalización, ficheros de propiedades *".properties"* y cualquier otro destinado a estos fines.

dao: Se encontrarán las interfaces DAOs.

dao.impl: Se encontrarán las implementaciones de las interfaces DAOs.

dao.impl.mapeo: Se encontrarán los ficheros de mapeo en XML de los DAO, que serán utilizados por Hibernate para lograr la persistencia

dao.test: Se encontrarán las clases de prueba utilizadas para realizar las pruebas de unidad a las implementaciones de los DAO usando el Framework JUnit.

dao.mock: Se encontrarán las implementaciones falsas de las interfaces DAOs, rellenándola con datos de prueba, que no son reales, para continuar trabajando de forma paralela al desarrollar las pruebas de unidad.

entidades: Se encontrarán todas las entidades persistentes o no del dominio pertenecientes al módulo.

fachadas: Se encontrarán las interfaces de las fachadas de negocio que brindarán los servicios.

fachadas.impl: Se encontrarán las implementaciones de las fachadas de negocio.

presentacion: Se encontrarán las interfaces de presentación.

presentacion.modelos: Se encontrarán las implementaciones de los modelos; de los componentes de Framework Swing.

presentacion.vistas: Se encontrarán las vistas de los componentes de Framework Swing y los controladores de la capa de presentación.

2.3.6 Vista de Implementación

La vista de implementación proporciona una descripción de las principales capas y subsistemas de los componentes de la aplicación. Los paquetes principales de la aplicación por cada uno de los módulos son:

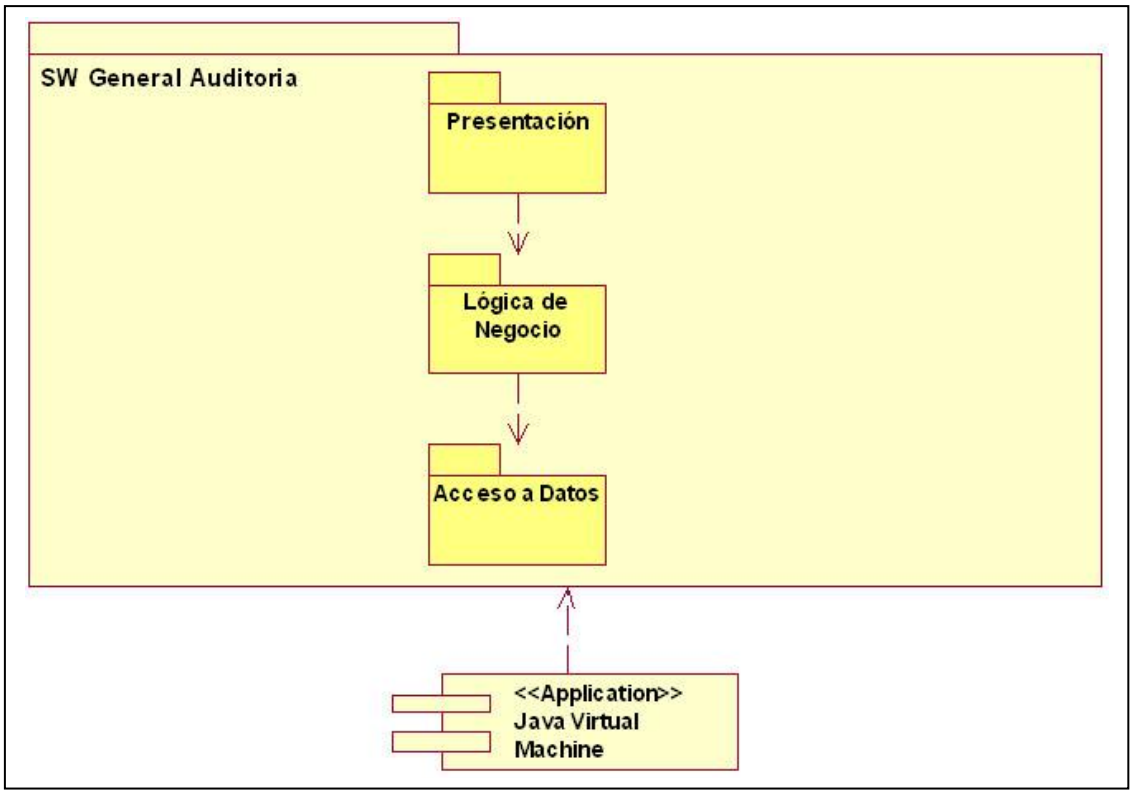


Fig. 20- Vista general del modelo de implementación

Cada uno de dichos paquetes encapsulan uno o más componentes que se interrelacionan entre ellos para darle solución a la aplicación y todos depende de la JDK que debe de estar instalada en la estación de trabajo donde se decida instalar la aplicación. Los componentes están distribuidos por capas y paquetes de la aplicación de la siguiente forma:

Capa de Presentación:

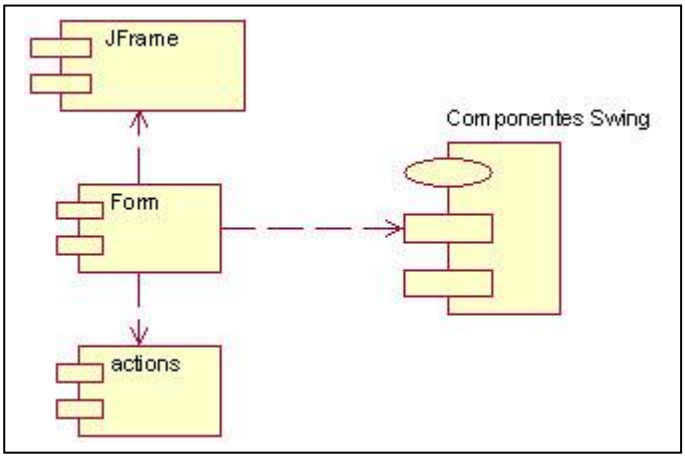


Fig. 21- Componentes de la capa de presentación

JFrame: Clase base de la cual heredan todas las formas de la aplicación.

Actions: contiene las clases Action que manejan el conjunto de eventos y las acciones a realizar por la presentación.

Componentes de Swing: Empaqueta el conjunto de componentes del Framework Swing.

Capa de Lógica del Negocio:

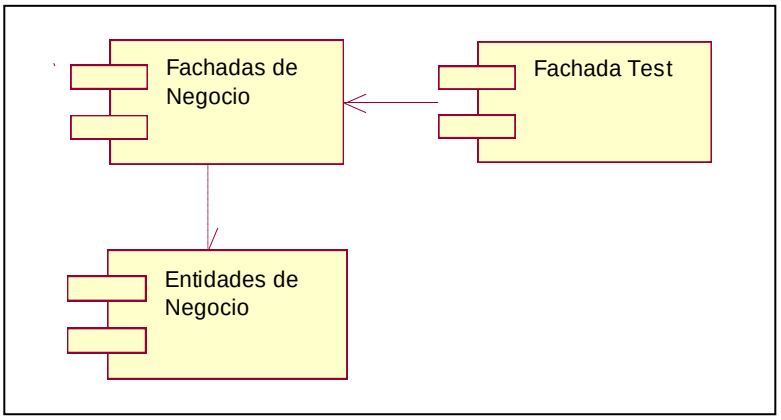


Fig. 22- Componentes de la capa lógica de negocio.

Entidades de Negocio: En este componente se agrupan todas las entidades del negocio.

Fachadas de Negocio: En este componente estará el conjunto de fachadas de esas entidades del negocio que contendrán los servicios que prestará la aplicación.

Fachada Test: Es un componente donde se implementarán las clases para las pruebas de unidad que con la ayuda del Framework JUnit se le harán a las entidades del negocio.

Capa de Acceso a Datos

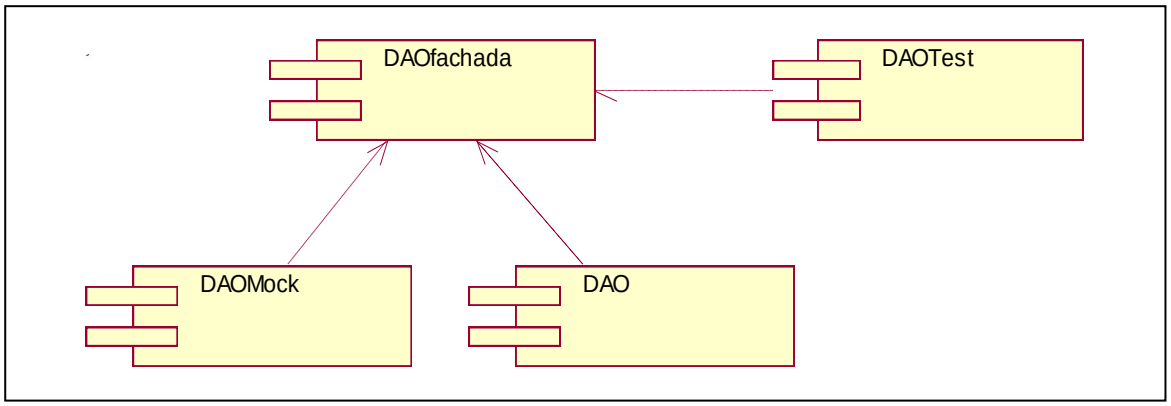


Fig. 23- Componentes de la capa de acceso a datos.

DAO: En este componente se agruparán los objetos de acceso a datos (DAO).

DAOMock: En este componente estarán las implementaciones falsas, o sea, con datos fijos de los DAOs.

DAOFachada: En este componente estarán las fachadas de cada DAO.

DAOTest: En este componente se implementarán las clases para las pruebas de unidad que con la ayuda del Framework JUnit se le harán a las fachadas de los DAO.

2.3.7 Vista de Despliegue

La vista de despliegue propone la distribución física del sistema y cómo estarán distribuidos los componentes de la aplicación en ellos, y cómo se satisfacen los requisitos no funcionales de software y hardware e influye en el rendimiento.

PC del Auditor: En esta computadora estará ubicada la aplicación. Contiene el ejecutable de la aplicación y la Java Virtual Machine (Maquina Virtual de Java).

Los dispositivos que se describen a continuación, no deberían formar parte de la vista de despliegue ya que en ellos no se encuentra ninguna capa lógica de la aplicación. Pero se ha considerado importante incluirlos para lograr una mayor ilustración del ambiente donde se desplegará el sistema, y además, porque sin la participación de alguno de ellos será imposible la utilización de la aplicación, pues no existirían datos para auditar.

PC del auditado: En esta computadora radicarán los datos que serán objetos de la auditoría. Estos datos pueden estar en un servidor de base de datos relacional, o pueden ser simplemente ficheros con información (*.xls, *.doc, *.pdf, *.txt, *.xml).

Dispositivo de Almacenamiento Externo: Este dispositivo puede ser cualquier tipo de dispositivo externo en el cual se transporten los datos a auditar desde la PC del Auditado hasta la PC del auditor. Ejemplo: memoria flash, cámara digital, disco duro externo, entre otros.

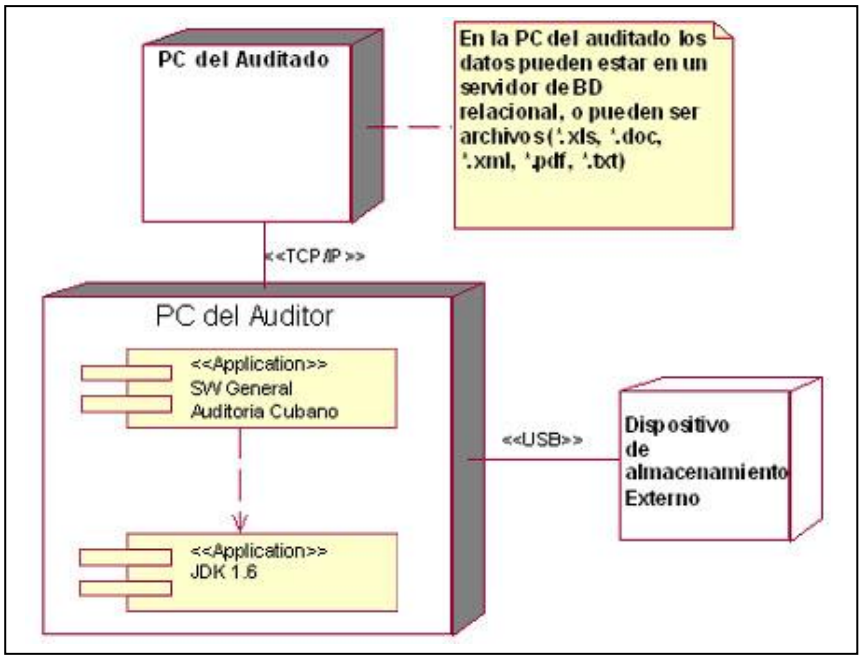


Fig. 24- Vista de Despliegue de la aplicación

2.4 Conclusiones

El propósito y objetivo fundamental de la arquitectura de software es crear la estructura o esqueleto del sistema, con unos pocos músculos de software, que cohesione las funcionalidades más relevantes, y sirva de soporte al resto de las funcionalidades finales. Todo esto hace del diseño de un software en particular, una tarea generalmente única. La propuesta de solución dada en este capítulo abarca dos documentos de obligada construcción, y constante refinamiento por parte del arquitecto de software del sistema. Durante la elaboración de la propuesta se hizo un estudio detallado de los principales aspectos de la descripción arquitectónica, así como una valoración de las principales tareas que debe llevar a cabo el rol de arquitecto según la metodología de desarrollo RUP (Proceso Unificado de Desarrollo).

Además se definieron los principales estilos arquitectónicos, sus componentes, configuraciones y restricciones impuestas para poder satisfacer las necesidades de los clientes finales. Las principales tecnologías y herramientas a utilizar durante el desarrollo se definieron en base a las necesidades del usuario final y la necesidad de mantener los principales atributos de calidad de la arquitectura de software.

La arquitectura de software propuesta cumple con:

- Las necesidades del usuario final.
- Puede ser construida por los desarrolladores.
- Es verificable, puede ser probada por los revisores.

CAPÍTULO 3: ESTRATEGIAS DE PRUEBAS PARA EVALUAR LA ARQUITECTURA

Hoy en día, las organizaciones reconocen la importancia y el valor de las arquitecturas de los sistemas de software para alcanzar sus objetivos. Una arquitectura correcta es el primer paso para el éxito y puede darle a las empresas una ventaja competitiva. Una arquitectura incorrecta, sin embargo, puede conducir un diseño a la calamidad. Por lo tanto sobre el arquitecto de software cae parte la responsabilidad del cumplimiento de los objetivos de las organizaciones.

Es vital para los arquitectos de software definir la estructura o estructuras del sistema, que contienen componentes de software, las propiedades externamente visibles de dichos componentes y las relaciones entre ellos; con el objetivo de razonar sobre la habilidad del sistema de alcanzar sus requerimientos de calidad, o de soportar la descomposición del sistema en partes independientemente implementables. Se puede identificar una conexión entre las decisiones del diseño hechas en la arquitectura y las cualidades y propiedades que deben tener dichos sistemas.

Si las decisiones arquitectónicas determinan los atributos de calidad del sistema, entonces es posible evaluar las decisiones arquitectónicas con respecto a su impacto sobre dichos atributos.

Evaluar es la manera más económica de prevenir los posibles desastres de un diseño arquitectónico, da una visión de los requerimientos de calidad que cumple o carece el sistema a implementar. Permite identificar dónde está el riesgo, fortalezas y debilidades identificadas de la arquitectura. Sirve para la toma de decisiones, por ejemplo:

- seguir el proyecto con las áreas de debilidad encontradas.
- reforzar la arquitectura en determinadas áreas.
- proponer nuevas herramientas de desarrollo.
- comenzar un diseño arquitectónico diferente.

¿Cuándo es recomendable evaluar?

Normalmente se evalúa la arquitectura una vez que se ha terminado y aún no se implementa. Se pueden desarrollar evaluaciones tempranas mientras se construye la arquitectura, o evaluaciones tardías cuando la arquitectura existe y la implantación se ha completado.

Existen dos reglas de oro que proponen Paul Clements, Rick Kazman y Mark Klein para determinar cuándo hacer una evaluación:

- Realizar una evaluación cuando el equipo de desarrollo comience a tomar decisiones que dependan de la arquitectura.
- Cuando el costo de realizar una evaluación, podría pesar menos, que la decisión de no llevarla a cabo.

¿Qué evaluar en una arquitectura de software?

Se evalúa el nivel de correspondencia y la capacidad y estrategias del diseño para cumplir con las especificaciones de los requerimientos de calidad. El potencial del diseño para responder o alcanzar los niveles requeridos en los requerimientos de calidad, dado las necesidades de la organización para la que se desarrolla la arquitectura del sistema.

Se pueden categorizar los requerimientos de calidad en dos, de desarrollo y operacionales:

- Los requerimientos de calidad de desarrollo son cualidades del sistema que son relevantes para la ingeniería de software, por ejemplo, mantenibilidad, reusabilidad y flexibilidad.
- Los requerimientos de calidad operacionales, son cualidades del sistema en operación, por ejemplo, rendimiento, robustez, tolerancias a fallos y confiabilidad.

Existen diferentes enfoques para evaluar los requerimientos de calidad, por ejemplo:

- Evaluación basada en escenarios
- Simulación
- Modelado matemático
- Experiencia
- Métricas

Existen diferentes técnicas de evaluar la arquitectura, clasificadas en cualitativas o cuantitativas, por ejemplo:

En las técnicas de evaluación cualitativas se pueden utilizar: escenarios, cuestionarios o listas de verificación. Por otro lado, en las técnicas de evaluación cuantitativas se pueden emplear: métricas, simulaciones, prototipos, experimentos o modelos matemáticos. La mayoría de los métodos de evaluación utilizan escenarios, que son secuencias específicas de pasos que involucran el uso o la modificación del sistema. Por lo regular, las técnicas de evaluación cualitativas son usadas cuando la arquitectura se encuentra en construcción, mientras que las técnicas de evaluación cuantitativas, se usan cuando la arquitectura ya ha sido implantada.

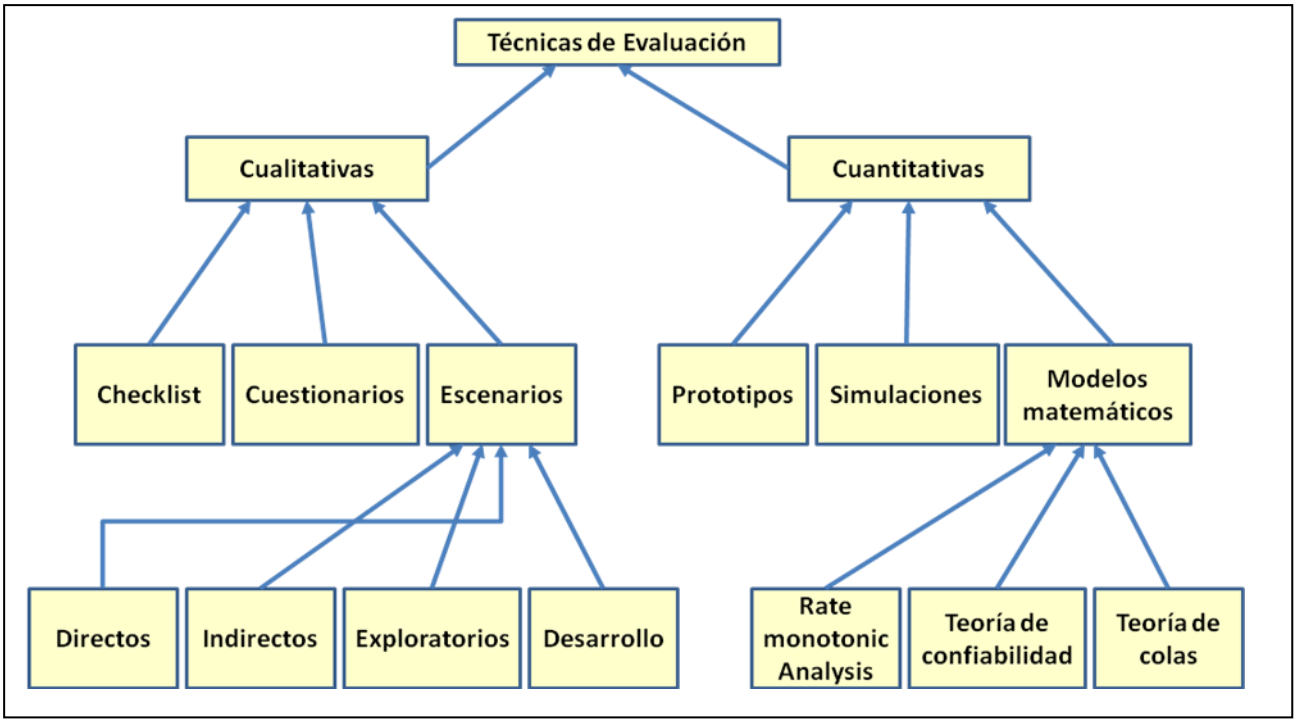


Fig. 25- Técnicas de Evaluación

3.1 Métodos de evaluación de Arquitecturas de Software.

Existen métodos de evaluación de las arquitecturas como los propuestos por Clements, Kazman y Klein entre ellos están:

- Software Architecture Analysis Method (SAAM) con tres perspectivas para el análisis de la arquitectura y cinco pasos para el análisis; este método está basado en escenarios y permite evaluar una arquitectura o comparar varias. Es más usado cuando el requerimiento de calidad: modificalidad es el de mayor interés.
- Architecture Tradeoff Analysis Method (ATAM) el cual esta diseñado para producir como respuestas las metas comerciales tanto del sistema como de la arquitectura, y usar esas metas y la participación de los stakeholders para centrar la atención de los evaluadores en la porción de la arquitectura que es esencial para el cumplimientos de dichas metas. No solo dice cuan bien satisface las metas de calidad, sino que provee ideas de cómo esas metas de calidad interactúan entre ellas. Es más profundo para evaluar aspectos como rendimiento y confiabilidad.
- ARID(An Evaluation Method for Partial Architectures) Este método es un híbrido de ADR(Revisiones de diseño activas) y ATAM que tienen características útiles para la evaluación de diseños preliminares, por lo que es conveniente para realizar la evaluación

de diseños parciales en las etapas tempranas del desarrollo y usarlo para evaluar la factibilidad de la arquitectura.

Las revisiones de diseño activas, ADR constituye una técnica efectiva para asegurar la calidad del diseño detallado en el software. Es utilizada para la evaluación de diseños detallados de unidades del software como los componentes o módulos. Las preguntas giran en torno a la calidad y completitud de la documentación y la suficiencia, el ajuste y la conveniencia de los servicios que provee el diseño propuesto.

3.2 Comparación entre los diferentes métodos de evaluación de la arquitectura.

	ATAM	SAAM	ARID
Atributos de Calidad contemplados	No está orientado a ningún atributo específico, pero se hace énfasis en: -Modificabilidad -Seguridad -Confiabilidad -Desempeño	Principalmente -Modificabilidad -Funcionalidad	-Conveniencia del diseño evaluado
Objetos analizados	-Estilos arquitectónicos -Documentación -Flujo de datos -Vistas Arquitectónicas	-Documentación -Vistas Arquitectónicas	-Especificación de los componentes
Etapas del proyecto en las que se aplica.	-Luego de que el diseño de la arquitectura ha sido establecido.	-Luego de que la arquitectura cuenta con funcionalidades implementadas.	-A lo largo del diseño de la arquitectura.
Enfoques utilizados	-Árbol de utilidades y lluvia de ideas para articular los requerimientos de calidad -Análisis Arquitectónico que detecta	-Lluvia de ideas para escenarios y articular los requerimientos de calidad -Análisis de	-Revisiones de diseños, lluvia de ideas para obtener escenarios

	puntos sensibles, puntos de balance y riesgos	los escenarios para Verificar funcionalidad o estimar el costo de los cambios	
Recursos necesarios	Normalmente 3 días más el tiempo de preparación y del sumario del ejercicio aplicado, incluye al cliente, arquitecto, stakeholders y 4 personas en el equipo de evalución	Normalmente 2 días más el tiempo del sumario del ejercicio aplicado, incluye al cliente, arquitecto, stakeholders y 3 personas en el equipo de evaluación	Normalmente 2 días más el tiempo de preparación y del sumario del ejercicio aplicado, incluye al arquitecto, diseñador, stakeholders y 2 personas en el equipo de evaluación

¿Quiénes participan en una evaluación?

La evaluación de la arquitectura las hacen arquitectos, analistas, probadores y la administración del proyecto. También puede contratarse a un grupo de personas especialistas para desarrollarla. Existen algunos métodos que proponen la vinculación directa de la organización o cliente del proyecto, interesándose por los resultados de la evaluación, para tomar la decisión de continuar o no con el proyecto.

¿Quiénes participan en una evaluación en la Universidad de las Ciencias Informáticas (UCI)?

En la universidad existe un grupo de calidad de software dentro de la infraestructura productiva (IP), que evalúan las arquitecturas que se proponen en los diferentes proyectos.

3.3 Evaluando la arquitectura propuesta

Los métodos de evaluación incluyen los escenarios, la simulación, los modelos matemáticos, el prototipo, entre otros, los cuales permiten una evaluación temprana. Sin embargo, no todas las técnicas son aplicables y efectivas para cualquier atributo de calidad. En el presente trabajo se hace

una evaluación para determinar el cumplimiento de los requerimientos de calidad que engloban la "utilidad" del sistema (desempeño, disponibilidad, seguridad, modificabilidad, usabilidad, etc.), especificados en forma de escenarios.

La propuesta de esta arquitectura se hace de manera temprana en relación con el desarrollo del SW como tal. Por esto se necesita la evaluación en las fases tempranas del diseño, se propone la utilización de las características que proveen tanto ADR como ATAM por separado. De ADR, resulta conveniente la fidelidad de las respuestas que se obtiene de los involucrados en el desarrollo. Así mismo, la idea del uso de escenarios generados por los involucrados con el sistema es tomada del ATAM. De la combinación de ambas filosofías surge ARID para efecto de la evaluación temprana de los diseños de una arquitectura de software que es el método utilizado en la evaluación del trabajo. Se decidió pre-validar la arquitectura evaluando cómo las vistas seleccionadas responden a cada uno de los principales requerimientos y restricciones del sistema.

Enfocándose en los siguientes escenarios:

- Seguridad del sistema
- Portabilidad
- Modificabilidad y evolución del sistema
- Rendimiento del sistema

Seguridad del sistema: En la vista de despliegue se muestra los puntos de contactos con el exterior, así como el software encargado de establecer la interacción. El empleo del framework Spring e Hibernate brinda las fortalezas necesarias para conservar la integridad y disponibilidad de los datos.

Portabilidad del sistema: El sistema será desarrollado utilizando la plataforma de desarrollo Java, sobre la máquina virtual de Java, funciona en plataformas diferentes como Linux, Windows y el Unix, soporta gran variedad de bases de datos embebidas como HSQLDB y SQLlite.

Modificabilidad. Para analizar el impacto que puede tener un cambio en el sistema se puede consultar la vista lógica, donde el uso de las capas muestra como un cambio en una capa más baja puede ser ocultado detrás de sus interfaces y no impactará las capas encima de ella. También es muy útil la descripción de la arquitectura horizontal y la vista de implementación.

Rendimiento del sistema. Se debe consultar la vista lógica, y la vista de casos de usos para analizar la concurrencia en el sistema. También se puede observar la vista de despliegue.

Producto a que la arquitectura del sistema está detallada desde un nivel de abstracción bastante alto, en este capitulo se realizó una pre-evaluación de la arquitectura propuesta y descrita en el Capítulo 2, mostrando como se puede analizar el cumplimiento los atributos de calidad atendiendo a cada una de las vistas desarrolladas.

3.3.1 Evaluando la arquitectura según las normas UCI

Como se mencionó anteriormente en la UCI existe un grupo revisor de las arquitecturas de software. Dicho grupo basa su trabajo en un documento oficial para la descripción de la arquitectura, que es de uso interno de la universidad. Para que la propuesta de arquitectura se ajuste a dichas normas; y con el objetivo de sentar las bases para su futura evaluación por parte del equipo revisor de la universidad, se procederá a incorporar a nuestra investigación los aspectos más relevantes de dicho documento.

Atributos de la Arquitectura de Software.

TECNOLOGICOS	
Modelo de persistencia	Framework Hibernate
Manipulación de excepciones	Jerarquía de excepciones propia del sistema
Manipulación de transacciones	
• Nivel de servicio.	
• Nivel de negocio.	Manipulación de transacciones a niveles de objetos de negocio con Spring, a través la programación orientada a aspectos.
• Nivel de acceso a datos.	
Suscripciones a eventos y notificaciones.	No requiere
Mensajería, protocolos sincrónicos y asincrónicos.	No requiere
Manipulación de dispositivos externos	Cámara | Scanner | Impresora | Otros:
Manipulación de estándares de información.	
Generación de reportes.	Se lograr la generación de Reportes con JasperReport en el módulo de Importación/Exportación.
CALIDAD	
Modificabilidad Adaptación a las variaciones de los requisitos	

Mantenibilidad *Resolución de problemas*	
Aplicación de patrones que permitan bajo acoplamiento y la alta cohesión del diseño arquitectónico y del diseño modular.	
• Presentación: Model-View-Crontroller.	Se hace uso del patrón. Remitirse a la sección 2.2.3 Organigrama de la Arquitectura, dentro de este documento
• Despliegue Lógico: Layered Aplication.	Se hace uso del patrón. Remitirse a la sección 2.2.3 Organigrama de la Arquitectura, dentro de este documento
• Despliegue Físico: Tiered Distribution.	No
Capacidad de prueba. Posibilidad de verificar el estado de los componentes del sistema.	Uso clases de prueba de unidad a los DAO y las clases del negocio
Extensibilidad *Extender el software con nuevas funcionalidades*	
Aplicación de modelos arquitectónicos orientados a componentes o plugins que permitan hacerlo extensible.	La distribución del sistema en capas y la modularidad permite que el mismo se haga extensible.
Reestructuración *Reorganización y ubicaciones de módulos*	
Transformar la forma de representación a otra al mismo nivel de abstracción, preservando el objetivo del comportamiento externo del	Al trabajar orientado a interfaces se puede

sistema.	cambiar la implementación de una capa, sin que se afecten las demás
Interoperabilidad *Integración con otros sistemas*	
Utilización de estándares de información para la configuración e integración entre sistemas o subsistemas.	Actualmente inexistentes los estándares, dada la inexistencia de otros software de auditoría cubano.
Exposición de las funcionalidades vía servicios Web que permita la reutilización de los componentes a través de protocolos abiertos.	No
Seguridad Condición resultante de establecer y mantener medidas de protección que aseguren un estado de inviolabilidad ante actos o influencias hostiles.	
No Repudio	
Garantía que un emisor no pueda negar el envío de un mensaje.	No requiere
Confidencialidad	
Garantía de la confidencialidad de los datos e información, solo puede ser accedida por una la entidad autenticada que tenga privilegios de acceso.	El acceso a la BD es con usuario y contraseña que solo el sistema puede manipular.
Integridad	
Garantía de la integridad (no corrupción) de los datos e información almacenada.	El sistema de BD, HSQLDB es relacional y vela por la integridad referencial.
Aseguramiento	

Garantía de no pérdida de datos e información.	El sistema de BD, HSQLDB posee una copia de seguridad de la BD en un fichero que se llama: Nombre_BD.backup.
Disponibilidad	
Capacidad de atender múltiples solicitudes de ejecución de una operación.	No requiere concurrencia.
Capacidad de recuperarse o trabajar en condiciones extremas voluntarias o involuntarias para las que no ha sido concebido (tolerancia a fallos).	Si
Capacidad para la clusterización o el balance de carga ante el aumento de la demanda de los servicios, o altos niveles de concurrencia.	No requiere.
Auditoría	
Capacidad de registrar los eventos ocurridos o acciones ejecutadas sobre el sistema.	Cada tabla posee un historial en el que se reflejan todos los cambios ocurridos sobre ella. Además el sistema de BD posee un fichero donde se registran todos los eventos ocurridos sobre la BD.
Rendimiento Grado en el cual el sistema o componente lleva a cabo una funcionalidad específica dada una restricción de velocidad, precisión, etc. y el uso eficiente de los recursos**.**	
Eficiencia	
Eficiencia Operacional	
Automatización de tareas programadas.	No requiere

Provee una configurable automatización del sistema y de procesos de flujos de trabajo (workflows).	
Requiere que los datos sean ingresados una sola vez (no re tecleados)	Los datos son ingresados en la aplicación una sola vez. Después de eso pasan a ser de solo lectura para evitar modificaciones de los mismos.
Provee facilidad de importación de datos de otros programas.	Posee un módulo de importación. Dirigirse a la sección 2.3.4 Vista de casos de uso del sistema, en este documento.
Opera rápida y eficientemente, con aceptable rendimiento en términos de tiempo de respuesta.	Gran velocidad en el acceso a datos, pues cuenta con una BD de datos embebida, que tiene excelentes tiempos de respuesta a las peticiones.
Permite procesamiento en tiempo real y / o en lote.	Permite el procesamiento en lote a la hora de importar.
Permite su ejecución en hardware	
• Altas prestaciones.	
• Medias prestaciones.	Remitirse a la sección 2.3.3.1 Requerimientos de Hardware

• Bajas prestaciones.	
Permite su ejecución independiente del sistema operativo.	Solo necesita de la Maquina Virtual de Java para correr.
Fácil uso	
Fácil de aprender, con:	
Sistema estándar de comandos.	
Codificación de color.	Selección de colores agradables a la vista que orientan al usuario.
Facilidades de ayudas en la pantalla.	Globos de sugerencias en pantalla.
Claramente escritas guías de usuario y documentación del sistema.	Si
Fácil de usar, con:	
Completo acceso a todas las funcionalidades del sistema, sujeto a individuales perfiles de seguridad asociado a usuarios.	Si
Navegación por el sistema es lineal (straight forward)	Si
Teclas de acceso directo, menús desplegables, scrolls.	Si
Multitarea (múltiples aplicaciones / ventanas abiertas al mismo tiempo)	No requiere. Posee un solo marco de trabajo para el auditor.
En la pantalla se muestra avisos y mensajes con claridad y útiles (entradas requeridas, validación de errores por entradas, o errores de procesamiento)	Notificación de errores en datos de entrada.
Escalabilidad	
Capacidad expandir la solución para soportar gran cantidad de usuarios concurrentes sin degradar el rendimiento del sistema.	No requiere concurrencia
Capacidad aumentar los volúmenes de datos a procesar sin	Si

impactar de forma significativa en el rendimiento.	
Multiprocesamiento	
Estructuras para programación concurrente.	
• Thread.	Si
• Critical Sections | Mutex | Lockers.	
• Sincronización a nivel de lenguaje.	Si
Control de concurrencia	
• Pesimista	Si
• Optimista	
Optimización de recursos	
Manejo de memoria.	Si
Manipulación del ciclos de vida de los objetos	Con Spring se logra un excelente control del ciclo de vida de los objetos
• Recolección de basura.	El entorno de desarrollo, Netbeans 6.0 tiene su propio recolector de basura
• Reciclaje de objetos (Típicamente de Conexiones e Hilos a través de ConnectionPool o ThreadPool respectivamente)	Se reciclan los objetos de conexión a las BD
Mecanismos de cache.	Si
AMBIENTE DE DESARROLLO	
Componentes técnicos horizontales	
Sistemas operativos.	Windows, Linux, Unix
Seguridad (Antivirus, Autenticación)	
Gestión de recursos (Monitoreo)	
Salvas automáticas.	
Control de versiones.	Subversion
Gestión documental.	dotProject
Control y seguimiento de errores.	
Gestión de proyecto.	dotProject

Componentes técnicos verticales	
Lenguajes de programación.	Java
Frameworks, toolkits o bibliotecas de componentes.	Swing, Spring, Hibernate
Compiladores, maquina virtuales, interpretes.	Máquina Virtual de Java
Frameworks para realización de pruebas unitarias.	JUnit
Sistemas automatizados de compilación (Continuous Integration).	No
Servidor de aplicaciones.	No requiere
Herramientas de modelado.	Rational Rose 2003
Herramientas de generación de documentación a partir del código fuente.	NetBeans 6.0
Entornos de desarrollo integrados (IDE)	NetBeans 6.0
Formateado de código.	Si
Refactorización.	Si
Sistema gestor de bases de datos.	HSQLDB
POLÍTICOS Y LEGALES	
Licencias de software cada uno de los componentes a utilizar.	Todas las herramientas son libres, a excepción del Rational Rose 2003
Certificaciones obligatorias a obtener.	
Leyes y regulaciones con las que debe cumplir el sistema.	

[Dirección Técnica IP, 2008]

3.4 Conclusiones

En este capítulo se analizaron los métodos estandarizados para la evaluación de la arquitectura. Además se llegó a la conclusión de que el más efectivo a aplicar es el ARID, debido a que es la primera propuesta que se hace en una fase temprana del desarrollo del sistema.

Por otra parte se sentaron las bases sentadas para que la propuesta pueda ser evaluada siguiendo los estándares de revisión de arquitecturas que rigen actualmente en la UCI.

CONCLUSIONES

1. Se realizó un estudio del estado del arte sobre arquitectura para software general de auditoría y TAAC encontrando así, cuál de las características presentes en ellos eran la más apropiadas para aplicar a la descrita en este trabajo.

2. Se seleccionó y argumentó la metodología de desarrollo: Proceso Unificado de Desarrollo (RUP) y se realizó una valoración de las principales tareas que debe llevar a cabo el rol de arquitecto según esta metodología.

3. Se cumplió con cada uno de los aspectos de la descripción de la arquitectura propuesta por la metodología de desarrollo, y se propuso la implementación de componentes de software para facilitar la reutilización de los mismos.

4. Con el trabajo desarrollado se generaron los artefactos necesarios que contemplan y permiten la implementación del sistema a desarrollar.

5. Se estableció una arquitectura base para un software general de auditoría, sobre la cual se pueden ir agregando nuevos componentes y funcionalidades.

6. Se definió los principales estilos arquitectónicos a aplicar, sus componentes, configuraciones y restricciones, impuestas por las necesidades de los usuarios finales.

7. Se realizó un estudio detallado de las principales categorías de patrones arquitectónicos, se fundamentó la elección de los mismos para su aplicación en el sistema y se brindó la solución al mismo.

8. Dada las restricciones impuestas por algunos requisitos no funcionales del cliente y en función de los principales atributos de calidad de la Arquitectura de Software se definieron las principales tecnologías y herramientas a utilizar en el desarrollo del sistema, y se configuró cada uno de los puestos de trabajo por roles en el equipo de desarrollo.

9. La Arquitectura de Software propuesta cumple con: satisface los requisitos de los clientes, es “construible” por los desarrolladores, es “verificable” puede ser evaluada por los revisores.

10. La propuesta de arquitectura permitirá construir una herramienta totalmente libre, cosa importante y única en todo el mundo; pues todo el software de auditoría existente a nivel mundial es propietario.

11. Es difícil, o prácticamente imposible probar una arquitectura en una fase tan temprana del desarrollo de software. Lo que más se puede acercar a verificar qué tan factible es o no la arquitectura, es una evaluación de las principales características que toda arquitectura debe poseer (robusta, flexible, re-usable), así como el cumplimiento de la metodología de desarrollo que se propone y la compatibilidad y acoplamiento de las herramientas que se propongan para llevar a cabo el desarrollo del software

12. Se logró diseñar y modelar una arquitectura que cumple con los requisitos funcionales y no funcionales del sistema al cual da solución informática, por tanto se cumplió con el objetivo de diseñar una arquitectura que permita tomar decisiones sobre los aspectos dinámicos y estáticos más significativos del sistema.

RECOMENDACIONES

- Mantener un constante refinamiento de la arquitectura a lo largo del ciclo de desarrollo.
- El uso de estándares de codificación de información, para fortalecer la comunicación entre diferentes sistemas de auditoría informática.

- La revisión de cada uno de los escenarios de la aplicación para evaluar a fondo las restricciones que se le imponen a la arquitectura.
- Realizar una evaluación de los costos de la tecnología utilizada, aunque la mayor parte de esta es libre, es necesario evaluar los costos del hardware que la soportan.
- Validar la factibilidad de la construcción del proyecto SOGAC, con la evaluación por parte del grupo de calidad, del documento de evaluación de arquitectura de software de la infraestructura productiva, para ratificar los resultados de la evaluación con el método ARID.
- La presentación de esta propuesta de arquitectura a auditores y personal calificado que necesite de una herramienta para auditar sistemas contables y financieros en su labor diaria.
- Que finalmente, después de quedar validada la propuesta de arquitectura por parte del equipo revisor técnico de la UCI, se lleve a cabo la construcción del software; para dotar de esta manera al país de una herramienta poderosa en la detección de fraudes y delitos sobre los sistemas contables y financieros.

REFERENCIAS

- [Booch et al., 1999] Booch, G. et al. (1999). *The Unified Modeling Language User Guide*. Addison-Wesley.
- [Booch, Rumbaugh & Jacobson, 1999] Grady Booch, James Rumbaugh e Ivar Jacobson. El Lenguaje Unificado de Modelado. Madrid, Addison-Wesley, 1999.
- [Brooks Jr., 1975] Frederick Brooks Jr. The mythical man-month. Reading, Addison-Wesley, 1975.
- [CRAIG & RYAN, 2005] Craig Walls & Ryan Breidenbach, 2005. “Spring in Action”.
- [Dijkstra, 1983] Edsger Dijkstra. “The Structure of the THE Multiprogramming system.” Communications of the ACM, 26(1), pp. 49-52, Enero de 1983.
- [Dirección Técnica IP, 2008]. Dirección Técnica de Infraestructura Productiva UCI. Documento de Evaluación de Arquitectura de Software. UCI 2008.
- [Gamma, 2002] Gamma, E. (2002). Patrones de Diseño, Addison-Wesley.
- [Garlan & Shaw, 1994] David Garlan y Mary Shaw. “An introduction to software architecture”. CMU Software Engineering Institute Technical Report, CMU/SEI-94-TR-21, ESC-TR-94-21, 1994.
- [GoF95] Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, Reading. MA, 1995.
- [Gómez, 2006] Gómez, S. P. (2006) Swing Univ. Politécnica de Madrid
- [HSQLDB Development Group, 2008] Grupo de desarrollo de HSQLDB, 2008. http://hsqldb.org 22/05/2008.
- [Jacopbson, 1999] Jacopbson, I. (1999). The Unified Software Development Process, Addison Wesley.
- [KAISER 2005] KAISER, S. H. Software Paradigms. 2005.
- [Kruchten, 1995] Kruchten, P. B. (1995). The 4+1 View Model of Architecture. *IEEE Software*.

- [Palos, 2006] Juan Antonio Palos (2006) Catálogo de Patrones de Diseño J2EE. Y II: Capas de Negocio y de Integración, Sun Microsystem, http://www.programacion.net/java/tutorial/patrones2/8/ . 15/04/2008
- [Perry & Wolf, 1992] Dewayne E. Perry y Alexander L. Wolf. “Foundations for the study of software architecture”. ACM SIGSOFT Software Engineering Notes, 17(4), pp. 40–52, Octubre de 1992.
- [Rumbaugh et al., 1999] Rumbaugh, J. et al. (1999). *The Unified Modeling Language Reference Manual*. Addison Wesley.
- [Shaw, 1996]Mary Shaw. “Some Patterns for Software Architecture,” en Pattern Languages of Program Design, Vol. 2, J. Vlissides, J. Coplien, y N. Kerth (eds.), Reading, Addison-Wesley, pp. 255-269, 1996.
- [SQLite, 2007] Grupo de Desarrollo SQLite, 2007. http://www.sqlite.org. 20/05/2008
- [Sun Microsystem, 2005] Microsystem, S. (2005). "The Java(tm) Virtual Machine Specification."
- [White & Lemus, 1997] Sharon White y Cuauhtémoc Lemus-Olalde. “The software architecture process”. http://nas.cl.uh.edu/whites/webpapers.dir/ETCE97pap.pdf, 1997.

BIBLIOGRAFÍA

- Adrian Lasso, 2003. Arquitectura de Software
- Christian Bauer & Gavin King, 2005. Hibernate in action.
- ComputerWorld. Java Database Connectivity por Carol Sliwa http://www.computerworld.com/cwi/story/0,1199,NAV47-68-85-98_STO43545,00.html. 10/02/2008.
- Craig Walls & Ryan Breidenbach, 2005. “Spring in Action”.
- David Garlan y Mary Shaw. “An introduction to software architecture”. CMU Software Engineering Institute Technical Report, CMU/SEI-94-TR-21, ESC-TR-94-21, 1994.
- Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, Reading, MA, 1995.
- Erika Cmacho, Fabio Cardeso, Gabriel Nuñes. Arquitecturas de Software, Guía de Estudio. Abril 2004.
- Gamma, E. (2002). Patrones de Diseño, Addison-Wesley.
- Gómez, S. P. (2006) Swing Univ. Politécnica de Madrid.
- Grupo de desarrollo de HSQLDB, 2008. http://hsqldb.org 22/05/2008.
- Grupo de Desarrollo SQLite, 2007. http://www.sqlite.org 20/05/2008.
- Hoffman. Tutorial de SQL -- http://w3.one.net/~jhoffman/sqltut.htm 20/02/2008
- http://www.caats.ca/. 12/03/2008
- Jacopbson, I. (1999). The Unified Software Development Process, Addison Wesley.
- Joe Dayz, 2007. Spring – JDBC. &
- Juan Antonio Palos (2006) Catálogo de Patrones de Diseño J2EE. Y II: Capas de Negocio y de Integración, Sun Microsystem, http://www.programacion.net/java/tutorial/patrones2/8/ 17/01/2008.
- Kruchten, P. B. (1995). The 4+1 View Model of Architecture. *IEEE Software*.
- Lasso, A. Arquitectura de Software. Disponibilidad http://www.microsoft.com/spanish/msdn/comunidad/mtj.net/voices/art110.asp. 05/02/2008
- Martin Fowler, 2007. Contenedores de Inversión de Control y el patrón de Inyección de Dependencias. http://www.programacion.net/java/articulo/jap_injection/ 10/05/2008
- Mauricio Naranjo. Lucasian Labs Ltda, 2005. Fundamentos de Definición de Arquitectura de Software.
- Normas internacionales de Auditoría. http://fccea.unicauca.edu.co/old/taac.htm. 10/05/2008

- Paul Clements, Rick Kazman & Marck Klein. Evaluation Software Architecture: Methods and Case Studies.
- Ralph R.Young, 2004.The Requirements Engineering Handbook.
- Reynoso,C. B. Introducción a la Arquitectura de Software, 2004. Disponible en http://www.microsoft.com/spanish/msdn/arquitectura/roadmap_arq/intro.asp. 21/04/2008
- Sharon White y Cuauhtémoc Lemus-Olalde. "The software architecture process". http://nas.cl.uh.edu/whites/webpapers.dir/ETCE97pap.pdf . 02/02/2008
- Steve Adolph & Paul Bramble, 2001. Patterns for effective Use cases.
- Sun Microsystems. JDBC Data Access API -- http://java.sun.com/products/jdbc/index.html 04/05/2005
- Sun Microsystems. JDBC Data Access API: Articles and Success Stories http://java.sun.com/products/jdbc/articles.html 04/05/2008
- Sun Microsystems. JDBC Data Access API: Drivers o Programas Pilotos http://industry.java.sun.com/products/jdbc/drivers 04/05/2008

GLOSARIO

Aplicación Web: Sistema informático que los usuarios utilizan accediendo a un servidor Web a través de Internet o de una intranet.

Bean binding: Es la capacidad para mantener sincronizadas propiedades de dos objetos. Un énfasis adicional ha sido puesto en atar componentes Swing y la fácil integración con IDEs como Netbeans.

Byte-code: El bytecode es un código intermedio más abstracto que el código máquina. Habitualmente es tratado como un fichero binario que contiene un programa ejecutable similar a un módulo objeto, que es un fichero binario producido por el compilador cuyo contenido es el código objeto o código máquina.

CASE: Ingeniería de Software Asistida por Ordenador (Computer Aided Software Engineering) son diversas aplicaciones informáticas destinadas a aumentar la productividad en el desarrollo de software reduciendo el coste de las mismas en términos de tiempo y de dinero.

HTTP (HyperText Transfer Protocol): Protocolo de transferencia de hipertexto, es el protocolo usado en cada transacción de la Web (WWW). El hipertexto es el contenido de las páginas web, y el protocolo de transferencia es el sistema mediante el cual se envían las peticiones de acceso a una página y la respuesta con el contenido.

IBM: IBM (International Business Machines): es una empresa que fabrica y comercializa hardware, software y servicios relacionados con la informática.

IDE: (Integrated Development Environment). Es una aplicación de software que proporciona facilidades a programadores de computadoras para el desarrollo de software. Normalmente consiste en editores de códigos fuentes, un compilador o intérprete.

IEEE: Corresponde a las siglas de The Institute of Electrical and Electronics Engineers, El Instituto de Ingenieros Eléctricos y Electrónicos, una asociación técnico-profesional mundial dedicada a la estandarización, entre otras cosas. Es la mayor asociación internacional sin fines de lucro formada por profesionales de las nuevas tecnologías, como ingenieros eléctricos, ingenieros en electrónica, científicos de la computación e ingenieros en telecomunicación.

IRIX: Es un sistema operativo compatible con Unix, creado por SGI (Silicon Graphics) para su plataforma MIPS de 64 bits. IRIX tiene un particular soporte para gráficos 3D, video y transferencia de datos de gran ancho de banda. Fue una de las primeras versiones de Unix en tener una interfaz gráfica de usuario (GUI) para el escritorio principal y es actualmente ampliamente utilizado, debido a su extremadamente alta calidad en gráficos 3D, en la industria de la animación por computadora y para visualización científica.

Kernel: Kernel en inglés quiere decir " núcleo" y es de hecho la parte principal del sistema operativo, la que se ocupa de gestionar los recursos de la memoria, habilitar el acceso a los sistemas de archivo, gestionar las funciones de red.

OS/360: Sistema operativo producido por IBM.

p-code: En la programación de computadoras, p-código máquina o pseudo-código máquina es una especificación de una unida central de procesamiento (CPU) cuyas instrucciones se espera que sean ejecutadas en software en vez de en hardware (es decir, interpretado).

POSA: Acrónimo de Pattern Oriented Software Architecture, o Patrones Orientados a la Arquitectura de Software.

Protocolo HTTP: Protocolo de Transferencia de Hipertexto (hypertext transfer protocol). Este protocolo está diseñado para transferir lo que se llama hipertextos, páginas Web o páginas HTML.

Pruebas de Cumplimiento: Es un tipo de Prueba de Control que se diseñan para obtener una cierta seguridad de que se cumplen las disposiciones y procedimientos establecidos sobre el control interno.

Pruebas Sustantivas: Es otro tipo de Prueba de Control que consiste en las comprobaciones diseñadas para obtener la evidencia de la validez de las operaciones, propiedad de las transacciones y saldos que conforman los estados financieros

RAM: (Random Access Memory): memoria de acceso aleatorio ó memoria de acceso directo.

Ruby: Es un lenguaje de scripts para una programación orientada a objetos rápida y sencilla, es un lenguaje de programación interpretado, de muy alto nivel y orientado a objetos.

RUP: Proceso Racional Unificado (Rational Unified Process). Es un proceso de desarrollo de software. Herramienta de modelado visual para el análisis y diseño de sistemas basados en objetos.

SEI: Acrónimo de Software Engineering Institute, o Instituto de Ingeniería de Software, creado en la Universidad Carnegie Mellon, en Pittburg, Pensilvania, EU.

Servidor Web: Programa que implementa el protocolo HTTP.

SOA: Acrónimo de Service-Oriented Architecture, en español, arquitectura Orientada a Servicios.

Solaris 2X: Solaris es un sistema operativo desarrollado por Sun Microsystems. Es un sistema certificado como una versión de UNIX. Aunque Solaris en sí mismo aún es software propietario, la parte principal del sistema operativo se ha liberado como un proyecto de software libre denominado Opensolaris. Solaris puede considerarse uno de los sistemas operativos más avanzados

TCP/IP: Es el protocolo común utilizado por todos los ordenadores conectados a Internet, de manera que éstos puedan comunicarse entre sí. TCP/IP no es un único protocolo, sino que es en realidad lo que se conoce con este nombre es un conjunto de protocolos que cubren los distintos

niveles del modelo OSI. Los dos protocolos más importantes son el TCP (Transmission Control Protocol) y el IP (Internet Protocol),

UML: Lenguaje Unificado de Modelado (Unified Modeling Language). Es un lenguaje para visualizar, especificar, construir y documentar los artefactos de un sistema que involucra una gran cantidad de software. Compuesto por diversos elementos gráficos que se combinan para conformar diagramas. Cuenta con reglas para combinar tales elementos.

URL: (Uniform Resource Locator): es un localizador uniforme de recurso. Es una secuencia de caracteres, de acuerdo a un formato estándar, que se usa para nombrar recursos, como documentos e imágenes en Internet, por su localización.

WWW (World Wide Web): es un sistema de documentos de hipertexto enlazados y accesibles a través de Internet. Con un navegador Web, un usuario visualiza páginas Web que pueden contener texto, imágenes u otros contenidos multimedia.

XML: Lenguaje de Marcas Extensible (Extensible Markup Language, en inglés), es un metalenguaje extensible de etiquetas. Es una simplificación y adaptación del SGML y permite definir la gramática de lenguajes específicos (de la misma manera que HTML es a su vez un lenguaje definido por SGML). Por lo tanto XML no es realmente un lenguaje en particular, sino una manera de definir lenguajes para diferentes necesidades. Algunos de estos lenguajes que usan XML para su definición son XHTML, SVG, MathML.