Universidad de las Ciencias Informáticas

Sistema para la realización de auditorías a Sistemas Gestores de Bases de Datos.

TRABAJO DE DIPLOMA PARA OPTAR POR EL TÍTULO DE INGENIERO EN CIENCIAS INFORMÁTICAS

Autores: Ismani Rodríguez Gómez

Angel Pablo Martín Olivera.

Tutores: Ing. Wilfredo Rosales Romero.

Ing. Ramón Alexander Anglada Martínez.

La Habana, junio de 2012

"Año 54 de la Revolución"

PENSAMIENTO

Nunca consideres el estudio como una obligación, sino como una oportunidad para penetrar en el bello y maravilloso mundo del saber.

Albert Einstein.

DECLARACIÓN DE AUTORIA

Declaro que soy el único autor de este trabajo y autorizo al Centro de Telemática de la Facultad 2 de la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

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

Angel Pablo Martín Olivera
Ismani Rodríguez Gómez

_____

Wilfredo Rosales Romero
Ramón Alexander Anglada Martínez.

_____

DATOS DE CONTACTO

Tutor: Ing. Wilfredo Rosales Romero.

Institución: Universidad de las Ciencias Informáticas

Correo electrónico: wrosales@uci.cu

Título de graduado: Ingeniero en Ciencias Informáticas.

Categoría docente: Trabajador no docente.

Dirección de la institución: Carretera a San Antonio de los Baños, Km. 2 ½, Torrens, municipio: Boyeros, La Habana, Cuba.

Tutor: Ing. Ramón Alexander Anglada Martínez.

Institución: Universidad de las Ciencias Informáticas

Correo electrónico: raanglada@uci.cu

Título de graduado: Ingeniero en Ciencias Informáticas.

Categoría docente: Instructor.

Dirección de la institución: Carretera a San Antonio de los Baños, Km. 2 ½, Torrens, municipio: Boyeros, La Habana, Cuba.

AGRADECIMIENTO

A los padres que han sido la guía y el motor impulsor en la realización de nuestros sueños.

A los amigos que siempre han estado a nuestro lado, por el apoyo y la confianza brindada.

A los maestros que han sembrado en nosotros los conocimientos que hoy tenemos.

DEDICATORIA

De Angel:

A novia y a mi hijo.

A mis hermanos.

A mis padres.

A mi abuela.

A mis amigos.

A todos los que siempre me han apoyado.

De Ismani:

A mi novia.

A mis padres.

A mis hermanas.

A mi abuela.

A mis amigos.

A todos los que siempre me han apoyado.

RESUMEN

La Empresa de Telecomunicaciones de Cuba S.A. por sus siglas ETECSA, es una organización que tiene como objetivo: proveer servicios de telecomunicaciones a todo el territorio nacional. En su división empresarial cuenta con un departamento de Seguridad Informática (SI) encargado de la protección de los activos informáticos así como la realización de las auditorías a los Sistemas Gestores de Bases de Datos. Actualmente dichas auditorías se realizan de forma manual apoyándose en el uso de *scripts*[1] y documentos de texto, lo que dificulta y ralentiza el proceso. Así surge la idea de desarrollar el "Sistema para la realización de auditorías a Sistemas Gestores de Bases de Datos"(SASGBD). Este trabajo de diploma ofrece una solución que permite realizar las auditorías a Sistemas Gestores de Bases de Datos (SGBD) con el uso de tecnologías libres, se exponen los resultados del estudio realizado para el desarrollo de la aplicación, además de una serie de propuestas de recomendaciones para el mejoramiento futuro del mismo.

PALABRAS CLAVES.

Auditorías, Bases de Datos, Seguridad Informática.

[1] En informática un guión, archivo de órdenes o archivo de procesamiento por lotes**,** es un programa usualmente simple, que por lo regular se almacena en un archivo de texto plano.

ÍNDICE

ÍNDICE FIGURAS

ÍNDICE TABLAS

INTRODUCCIÓN

El avance indetenible de la informática en el nuevo siglo ha permitido que la información se convierta en un activo de altísimo valor para muchos países, que debe protegerse para garantizar su integridad, confidencialidad y disponibilidad. Del mismo modo en que la informática expande sus límites, el auge de internet acorta la distancia entre las personas, provee a los usuarios cada vez de más información, los negocios se benefician del comercio electrónico y de transacciones financieras. Pero aparejado a este desarrollo virtual crece una contra parte dedicada a falsear, robar, alterar y controlar la información almacenada por empresas, organizaciones o gobiernos. En la actualidad, dados los altos niveles de complejidad que han alcanzado los procesos tecnológicos dentro de las empresas y la importancia de los sistemas de información, ha surgido una creciente necesidad de supervisar los sistemas informáticos (1).

ETECSA, destinada a brindar servicios de telecomunicación a todo el territorio de Cuba no está exenta a estos ataques, por tanto cuenta con especialistas en SI, dedicados a velar por el resguardo de la información de la empresa y de los usuarios a los que esta les ofrece servicios. Este departamento realiza auditorías planificadas a los SGBD en todas sus entidades para velar por la integridad de los datos que se almacena en estos activos informáticos.

Los distintos gestores usados por esta empresa son auditados por el grupo de SI siguiendo buenas prácticas de usabilidad. Para cada uno de estos en busca de errores de configuración así como posibles brechas de seguridad. Estas buenas prácticas son un conjunto de investigaciones realizadas por las compañías desarrolladoras de estos SGBD, así como empresas dedicadas a la SI a nivel internacional certificadas en términos de seguridad en Bases de Datos.

ETECSA se encuentra actualmente en un desarrollo vertiginoso de su infraestructura y la aplicación de tecnologías de punta en vista de igualarse a grandes proveedores de servicios de telecomunicaciones del mundo (2). Sin embargo, la gestión de la SI en esta empresa se ve retrasada en materia de informatización de los procesos de auditoría.

El grupo de auditores informáticos de ETECSA tiene estandarizado con matrices de diagnósticos todo el proceso de revisión de los SGBD. Las herramientas que existen

actualmente para la realización de auditorías a los SGBD no se adaptan a las características de los procesos de auditoría del departamento de SI, por lo que las supervisiones se realizan ejecutando *scripts* con consultas SQL[2] a cada uno de los SGBD.

Actualmente, la comprobación de los resultados es realizado manualmente a través de matrices de diagnósticos, por tanto este proceso puede traer consigo errores humanos. El resultado final de la auditoría es un Informe General, documento que comprende los resultados de las auditorías realizadas, el cual es confeccionado manualmente a partir de las deficiencias detectadas por las matrices en la auditoría. Por otra parte todos los informes son almacenados en los archivos del departamento, por lo que los auditores, supervisores o personas interesadas en estos datos se ven obligadas a hacer grandes búsquedas a la hora de realizar reportes de auditorías. El problema anterior insta al departamento de SI de ETECSA a informatizar este proceso para incrementar la rapidez y disminuir los errores en las auditorías, con el objetivo de realizar una mejor toma de decisiones en el aseguramiento de la integridad, disponibilidad y confidencialidad de los datos almacenados.

Después del análisis de la situación anterior se formula el siguiente **problema a resolver**: *¿Cómo contribuir a la realización de las auditorías a los Sistemas Gestores de Bases de Datos de ETECSA?* Identificando como **objeto de estudio**: Los procesos de auditorías a los sistemas informáticos; y como **campo de acción**: Los procesos de auditorías a Sistemas Gestores de Bases de Datos.

Para la solución al problema existente se plantea como **objetivo general:** Desarrollar una aplicación de escritorio que permita realizar las auditorías a SGBD para el departamento de SI de ETECSA.

La **idea a defender** que guiará la presente investigación es: Con el desarrollo del SASGBD se contribuye a la realización de las auditorías a los SGBD por el departamento de SI en ETECSA.

De dicho objetivo general se derivan los siguientes **objetivos específicos**:

- Fundamentar teóricamente la necesidad del desarrollo de la aplicación.
- Diseñar e implementar una herramienta para la gestión de las auditorías.
- Definir una arquitectura que permita incorporar complementos para la evaluación de

[2] El lenguaje de consulta estructurado

resultados de auditorías.

Para dar cumplimiento a lo antes descrito se trazaron las siguientes **tareas de la investigación**:

- Definición de la situación problemática actual del departamento de SI.
- Realización de un análisis del estado del arte de las soluciones encontradas a raíz de la investigación.
- Ejecución de un levantamiento de requisitos para una propuesta del sistema.
- Selección y fundamentación de la metodología, herramientas y lenguajes de desarrollo.
- Definición de los patrones de arquitectura y estándares de diseño a utilizar para el desarrollo de una arquitectura que permita la instalación de componentes de evaluación de resultados.
- Implementación por etapas de la aplicación, dependiendo de los requisitos de mayor prioridad.
- Realización de pruebas a la aplicación.

Durante la realización de las tareas se emplearon los siguientes **métodos científicos**[3]:

Métodos teóricos:

Histórico lógico: Este método se utiliza para estudiar la teoría conocida hasta el momento, así como para conocer los antecedentes de las auditorías a los SGBD y las diferentes aplicaciones informáticas existentes en el mundo para la realización de auditorías.

Modelación: Es el método mediante el cual se crean abstracciones con el objetivo de explicar la realidad. El modelo como sustituto del objeto de investigación es semejante a él, existiendo una correspondencia objetiva entre el modelo y el objeto, siendo el investigador quien elabora dicho modelo (3). Se utilizará para modelar el proceso de negocio y los artefactos necesarios para construir el SASGBD.

Analítico – sintético: Este método permite la división mental del fenómeno en sus múltiples relaciones y componentes para facilitar su estudio, y establece mentalmente la unión entre las partes previamente analizadas. Posibilita descubrir sus características generales y las relaciones esenciales entre ellas (3). Se utilizará para captar y resumir documentos y

[3] El método científico es un método de investigación usado principalmente en la producción de conocimiento en las ciencias.

procedimientos para la realización de auditorías a SGBD y al mismo tiempo se detallará la información necesaria para el modelado correcto del negocio.

Análisis documental: Constituye un proceso ideado por el individuo como medio para organizar y representar el conocimiento registrado en los documentos, cuyo índice de producción excede sus posibilidades de lectura y captura (3). Se utilizará para el análisis y síntesis de la bibliografía consultada.

Métodos empíricos:

Entrevista: Se utilizará la entrevista como una conversación planificada con los clientes, para obtener información acerca del problema en cuestión. Su uso constituye un medio para el conocimiento cualitativo de las características particulares de un proceso (3) y puede influir en el posterior análisis y diseño del producto de *software*[4] que realizará las auditorías en el departamento de SI de ETECSA.

El presente trabajo de diploma, está estructurado en 4 capítulos. El primer capítulo: "Fundamentación Teórica", permite encontrar los principales conceptos que se manejan a lo largo del trabajo; estado del arte del tema tratado tanto internacional como nacional, así como las tendencias, técnicas, tecnologías, metodologías y software usados en la actualidad para la solución del problema a resolver. En el segundo capítulo: "Características del sistema", se realiza el análisis de la situación problemática existente, la construcción de un modelo de negocio adecuado; la especificación de los requisitos que constituirán las bases de la propuesta del sistema, así como la elaboración de los diagramas de casos de uso del sistema, descripción y prototipos de interfaces. El tercer capítulo: "Diseño del sistema", representa la base de la futura implementación del sistema; se definen los patrones de diseño y arquitectura, también se elaboran de diagramas de clases del diseño y los diagramas de interacción, específicamente los diagramas de secuencias. En el cuarto capítulo: "Implementación y Prueba", se muestra cómo será implementado el sistema en término de componentes, así como la realización de pruebas al resultado de la implementación.

[4] Se conoce como software al equipamiento lógico o soporte lógico de un sistema informático, comprende el conjunto de los componentes lógicos necesarios que hacen posible la realización de tareas específicas

CAPÍTULO 1 FUNDAMENTACIÓN TEÓRICA

1.1 Introducción

La creciente necesidad del departamento de SI de ETECSA de informatizar los procesos de gestión de auditoría ha propiciado la búsqueda de una solución capaz de sumarse al proceso de auditoría de este departamento sin que influya negativamente en el mantenimiento de la seguridad. Este capítulo centra su investigación en herramientas y soluciones similares a las requeridas por el departamento de SI, el estudio y análisis de metodologías, lenguajes y herramientas de desarrollo de *software* para la construcción de SASGBD.

1.2 Auditorías Informáticas

La auditoría informática es un proceso llevado a cabo por profesionales especialmente capacitados para el efecto, y que consiste en recoger, agrupar y evaluar evidencias para determinar si un sistema de información salvaguarda el activo empresarial, mantiene la integridad de los datos, lleva a cabo eficazmente los fines de la organización, utiliza eficientemente los recursos, y cumple con las leyes y regulaciones establecidas. Permiten detectar de forma sistemática el uso de los recursos y los flujos de información dentro de una organización y determinar qué información es crítica para el cumplimiento de su misión y objetivos, identificando necesidades, duplicidades, costes, valor y barreras, que obstaculizan los flujos de información eficientes. (4)

Auditar consiste principalmente en estudiar los mecanismos de control que están implantados en una empresa u organización, determinando si los mismos son adecuados y cumplen determinados objetivos o estrategias, estableciendo los cambios que se deberían realizar para la consecución de los mismos. Los mecanismos de control pueden ser directivos, preventivos, de detección, correctivos o de recuperación ante contingencias. (4)

1.3 Objetivos de la auditoría Informática

- El análisis de la eficiencia de los Sistemas Informáticos.
- La verificación del cumplimiento de la normativa[5] en este ámbito.
- La revisión de la eficaz gestión de los recursos informáticos. (5)

[5] Reglamento sobre la seguridad y protección de la información oficial y el modo en que se aplicarán las normas de seguridad establecidas, Resolución 1, de 26 de diciembre de 2000 en el Decreto Ley 199 del Ministerio de Interior.

1.4 Tipos de auditorías de Sistemas

Dentro de la auditoría informática destacan los siguientes tipos (5):

- **Auditoría de la gestión**: la contratación de bienes y servicios, documentación de los programas.
- **Auditoría de los datos**: Clasificación de los datos, estudio de las aplicaciones y análisis de diagramas de flujo.
- **Auditoría de las Bases de Datos**: Controles de acceso, de actualización, de integridad y calidad de los datos.
- **Auditoría de la seguridad**: Referidos a datos e información verificando disponibilidad, integridad, confidencialidad, autenticación y no repudio.
- **Auditoría de la seguridad física**: Referido a la ubicación de la organización, evitando ubicaciones de riesgo, y en algunos casos no revelando la situación física de esta. También está referida a las protecciones externas y protecciones del entorno.
- **Auditoría de la seguridad lógica**: Comprende los métodos de autenticación de los sistemas de información.
- **Auditoría de las comunicaciones**: Se refiere a la auditoría de los procesos de autenticación en los sistemas de comunicación.
- **Auditoría de la seguridad en producción**: Frente a errores, accidentes y fraudes.

Atendiendo a entrevistas hechas a los clientes se identificó que el tipo de auditoría que realizará la herramienta es la auditoría a Bases de Datos.

1.5 Principales pruebas y herramientas para efectuar una auditoría informática

En la realización de una auditoría informática el auditor puede realizar las siguientes pruebas (5):

Pruebas sustantivas: Verifican el grado de confiabilidad de la SI de los organismos. Se suelen obtener mediante observación, cálculos, muestreos, entrevistas, técnicas de examen analítico, revisiones y conciliaciones. Verifican asimismo la exactitud, integridad y validez de la información.

Pruebas de cumplimiento: Verifican el grado de cumplimiento de lo revelado mediante el análisis de muestras. Proporciona evidencias de que los controles claves existen y que son aplicables efectiva y uniformemente.

Las principales técnicas de las que dispone un auditor informático son:

- Observación.

- Realización de cuestionarios.
- Entrevistas a auditados y no auditados.
- Muestreo estadístico.
- Diagramas de flujo.
- Listas de chequeo.
- Mapas conceptuales.

Para la realización de las auditorías a Bases de Datos se realizarán pruebas de cumplimiento con la técnica Listas de chequeo.

1.6 Análisis de soluciones existentes

1.6.1 DB Audit

DB Audit Expert, es una herramienta de seguridad y de auditoría para los SGBD Oracle, SyBase, DB2, MySQL y Microsoft SQL Server. Esta herramienta permite a los administradores de seguridad o auditores realizar un seguimiento y análisis de las Bases de Datos. Entre sus principales características se encuentra la gestión de los informes de los resultados de auditorías, reduciendo en gran medida la cantidad de datos, identificando mejor las violaciones de seguridad y proporcionando mayores detalles que en los sistemas nativos (6). Entre las deficiencias que posee DB Audit está la carencia de soporte para PostgreSQL y para otros Sistemas Operativos (SO) distintos de Windows.

1.6.2 DB Watch

DB Watch, es una herramienta de seguimiento e información de sistemas. Este actúa como un sistema de vigilancia para evitar caídas en los servicios de datos. Además proporciona un estandarizado mecanismo para la presentación de los reportes y notificaciones en tiempo real por vía SMS[6] o correo electrónico. DB Watch puede ser ejecutada en cualquier sistema operativo con JRE[7] instalado. Este sistema realiza el monitoreo por medio de la ejecución de procedimientos almacenados, los cuales pueden ser gestionados por los administradores de la misma manera que se gestionan los que brinda la aplicación por defecto. (7)

1.6.3 DB Protect

DB Protect es una avanzada solución para las auditorías de seguridad y funcionamiento de los SGBD: Oracle, Microsoft SQL Server y MySQL aunque actualmente no brinda una solución para PostgreSQL. La presente herramienta mediante un análisis automático recopila los datos de los ecosistemas de

[6] SMS, en inglés es acrónimo de servicio de mensajes cortos ("*Short Message Service*"), sistema de mensajes de texto para teléfonos móviles

[7] Java Runtime Environment o JRE es un conjunto de utilidades que permite la ejecución de programas Java.

Bases de Datos de una organización para luego identificar las áreas de riesgo y las mejoras en los procesos de seguridad que sean necesarias (8). En base a este análisis DB Protect proporciona los métodos para la corrección de las vulnerabilidades detectadas.

1.6.4 AppDetectivePro

Herramienta de análisis de vulnerabilidades en Base de Datos. Es un escáner de red y evaluador de vulnerabilidades, detecta las aplicaciones de Bases de Datos existentes en la infraestructura de las empresas y evalúa sus características de seguridad. Soporta Bases de Datos: MySQL, Oracle, Sybase, IBM DB2, IBM DB2 en Mainframe, Microsoft SQL Server, Oracle Application Server y Lotus Notes/Domino (9). No consta con soporte para PostgreSQL y solamente está disponible para la plataforma Microsoft Windows.

1.6.5 Resultado del análisis

Después del análisis de algunas de las herramientas existentes, se concretó que no se adaptan a los procedimientos de auditorías por los cuales se rige el departamento de SI de ETECSA, además es notable la ausencia de soporte para PostgreSQL y para otros SO diferentes de Windows.

Estas herramientas solo brindan gratuitamente una versión de prueba con funcionalidades limitadas debido a que son patentadas bajo licencias Shareware, lo que va en contra del proceso de migración hacia *software* libre, con el propósito de fomentar la soberanía tecnológica en Cuba. Aunque las soluciones vistas sirven de referencia a la hora de desarrollar un sistema para la realización de auditorías se decidió no utilizar ninguna de las propuestas, teniendo en cuenta las desventajas que poseen.

1.7 Localizador de servicios

El Localizador de Servicios abstrae toda la utilización de JNDI[8] para ocultar las complejidades de la búsqueda y creación de objetos de forma dinámica. Este comprueba en el contexto de una aplicación los servicios solicitados en tiempo de ejecución. Para lograr la localización se define en los componentes, la interfaz del servicio que implementa y la dirección de la implementación de esta (ver Figura No. 1).

[8] La Interfaz de Nombrado y Directorio Java (Java Naming and Directory Interface) es una Interfaz de Programación de Aplicaciones (API) de Java para servicios de directorio. Permite a los clientes descubrir y buscar objetos y datos a través de un nombre.

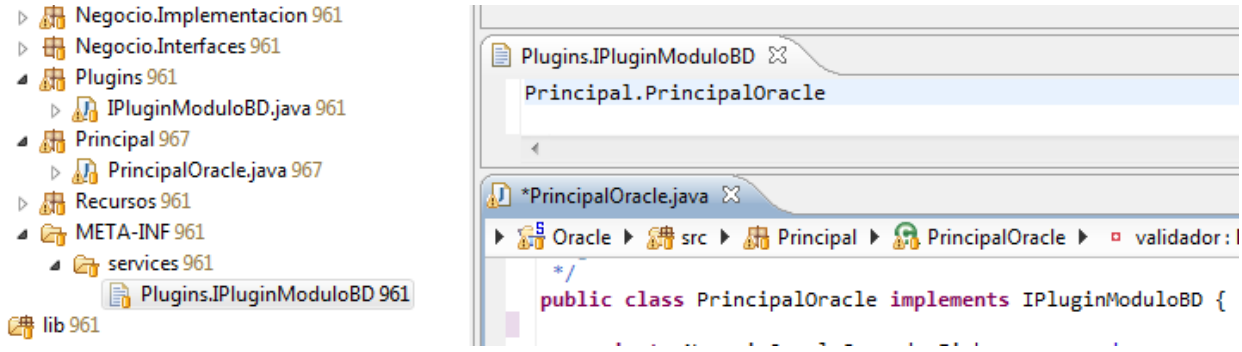


Figura No. 1 Definición del servicio en un componente

Por su parte la aplicación que solicita el servicio agrega los componentes solicitados a su contexto. Con la utilización del Localizador de Servicios es posible encontrar las implementaciones solicitadas por un sistema (ver Figura No. 2). El Localizador de Servicios será utilizado en la construcción del SASGBD como mecanismo de localización e instanciación de los componentes de evaluación de resultados de auditoría.

```
public final void LocatePlugins() throws Exception{
    ServiceLoader< IPluginModuloBD > sl =  ServiceLoader.load(IPluginModuloBD.class);
    sl.reload();
    HashMap<String, IPluginModuloBD> lista_componentes = new HashMap<String, IPluginModuloBD>();
    for (Iterator< IPluginModuloBD  > it = sl.iterator(); it.hasNext();) {
        try {
            IPluginModuloBD pl = it.next();
            lista_componentes.put(pl.getGestor(), pl);
        } catch (Exception ex) {
            throw ex;
        }
    }
}
```

Figura No. 2 Localización del servicio

1.8 Lenguaje de modelado UML

Lenguaje de Modelado Unificado (UML, por sus siglas en inglés, Unified Modeling Language). Es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema de *software* (10). Es uno de los lenguajes de modelado de sistemas de *software* más conocidos y utilizados en la actualidad puesto que sirve para el modelado completo de sistemas complejos. Se puede aplicar en una gran variedad de formas para dar soporte a una metodología de desarrollo de *software* (Ej.: RUP[9]), pero no especifica en sí mismo qué metodología o proceso usar.

Otra ventaja de este lenguaje de modelado es su independencia del lenguaje de implementación, de tal forma que los diseños realizados usando UML se puedan implementar en cualquier lenguaje que

[9] Proceso Unificado de Desarrollo de Software.

soporte las posibilidades de UML (Lenguajes Orientados a Objetos). UML aporta mayor rigor en la especificación, permite automatizar determinados procesos, generar código a partir de los modelos y a la inversa (a partir del código fuente generar los modelos), esto permite que el modelo y el código estén actualizados. (10)

Teniendo en cuenta que se utiliza para el modelado completo del sistema, brinda soporte a la metodología utilizada (RUP), es independiente del lenguaje de programación y brinda rigor en la especificación, se selecciona como lenguaje de modelado a UML.

1.9 Notación para el modelado del negocio BPMN

Para el modelado de negocio pueden utilizarse técnicas y notaciones. Esto permite conocer los objetivos del negocio y plasmarlos en un modelo. El objetivo principal de BPMN es proporcionar una notación que sea fácilmente comprensible por todos los usuarios del negocio, desde los analistas de negocio que crean los borradores iniciales de los procesos, a los desarrolladores técnicos responsables de la aplicación de la tecnología que llevará a cabo los procesos y, finalmente, al personal de negocios que administran y supervisan los procesos. Un Modelo de Procesos de Negocio, es una red de objetos gráficos, que son actividades y los controles de flujo que definen su orden de funcionamiento. BPMN define un diagrama de procesos de negocio (BPD), que se basa en una técnica de diagramas de flujo para la creación de modelos gráficos de las operaciones de proceso de negocio. BPMN es independiente de cualquier metodología de modelado de procesos y permite modelar los procesos de una manera unificada y estandarizada permitiendo un entendimiento al personal de una organización (11), por lo tanto es seleccionado como notación para el modelado del negocio del sistema.

1.10 Metodología y herramientas Cases

1.10.1 Rational Unified Process (RUP)

Las metodologías de desarrollo son un conjunto de procedimientos, técnicas y normas que ayudan al proceso de desarrollo de *software* así como a su documentación. Existen diferentes metodologías para hacer un *software*, y cada una se adapta a determinada situación, sin embargo, para proyectos grandes y complejos muchos autores recomiendan el uso de RUP (12).

Se decide utilizar el Proceso Unificado de Desarrollo como metodología para el desarrollo de esta herramienta, por ser esta metodología más que un simple proceso, es un marco de trabajo que puede ser usado para desarrollar una gran variedad de aplicaciones informáticas para diferentes áreas de aplicación, tipos de organización, niveles de aptitud y tamaños de proyecto. Además guía el diseño,

implementación y prueba, a través de casos de uso, convirtiéndose en uno de los elementos fundamentales dentro del proceso de desarrollo. Es centrada en la arquitectura al exigir que esta deba estar diseñada para que el sistema evolucione en sus fases de desarrollo y por último permite dividir el trabajo en varias iteraciones o mini proyectos los cuales representan un incremento del producto, permitiendo a proyectos muy complejos y de larga duración, la entrega de pequeñas versiones, para ir satisfaciendo paulatinamente las necesidades de los clientes.

1.10.2 Herramienta Case Visual Paradigm

Las herramientas CASE proporcionan un conjunto de herramientas semi-automatizadas y automatizadas que brindan ayuda y asistencia a los analistas, ingenieros de *software* y desarrolladores, durante todos los pasos del ciclo de vida de desarrollo de un *software*. Visual Paradigm para UML es una herramienta profesional que soporta el ciclo de vida completo del desarrollo de *software*. Brinda la posibilidad de crear un conjunto amplio de artefactos utilizados con mucha frecuencia durante la confección de un *software*, cumpliendo con el Standard UML 2.0 (11).

Para el modelado de la solución se utilizará Visual Paradigm (versión 8.0), porque permite desarrollar la mayoría de los artefactos que genera cada flujo de trabajo de RUP.

1.11 Lenguaje de programación Java.

Java es un ambiente completo para producir aplicaciones distribuidas, a efecto de que los programas de Java se ejecuten en forma segura e idéntica en diversos tipos de computadoras y plataformas. (13)

Entre sus características fundamentales se encuentran: (13)

- Multiplataforma: Java es multiplataforma, es decir, el código generado por un compilador Java, conocido como *byte codes*, puede ser ejecutado en gran variedad de tipos de computadores sin recompilar ni cambiar un solo byte. Los *byte codes* deben ser interpretados por una máquina virtual (JVM[10]). Esta máquina virtual es una aplicación que debe existir en toda plataforma en donde se desee ejecutar una aplicación Java. Actualmente la mayoría de los SO poseen una implementación oficial de la JVM (13).
- Lenguaje poderoso y moderno: La sintaxis del Java está basada en C++[11]. Aunque no se puedan manejar los punteros directamente, se puede hacer uso de referencias que brindan casi la misma funcionalidad. Además posee un colector de basura que permite gestionar la memoria

[10] Una **máquina virtual Java** (en inglés *Java Virtual Machine*, **JVM**) es una máquina virtual de proceso nativo, es decir, ejecutable en una plataforma específica, capaz de interpretar y ejecutar instrucciones expresadas en un código binario especial (el byte code Java), el cual es generado por el compilador del lenguaje Java

[11] **C++** es un lenguaje de programación diseñado a mediados de los años 1980 por Bjarne Stroustrup. La intención de su creación fue el extender al exitoso lenguaje de programación C con mecanismos que permitan la manipulación de objetos. En ese sentido, desde el punto de vista de los lenguajes orientados a objetos, el C++ es un lenguaje híbrido.

más fácil (13).

Java es seleccionado como lenguaje de programación de esta herramienta, además, porque cuenta con las características para garantizar la portabilidad del sistema no solo a otros SO sino también a un futuro ambiente web.

1.12 Marco de trabajo *Spring Framework*[12]

Spring Framework (también conocido simplemente como *Spring*) es un *framework* de código abierto de desarrollo de aplicaciones para la plataforma Java (14).

A pesar de que *Spring Framework* no obliga a usar un modelo de programación en particular, se ha popularizado en la comunidad de programadores en Java considerársele como una alternativa, y sustituto, del modelo de Enterprise JavaBean[13] (EJB). Por su diseño el *framework* ofrece mucha libertad a los desarrolladores en Java y soluciones muy bien documentadas y fáciles de usar para las prácticas comunes en la industria (14). Entre las principales características de *Spring* están:

- La inyección de dependencias: *Spring* promueve un bajo acoplamiento a través de una técnica conocida como la inyección de dependencias. Cuando se aplica la inyección de dependencias, al crear los objetos se les pasan a esos sus respectivas dependencias, en lugar de ellos mismos buscar de los objetos de los cuales dependen (14).
- Contenedor: *Spring* es un contenedor en el sentido de que contiene y administra el ciclo de vida y la configuración de los objetos de aplicación. En *Spring*, se puede declarar cómo cada uno de los objetos de su aplicación debe ser creado, la forma en que debe ser configurado, y cómo deben estar asociado con cada otro (14).
- *Framework*: *Spring* permite configurar y componer complejas solicitudes desde componentes más simples. *Spring* también ofrece muchas funcionalidades de infraestructuras (gestión de transacciones, *framework* de integración de persistencia[14]), dejando el desarrollo de la lógica de la aplicación a los desarrolladores (14).

Spring (versión 3.0.6) es seleccionado además porque es totalmente compatible con ambientes de escritorio o web sin realizar ningún cambio en la aplicación, requisito de la aplicación para garantizar su portabilidad.

---

[12] La palabra inglesa "**framework**" define, en términos generales, un conjunto estandarizado de conceptos, prácticas y criterios para enfocar un tipo de problemática particular, que sirve como referencia para enfrentar y resolver nuevos problemas de índole similar.

[13] Los **Enterprise JavaBeans** (también conocidos por sus siglas **EJB**) son una de las API que forman parte del estándar de construcción de aplicaciones empresariales J2EE(ahora JEE 6.0) de Oracle Corporation

[14] Persistencia en informática de modo genérico, se refiere a la propiedad de los datos para que estos sobrevivan de alguna manera. De forma sencilla puede entenderse que los datos tienen una duración efímera, desde el momento en que estos cambian de valor se considera que no hay persistencia de los mismos. Sin embargo en informática hay varios ámbitos donde se aplica y se entiende la persistencia.

1.13 IDE de desarrollo *Spring Source Tool Suite*

Spring Source Tool Suite ™ (STS) es un entorno de desarrollo para la construcción de aplicaciones empresariales con *Spring*. STS proporciona herramientas para muchas de las tecnologías basadas en Java, *Spring*, y *Grails*[15].*Spring STS* permite la instalación de *plugins* por ser basado en Eclipse, estos aceleran el proceso de desarrollo de *software*, por lo que es posible extenderle funcionalidades para la integración como el control de versiones (*Subversive SVN Team Provider Plugin*), un asistente para el mapeo y generación de código instantáneo de *Hibernate*[16] (*Hibernate Core Plugin*), y un plugin para realizar reportes de pruebas unitarias con *JUnit* de manera visual (*JUnit Testing Framework*), los cuales son de utilidad para el desarrollo del SASGBD Es seleccionado este IDE[17](versión 2.8.1) porque provee herramientas por defecto que aceleran el desarrollo con el marco de trabajo *Spring* como son: un asistente para la creación de proyectos *Spring*, archivos XML[18] y *Beans*[19], un editor gráfico para la configuración de *Spring* y la comprobación de sintaxis y de chequeo de tipos en los archivos de configuración de *Spring*.

1.14 Marco de trabajo para la realización de pruebas unitarias *JUnit*.

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. (15) 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. El propio *framework* incluye formas de ver los resultados que pueden ser en modo texto, gráfico (AWT[20] o Swing[21]) o como tarea en Ant[22].

En la actualidad las herramientas de desarrollo NetBeans[23] y Eclipse[24] cuentan con *plugins* que permiten que la generación de las plantillas necesarias para la creación de las pruebas de una clase

15 *Grails* es un *framework* para aplicaciones web libre desarrollado sobre el lenguaje de programación *Groovy* (el cual a su vez se basa en la plataforma Java).

16 *Hibernate* es una herramienta de Mapeo objeto-relacional (ORM) para la plataforma Java (y disponible también para .Net con el nombre de *NHibernate*).

17 Un entorno de desarrollo integrado, llamado también IDE (sigla en inglés de *integrated development environment*), es un programa informático compuesto por un conjunto de herramientas de programación.

18 XML, siglas en inglés de eXtensible Markup Language ('lenguaje de marcas extensible'), es un lenguaje de marcas desarrollado por el World Wide Web Consortium (W3C).

19 Componente software de Java que tiene la particularidad de ser reutilizable y así evitar la tediosa tarea de programar los distintos componentes uno a uno.

20 Abstract Window Toolkit (AWT, en español Kit de Herramientas de Ventana Abstracta) es un kit de herramientas de gráficos, interfaz de usuario, y sistema de ventanas independiente de la plataforma original de Java.

21 Swing es una biblioteca gráfica para Java. Incluye widgets para interfaz gráfica de usuario tales como cajas de texto, botones, desplegables y tablas.

22 Apache Ant es una herramienta usada en programación para la realización de tareas mecánicas y repetitivas, normalmente durante la fase de compilación y construcción.

23 NetBeans es un entorno de desarrollo integrado libre, hecho principalmente para el lenguaje de programación Java.

Java se realice de manera automática, facilitando al programador enfocarse en la prueba y el resultado esperado, y dejando a la herramienta la creación de las clases que permiten coordinar las pruebas. JUnit (versión 4.8.2) guiará las pruebas unitarias del SASGBD por ser un *framework* de código abierto que es nativo para Java y la existencia de *plugins* para el IDE elegido que facilitan la realización de las pruebas al producto a desarrollar.

1.15 Herramientas para la comprobación de métricas de calidad de *software*.

La medida de la calidad real de un software no es automatizable. La mayoría de los programas de software tienen problemas, por lo que se deben tomar medidas correctoras; sin embargo que no se encuentren errores no implica que sea de alta calidad, sino que no se han encontrado aquellos que se sabe que producen problemas (16). Existen muchas herramientas para la medición de la calidad del software, para la evaluación del resultado de la construcción del SASGBD se utilizarán las siguientes herramientas:

- ***FindBugs*:** es un programa de tipo código abierto que busca errores en programas escritos en código Java. Utiliza análisis estático para identificar cientos de tipos de errores potenciales en programas Java. *FindBugs* (versión 2.0.0) es una herramienta que analiza el código y no el programa en ejecución en busca de estructuras potencialmente peligrosas tales como: posibles bugs, código "muerto" (variables no accedidas, bloques de ejecución inalcanzables), código no óptimo, bloques con una estructura poco legible o más complicada de lo necesario (16). El software se distribuye como una aplicación de escritorio, pero además existe disponible un complemento para el IDE seleccionado para la construcción del SASGBD.

- ***Checkstyle*:** En el mundo Java, Sun proporcionó unas guías de estilo que han sido ampliamente adoptadas por la comunidad, actualmente son mantenidas por Oracle las cuales tienen como objetivo estandarizar la codificación en lenguaje Java (17). Es una herramienta que genera informes del nivel de seguimiento de estas convenciones. Checkstyle es una herramienta para ayudar a los programadores a escribir código Java según estándares de codificación.

1.16 Marco de trabajo para el mapeo objeto-relacional *Hibernate*.

Hibernate se centra en la solución al problema de la diferencia entre los dos modelos de datos coexistentes en una aplicación: el usado en la memoria de la computadora (orientación a objetos) y el usado en las Bases de Datos (modelo relacional). Para esto permite al desarrollador detallar su

[24] Eclipse es un entorno de desarrollo integrado de código abierto multiplataforma para desarrollar lo que el proyecto llama "Aplicaciones de Cliente Enriquecido", opuesto a las aplicaciones "Cliente-liviano" basadas en navegadores.

modelo de datos, las relaciones y sus propiedades. Con esta información *Hibernate* permite a la aplicación manipular los datos en la Base de Datos operando sobre objetos, con las características de la programación Orienta a Objetos. Este convierte los datos entre los tipos utilizados por Java y los definidos por SQL. Además genera las sentencias SQL y libera al desarrollador del manejo manual de los datos que resultan de la ejecución de dichas sentencias, manteniendo la portabilidad entre todos los motores de Bases de Datos con un ligero incremento en el tiempo de ejecución (18).

Hibernate está diseñado para ser flexible en cuanto al esquema de tablas utilizado, para poder adaptarse a su uso sobre una Base de Datos ya existente. También tiene la funcionalidad de crear la Base de Datos a partir de la información disponible. Ofrece también un lenguaje de consulta de datos llamado HQL[25], al mismo tiempo que una API[26] para construir las consultas programáticamente (conocida como *criteria*) (18). *Hibernate* (versión 3.0.5) es la herramienta elegida para el mapeo de la Base de Datos de SASGBD, además de las características antes expuestas porque para la generación de las clases persistentes y los archivos de mapeo existe un *plugin* para el IDE elegido, lo que acelera el proceso de desarrollo del *software*.

1.17 Herramienta para el almacenamiento de los datos PostgreSQL

PostgreSQL es un SGBD relacional Orientado a Objetos, publicado bajo la licencia BSD[27]. Como muchos otros proyectos de código abierto, el desarrollo de PostgreSQL no es manejado por una empresa o persona, sino que es dirigido por una comunidad de desarrolladores. Dicha comunidad es denominada el PostgreSQL Global Development Group[28]. (19)

PostgreSQL cuenta, entre otras, con las siguientes características:

- Aproxima los datos a un modelo objeto-relacional y es capaz de manejar complejas rutinas y reglas.
- Contiene consultas SQL declarativas, control de concurrencia, soporte multiusuario, optimización de consultas, herencia y arreglos.
- Es un SGBD altamente extensible ya que soporta operadores, funcionales métodos de acceso y tipos de datos definidos por el usuario.
- Soporta integridad referencial, la cual es utilizada para garantizar la validez de los datos.

25 En ingles: Hibernate Query Language.

26 Interfaz de programación de aplicaciones o API (del inglés Application Programming Interface) es el conjunto de funciones y procedimientos (o métodos, en la programación orientada a objetos) que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.

27 La licencia BSD es la licencia de software otorgada principalmente para los sistemas BSD (*Berkeley Software Distribution*). Es una licencia de software libre permisiva como la licencia de *Open SSL* o la *MIT License*. Esta licencia tiene menos restricciones en comparación con otras como la GPL estando muy cercana al dominio público. La licencia BSD al contrario que la GPL permite el uso del código fuente en software no libre.

28 Grupo Global de desarrollo de PostgreSQL.

- Basa su funcionamiento en una arquitectura proceso por usuario y cliente/servidor[29].

PostgreSQL (versión 9.1) es seleccionado para el almacenamiento de los datos de la aplicación porque es uno de los SGBD más avanzados y de *software* libre.

1.18 Conclusiones

En este capítulo se analizaron algunas de las herramientas existentes en el mundo para la realización de auditorías a SGBD las cuales no representan una solución para el departamento de SI de ETECSA porque lo que es necesario el desarrollo del SASGBD, además se decidió la utilización de la metodología RUP para el desarrollo de software para lograr la continuidad del desarrollo de la aplicación. Java fue el lenguaje de programación seleccionado para poder desarrollar una herramienta multiplataforma. Se decidió también la utilización de *JUnit* como *framework* para la realización de pruebas unitarias, *CheckStyle* y *FindBugs* serán las herramientas para la comprobación de la calidad del código para que la aplicación logre aceptables niveles de calidad y de cumplimiento de los estándares de codificación definidos por la Universidad. Estudiado este capítulo se deja orientada la guía de desarrollo del trabajo y sentadas las bases para un posterior análisis de la propuesta del sistema.

29 La arquitectura cliente-servidor es un modelo de aplicación distribuida en el que las tareas se reparten entre los proveedores de recursos o servicios, llamados servidores, y los demandantes, llamados clientes. Un cliente realiza peticiones a otro programa, el servidor, que le da respuesta. Esta idea también se puede aplicar a programas que se ejecutan sobre una sola computadora, aunque es más ventajosa en un sistema operativo multiusuario distribuido a través de una red de computadoras.

CAPÍTULO 2 CARACTERÍSTICAS DEL SISTEMA

2.1 Introducción

En este capítulo se realiza el modelamiento del negocio y la especificación de cada uno de los procesos y los actores participantes en el negocio. Además se detallan los requisitos funcionales y no funcionales. Se modela también el diagrama de CU del sistema y se describen cada uno de estos.

2.2 Modelo de negocio

El modelo de negocio describe los procesos de una empresa en términos de casos de uso y actores del negocio (ver Tabla 1) que se corresponden con los procesos (ver Tabla 2) y los clientes, respectivamente (20). En la Figura No. 3 se muestra la descripción del modelo de negocio para las auditorías a SGBD.

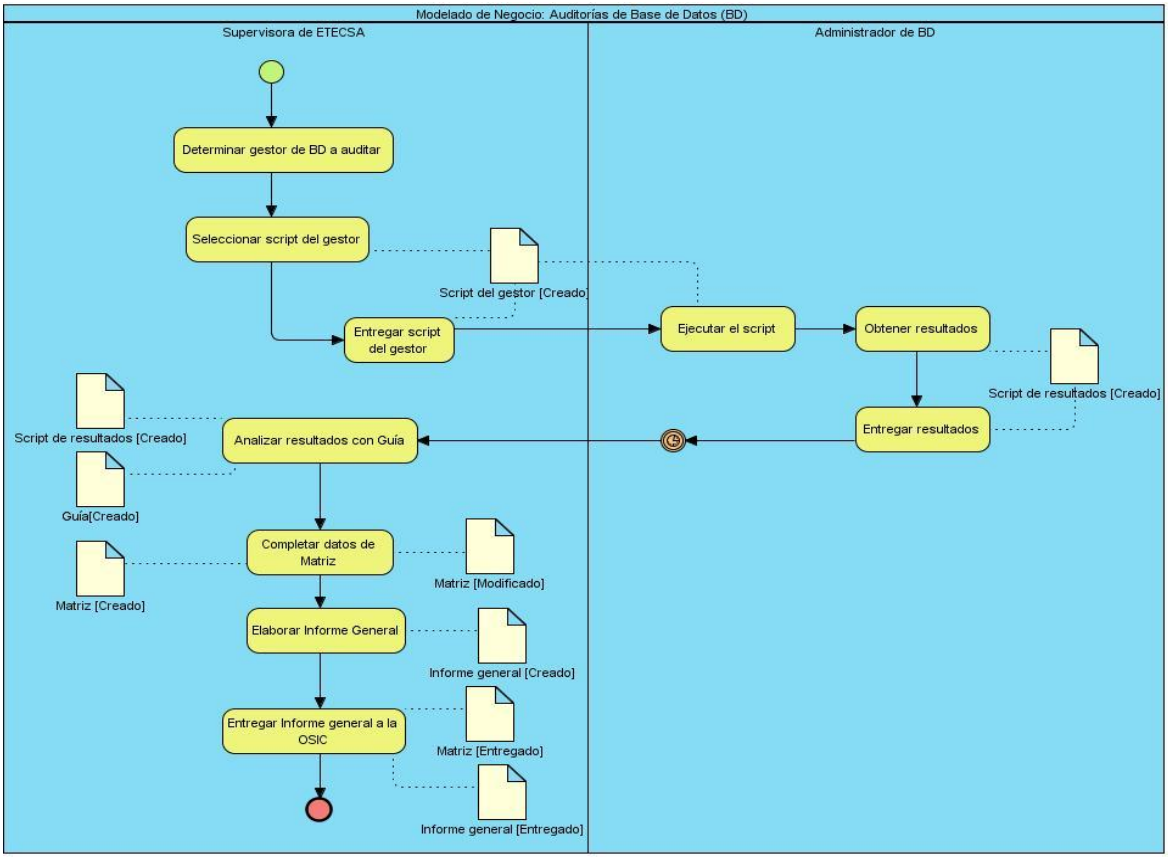


Figura No. 3 Descripción del proceso del negocio

Actor	Objetivo
Supervisor	Realiza supervisiones a los SGBD que se encuentran hospedados en los servidores de ETECSA y confecciona el Informe General de auditoría.
Administrador de BD	Ejecuta el *script* para cada SGBD a auditar y obtiene el *script* de resultados.

Tabla 1 Actores del negocio

No	Actividades	Descripción	Responsable	Entrada	Salida
1	Determinar SGBD a auditar	El negocio inicia cuando se determina por la planificación anual, realizar una supervisión a uno o varios SGBD instalados.	Supervisor de ETECSA.	-	SGBD seleccionado
2	Seleccionar *script* del gestor	Una vez seleccionado el SGBD se procede a seleccionar el *script* correspondiente para realizar la supervisión.	Supervisor de ETECSA.	SGBD seleccionado	*Script* del gestor.
3	Entregar *script* del gestor	El supervisor le entrega personalmente el *script* al administrador de la BD a supervisar.	Supervisor de ETECSA.	*Script* del gestor.	*Script* del gestor.
4	Ejecutar el *script*	El administrador ejecuta el *script* correspondiente a la BD a auditar.	Administrador de BD.	SGBD seleccionado	*Script* del gestor.

5	Obtener resultados	El administrador una vez que ejecuta el *script*, obtiene los resultados de la supervisión en otro *script*.	Administrador de BD.	*Script* del gestor.	*Script* de resultados.
6	Entregar resultados.	El administrador entrega el *script* de resultados para su posterior análisis.	Administrador de BD.	*Script* de resultados.	*Script* de resultados.
7	Analizar resultados con Guía	Se analizan los resultados con la Guía la cual permite valorar los parámetros del *script* resultante.	Supervisor de ETECSA.	*Script* de resultado, Guía.	
8	Completar datos de Matriz	Una vez que se analicen los datos del *script* resultante se procede a completar la Matriz la cual realiza una descripción de todas las vulnerabilidades detectadas.	Supervisor de ETECSA.	Matriz	Matriz
9	Elaborar Informe general	Se elabora un Informe general que sintetiza los resultados de la supervisión.	Supervisor de ETECSA.	Matriz	Informe general
10	Entregar el	Se entrega el Informe general y Matriz a la	Supervisor de	-	Informe general,

	Informe general	OSIC[30] para su revisión, archivo y distribución a las personas indicadas.	ETECSA.		Matriz.

Tabla 2 Descripción del flujo básico

2.3 Relación de los requerimientos

Posiblemente la etapa más compleja a lo largo de la producción de un software sea la de captura de requerimientos. En esta etapa el cliente determina qué características y funcionalidades desea que contenga el software a desarrollar. Para lograr esto se hace necesario el establecimiento de una visión común entre el cliente y el equipo de desarrollo, sobre los procesos que se llevan a cabo en el negocio y cómo estos se verán reflejados y mejorados en el futuro software. Esta tarea es generalmente complicada ya que la mayoría de los clientes no dominan las tecnologías informáticas. Esta es una etapa de vital importancia puesto que de ella depende que los resultados sean los esperados y que el cliente quede satisfecho con el producto. (10)

2.3.1 Listado de los requerimientos funcionales

Un requisito se define como una condición o capacidad que el sistema debe ser capaz de realizar, sin tomar en consideración ningún tipo de restricción física. Esta definición puede extenderse y ser aplicada a las condiciones que debe cumplir o poseer un sistema o uno de sus componentes para satisfacer un contrato, una norma o una especificación (10). Los requerimientos funcionales que se identificaron para el desarrollo del módulo fueron los siguientes:

No.	Requisito.	Descripción	No.	Sub-requisito.
1.	Gestionar consulta.	Permite insertar, modificar, eliminar y listar las consultas en la BD.	1	Insertar consulta.
			2	Modificar consulta.
			3	Eliminar consulta.
			4	Mostrar consulta.
			5	Mostrar información de consulta.

[30] Oficina secreta de información confidencial.

2.	Gestionar indicador.	Permite insertar, modificar, eliminar y listar los indicadores en la BD.	1	Insertar indicador.
			2	Eliminar indicador.
			3	Listar indicadores.
			4	Mostrar información de indicador.
			5	Modificar indicador.
3.	Generar *script*.	Permite al supervisor seleccionar los valores para conformar un *script* de auditoría: un gestor, una versión, y una lista consultas. Exporta en formato XML el *script* generado por el supervisor.		
4.	Importar fichero de resultado.	Importa un fichero XML, el cual es evaluado por el *plugin* que sea compatible con los datos que posee el fichero de resultados.		
5.	Cargar configuración.	Carga la configuración de un gestor desde un fichero XML y la inserta en la BD.		
6.	Instalar *plugin*.	Copia o elimina un *plugin* en la carpeta *Plugins* de la aplicación. Al instalar un *plugin* carga la configuración inicial de este.	1	Instalar *plugin*.
			2	Desinstalar *plugin*.

7.	Listar *plugin*.	Muestra una lista con los *plugins* instalados en la aplicación.		
8.	Exportar configuración del gestor.	Exporta en un fichero XML todos los datos relacionados de un gestor determinado.		
9.	Mostrar información de *plugin*.	Muestra la información de un *plugin* determinado.		
10.	Generar informe general.	Almacena en la BD un Informe General de auditoría a partir de una matriz de resultados seleccionada.		
11.	Exportar Informe General.	Exporta un Informe de auditoría a un fichero con formato PDF.		
12.	Exportar matriz.	Exporta una matriz de resultados a un fichero con formato PDF.		
13.	Buscar matriz.	Busca una matriz en la BD atendiendo a su identificador, fecha de confección o nivel de riesgo.		
14.	Activar consulta.	Permite a una consulta ser utilizada en el proceso de creación de *scripts* de auditorías.	1	Activar consulta
			2	Desactivar consulta.

15.	Activar indicador.	Permite a un indicador y sus consultas asociadas a ser utilizados en el proceso de crear *scripts* de auditorías.	1	Activar indicador.
			2	Desactivar indicador.

Tabla 3 Listado de los requerimientos funcionales

2.3.2 Definición de los requerimientos no funcionales

Los requerimientos no funcionales especifican propiedades del sistema, como restricciones del entorno o de la implementación, rendimiento, dependencias de la plataforma, facilidad de mantenimiento, extensibilidad, y fiabilidad. En muchos casos dichos requisitos son fundamentales en el éxito del producto (10). Los requerimientos no funcionales para el desarrollo de la aplicación son:

No.	Requisito	Descripción.
1.	Usabilidad	Podrá ser usado por cualquier persona que posea conocimientos básicos sobre informática y las auditorías a SGBD.
2.	Soporte	La aplicación debe contar con un manual de usuario.
3.	Portabilidad	El sistema debe ser multiplataforma, pudiéndose ejecutar sobre Microsoft Windows o GNU/Linux.
4.	Legales	SASGBD se desarrollará utilizando herramientas de *software* libre, la propiedad intelectual del producto será propio de la UCI, desarrollado para el cliente que es ETECSA, específicamente el departamento de Gestión de la SI.
5.	Hardware	El sistema debe desplegarse en una computadora con un espacio disponible en disco de 30 MB.
6.	*Software*	Como requisito fundamental para la ejecución del sistema los ordenadores deberán tener instalada la

		máquina virtual de java (JVM) versión 1.6.
7.	Seguridad	Para el aseguramiento de la seguridad se plantean los siguientes requisitos a tener en cuenta por la aplicación: • Auditoría de seguridad: Se registrarán todos los eventos que realizan los supervisores en la BD. • Autodefensa: Se validarán todos los datos de entrada. • Identificación y autenticación: Se empleará el uso de usuario y contraseña para la validación de la identidad de los usuarios del sistema. • Gestión de la seguridad: Se definirán perfiles de usuarios y niveles de acceso de cada uno de los perfiles.

Tabla 4 Listado de los requerimientos no funcionales

2.4 Descripción del sistema

El sistema a desarrollar será una aplicación de escritorio porque será utilizada por un personal reducido en el departamento de SI, por lo cual no es necesario habilitar un sistema web si solamente existirá un punto único de utilización del producto: el departamento de SI de ETECSA.

La aplicación soportará un sistema de *plugins* que lo harán flexible a la hora de agregarle soporte a nuevos SGBD. Al iniciar la aplicación el Servicio de Localización busca los *plugins* en la carpeta "*Plugins*" de la estructura de carpetas de la herramienta, y verifica su compatibilidad con el sistema. En el momento de la instalación de un *plugin* nuevo el sistema almacena en la BD toda la configuración inicial de este. La información que se inserta consta de un gestor y sus versiones, una lista de indicadores y de consultas, los cuales el *plugin* trae consigo por defecto para la realización de auditorías a ese gestor. Los *plugins* además pueden ser desinstalados, pero esto no produce la eliminación de los datos del gestor para el cual se instaló.

La gestión de las consultas es otra funcionalidad del sistema. Estas consultas pueden ser adicionadas a un gestor y para varias versiones pero es necesaria la existencia de un *plugin* instalado que valide la sintaxis de la sentencia SQL de la consulta a insertar. La búsqueda del *plugin* al insertar cada consulta se realiza en base al nombre del gestor y las versiones seleccionadas. Para la modificación al igual que en la adición es necesario de un *plugin* para la validación de la sentencia SQL. Al eliminar una consulta se propuso como restricción que no existan valores encontrados (resultado que devuelve una consulta después de ser ejecutado el *script* de consultas) asociados a esta consulta para preservar la integridad de los datos de auditoría. Las consultas serán mostradas en un árbol, el cual cuenta con 3 niveles (Gestores, Indicadores, Consultas), cada consulta será hija del Indicador al cual está asociada.

El *script* de consultas para ejecutar la auditoría por la "Herramienta Colaborativa para la realización de auditorías a SGBD" será exportado en un fichero XML como consecuencia de la selección del gestor, la versión y las consultas que serán utilizadas. El resultado devuelto por la auditoría es cargado en formato XML y será evaluado por un *plugin* según el gestor y la versión que el fichero tiene en sus propiedades. El *plugin* arroja una matriz de resultados con los valores encontrados en la auditoría y una evaluación para cada uno de estos. El proceso de evaluación de la matriz es calculada mediante un método estadístico aprobado por el cliente, el cual tiene en cuenta la evaluación de cada resultado encontrado y el peso de la consulta asociada a esta. Según el proceso de realización de auditorías en el departamento de SI de ETECSA, la evaluación de los resultados de una matriz de auditoría puede ser editado a consideración del supervisor, al realizar alguna modificación a estos resultados es recalculada la evaluación de la auditoría.

Los Informes Generales son los reportes finales de la realización de una auditoría, estos son creados a partir de una matriz de auditoría y son registrados en la BD y también es posible exportar en formato PDF los datos del informe.

2.5 Diagrama de Casos de Uso del Sistema

El Modelo de Casos de Uso del Sistema permite que los desarrolladores de *software* y los clientes lleguen a un acuerdo sobre los requisitos, es decir sobre las condiciones y posibilidades que debe cumplir el sistema. Además proporciona la entrada fundamental para el análisis, diseño, implementación y pruebas (10). La Figura No. 4 muestra el diagrama de CU del Sistema y en la Tabla 5 se describen los actores que intervienen en el sistema.

Actores del sistema	Justificación
Supervisor	Realiza supervisiones a los gestores de Bases de Datos que se encuentran hospedados en los servidores de ETECSA y confecciona el informe general de auditoría.

Tabla 5 Actores del sistema

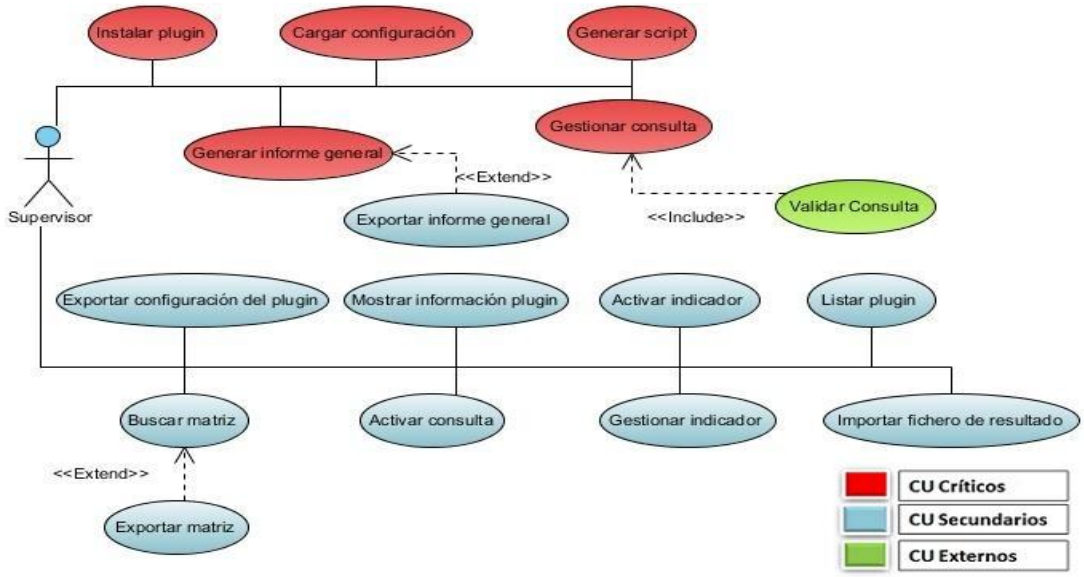


Figura No. 4 Diagrama de Casos de Uso del Sistema

2.5.1 Priorización de los Casos de Uso

Los casos de usos fueron priorizados según su importancia para el funcionamiento de la aplicación a continuación se presenta el resultado de esta clasificación:

CU	Ciclo	Justificación
CU 1 Gestionar consultas. **CU 6 Instalar *plugin*.** **CU 3 Generar *script*.** **CU 5 Cargar configuración.**	**1**	**Representan los casos de uso que responden a las funcionalidades vitales de la aplicación.**

CU 10 Generar informe general.		
CU 4 Importar fichero de resultado. **CU 2 Gestionar indicador.** **CU 7 Listar *plugin*.** **CU 8 Exportar configuración *plugin*.** **CU 9 Mostrar información de *plugin*.** **CU 11 Exportar Informe general.** **CU 12 Exportar matriz.** **CU 13 Buscar matriz.** **CU 14 Activar consulta.** **CU 15 Activar indicador.**	**2**	**Representan los casos de uso que sirven de apoyo a la aplicación.**

Tabla 6 Priorización de los Casos de Uso

2.6 Descripción de los Casos de Uso

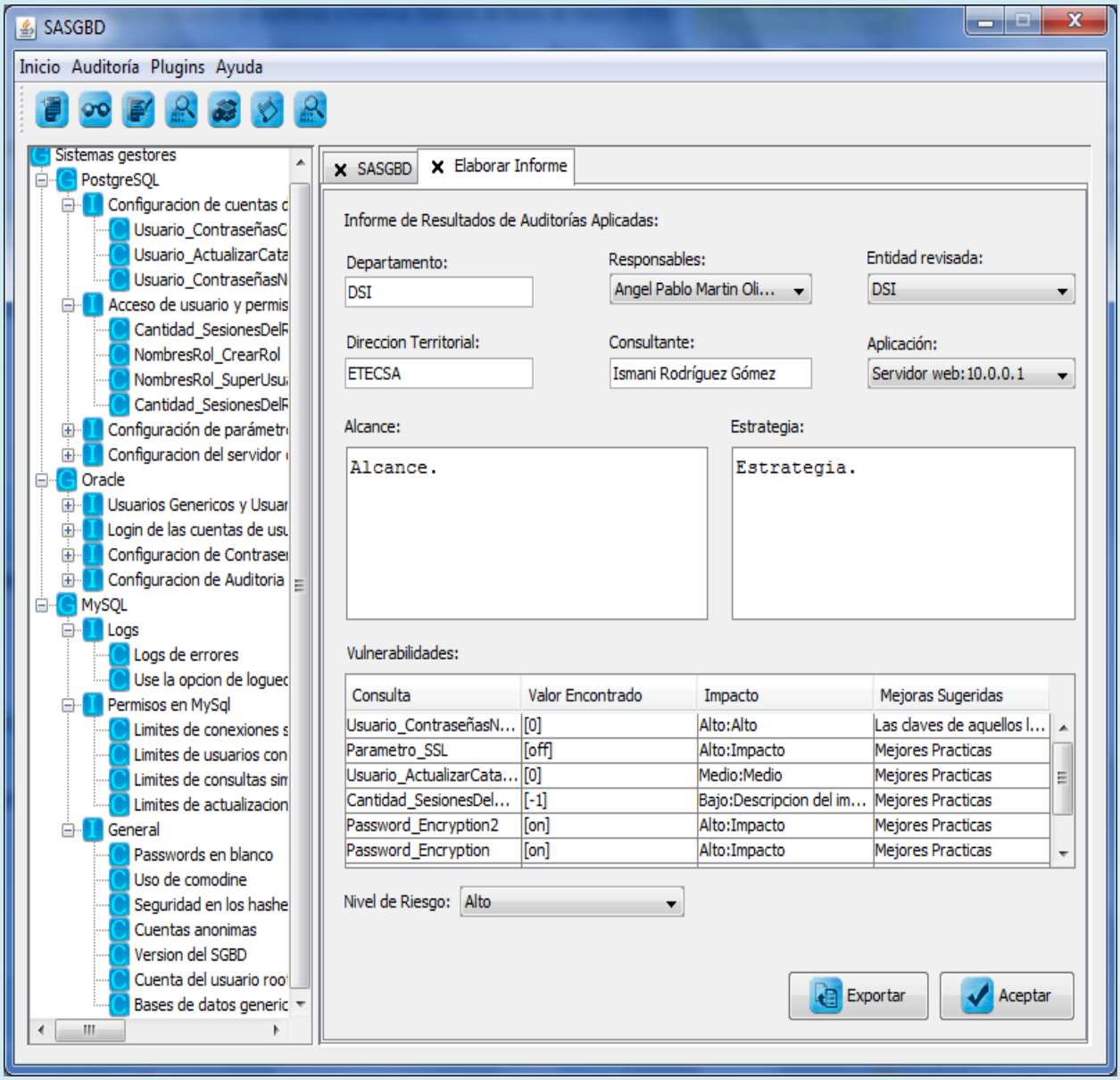
El objetivo principal de detallar los casos de uso de un sistema es describir su flujo de sucesos en detalle, incluyendo cómo inicia, termina e interactúan con los actores. En la siguiente tabla se muestra la descripción del CU Generar Informe General, las descripciones de los demás CU pueden ser vistas en los Anexos.

DCU Generar Informe General

Objetivo	Generar el Informe General de la auditoría.
Actores	Supervisor
Resumen	Se genera el Informe con los datos de la auditoría y se almacena en la Base de Datos.
Complejidad	Media.

Prioridad	Crítico.
Precondiciones	El supervisor debe seleccionar una matriz de resultado.
Postcondiciones	Se almacenó en la Base de Datos el Informe generado.
Flujo de eventos	
Flujo básico Generar Informe General	

Prototipo Interfaz



	Actor	Sistema
1.	Selecciona la matriz para generar el Informe.	**2.** Carga todos los datos relacionados con la matriz de resultados en una plantilla de Informe General.
3.	Llena todos los campos vacíos del Informe.	
4.	Presiona el botón Aceptar.	**5.** Valida que los datos del informe estén correctos.
		6. Se almacena el informe en la Base de Datos.
		7. Muestra el mensaje: “Informe generado correctamente”.
8.	Presiona el botón Aceptar.	**9.** Cierra la pestaña Generar informe.
		10. Termina el CU
Flujos alternos		
4ª Exportar informe.		
	Actor	**Sistema**
	4ª1 Presiona el botón Exportar.	**4ª2** Exporta el informe (Ver CU Exportar Informe General**).**
		4ª3 Termina el CU.

Relaciones	**CU Incluidos**	No procede
	CU Extendidos	CU Exportar Informe General en paso 4ª2 del flujo alterno Exportar informe.
Requisitos no funcionales	No procede	
Asuntos pendientes	No procede	

Tabla 7 DCU Generar Informe General

2.7 Conclusiones

El desarrollo del capítulo contribuyó a lograr una mejor comprensión del sistema que se desea construir. La modelación y descripción del proceso de negocio del departamento de SI demostró que no es un proceso genérico si no que consta de características a las cuales los requisitos funcionales tienen el objetivo de cumplir. Se decidió el desarrollo de una aplicación de escritorio debido a las características del departamento de SI: cuentan con poco personal y la aplicación será utilizada solamente por el departamento de SI de ETECSA. A partir de este momento es posible comenzar el diseñó del sistema.

3.1 Introducción

Partiendo de los artefactos generados en el capítulo anterior es posible comenzar a realizar el diseño de la aplicación, en este capítulo se expone el diseño de la arquitectura, se abordarán brevemente los patrones de diseño y arquitectura a utilizar. Se presentan además los diagramas de clases del diseño, de interacción, de componentes y de datos.

3.2 Arquitectura del sistema

3.2.1 Estilo arquitectónico

El diseño está estrechamente ligado a la arquitectura que se utiliza para el desarrollo. Siendo la arquitectura multicapas la seleccionada para la elaboración del SASGBD, la cual posee ventajas en la descentralización del sistema, organización del modelo de diseño en capas y facilidad en la detección y corrección de errores, además de ofrecer una mayor transparencia y entendimiento del sistema. La arquitectura en capas reduce las dependencias, lo cual se refleja en que los componentes de una capa sólo pueden hacer referencia a componentes en capas inmediatamente inferior o superior. La siguiente figura muestra como quedó estructurada la arquitectura del sistema.

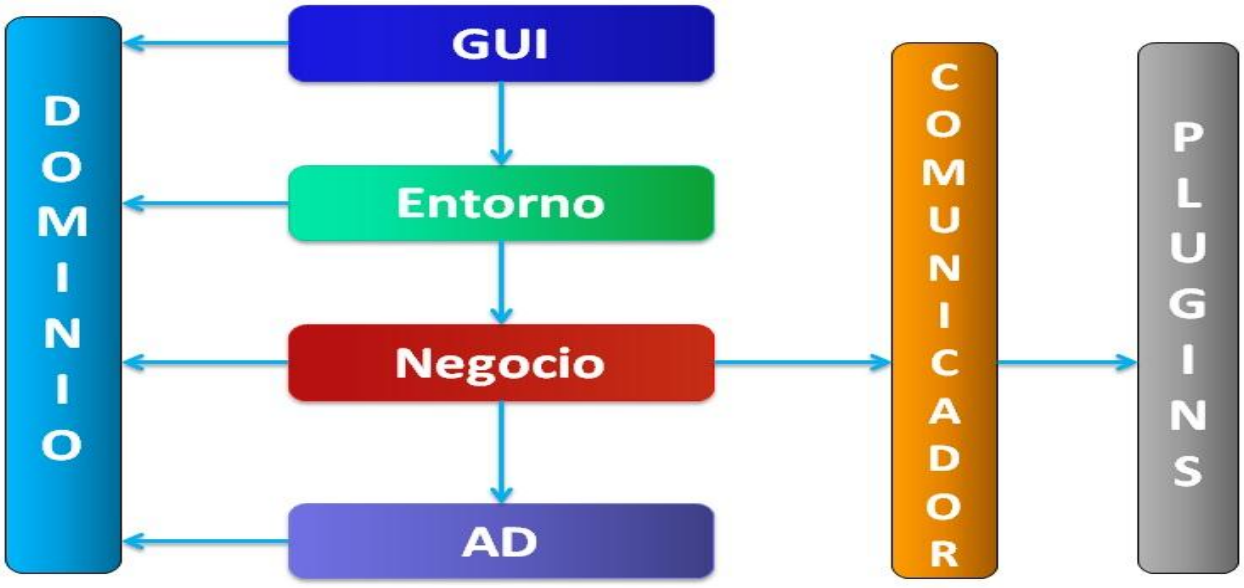


Figura No. 5 Arquitectura del Sistema

La capa **GUI** es la capa que contendrá las interfaces gráficas mediante las cuales los usuarios harán uso de la aplicación. La misma tendrá una carpeta UI que almacenará las interfaces gráficas, una segunda llamada Implementación donde estarán las implementaciones para controlar los eventos de las interfaces gráficas.

La capa **Entorno**, es una capa de abstracción entre la capa GUI y la capa de Negocio, es la encargada de establecer la comunicación entre ambas capas. Es el contenedor de los datos de las vistas de la aplicación.

La capa de **Negocio** contendrá toda la lógica de negocio, es quien implementa las funciones de la aplicación. La misma tiene una carpeta Interfaces que es donde estarán las clases interfaces que definirán el comportamiento de las clases del negocio. Tendrá además una carpeta Implementación donde estarán las clases que implementarán las interfaces del negocio. Esta capa se comunica con la capa de Acceso a Datos (AD).

La capa **AD** (Acceso a Datos) manejará todo lo relacionado con el acceso a los datos, es una capa que abstrae al negocio del modo en que la aplicación se conectará a la Base de Datos, esto permitirá que la Base de Datos pueda cambiar en un futuro y no haya que hacer muchos cambios en el negocio. Esta capa también contendrá una carpeta Interfaces donde estarán las clases de interfaces correspondientes y una carpeta Implementación que tendrá las clases que implementarán las interfaces.

La capa **Comunicador**, es una capa vertical que tendrá la interfaz de comportamiento que deberá ser implementada por cada uno de los *plugins*.

La capa ***Plugins***, es una capa vertical que tendrá dentro los *plugins* para el análisis de los resultados de los diferentes SGBD, los cuales serán accedidos a través de la capa Comunicador.

3.2.2 Patrones de arquitectura y diseño

El diseño de *software* es tanto un proceso como un modelo. El proceso de diseño es una secuencia de pasos que hacen posible que el diseñador describa todos los aspectos del *software* que se va construir. Sin embargo, es importante destacar que el proceso de diseño simplemente no es un recetario. Un conocimiento creativo, experiencia en el tema, un sentido de lo que hace que un *software* sea bueno y un compromiso general con la calidad son factores críticos de éxito para un diseño competente. Comienza representando la totalidad de todo lo que se va a construir. (21)

3.2.2.1 Patrones GRASP usados:

GRASP es un acrónimo de *General Responsibility Assignment Software Patterns* (patrones generales de *software* para asignar responsabilidades). El nombre se eligió para sugerir la importancia de aprehender (*grasping* en inglés) estos principios para diseñar con éxito el *software* Orientado a Objetos. (22)

- **Experto en Información**: El principio mediante el que se asignó cada responsabilidad fue el Experto en Información, colocando estas responsabilidades en el objeto que tiene la información necesaria para realizarlas (22). Es posible ver este patrón en la aplicación en la capa de Negocio donde cada interfaz realiza solamente las funcionalidades que debe tener, en la Interfaz *IGestionarFichero* solamente están las funciones de trabajo con ficheros.

```
public interface INegocioGestionarFichero {

    public void GenerarScript(List<TbDconsulta> listaconsulta,
            TbDversion version, TbNgestor gestor, String direccion)
            throws Exception;

    public void ExportarInformeGeneral(TbDinforme informe, String consultante)
            throws Exception;

    public void ExportarMatriz(TbDmatrizResultado matriz, String direccion)
            throws IOException, DocumentException;

    public void ExportarConfiguracionPlugin(TbNgestor gestor, String direccion)
            throws Exception;
}
```

Figura No. 6 Uso del patrón Experto en información

- **Creador:** Consiste en asignarle la responsabilidad de crear una instancia de una clase a aquella que agrega o contiene los objetos de la que se desea crear, registra las instancias de la misma, utiliza los objetos de ella, tiene los objetos de inicialización de ella cuando este objeto sea creado (22). Este patrón guía la asignación de responsabilidades relacionadas con la creación de objetos. Se brinda soporte a un bajo acoplamiento lo cual supone menos dependencias respecto al mantenimiento y mejores oportunidades de reutilización. La clase ImpUIPrincipal es la encargada de crear las instancias de las demás clases de interfaz gráfica con las que interactúa el usuario.

```
if (accion.compareTo("Importar Configuracion")==0) {
    IUIImportarConfiguracion importarconfig = new ImpUIImportarConfiguracion();
    JScrollPane scroll = new JScrollPane();
    scroll.setViewportView(importarconfig.mostrar());
    this.jTabbedPane0.addTab("Listar Matriz", scroll);
    this.jTabbedPane0.setSelectedIndex(this.jTabbedPane0.getTabCount()-1);
}
if (accion.compareTo("Buscar Matriz")==0) {
    IUIBuscarMatriz matriz = new ImpUIBuscarMatriz();
    JScrollPane scroll = new JScrollPane();
    scroll.setViewportView(matriz.mostrar());
    this.jTabbedPane0.addTab("Buscar Matriz", scroll);
    this.jTabbedPane0.setSelectedIndex(this.jTabbedPane0.getTabCount()-1);
}
if (accion.compareTo("Modificar Matriz")==0) {
    IUIModificarMatrizResultado modificarmatriz = new ImpUIModificarMatrizResultado();
    modificarmatriz.setMatrizResultado((TbDmatrizResultado) objeto);
    JScrollPane scroll = new JScrollPane();
    scroll.setViewportView(modificarmatriz.mostrar());
    this.jTabbedPane0.addTab("Editar Matriz", scroll);
    this.jTabbedPane0.setSelectedIndex(this.jTabbedPane0.getTabCount()-1);
}
```

Figura No. 7 Uso del patrón Creador

- **Bajo Acoplamiento:** El acoplamiento es una medida de la fuerza de como una clase está conectada a otras, como las conoce y como recurre a ellas. Una clase con alto (o fuerte) acoplamiento recurre a muchas otras. Esto provoca que sean más difíciles de entender cuando están aisladas y son más difíciles de reutilizar porque se requiere la presencia de otras clases de las que dependen (22). *Spring* plantea el uso de interfaces para evitar un alto acoplamiento, por lo que en el sistema este patrón se ve claramente reflejado, en las capas de negocio y acceso a datos no existe dependencia a las implementaciones de sus atributos si no a las interfaces que definen el comportamiento de estos.

- **Alta Cohesión**: La cohesión es una medida de cuán relacionadas y enfocadas están las responsabilidades de una clase. Una alta cohesión caracteriza a las clases con responsabilidades estrechamente relacionadas que no realicen un trabajo enorme. Una clase con baja cohesión las hace difíciles de comprender, conservar y reutilizar, además de que las afectan constantemente los cambios. Es un principio que debe estar presente durante el diseño, es un patrón evaluativo que se aplica al juzgar decisiones de diseño (22). Anteriormente se representó el uso del patrón Experto, el cual distribuye el comportamiento entre las clases que cuentan con la información, alentando con ello definiciones de clases sencillas y más cohesivas que son más fáciles de comprender y mantener. Así se brinda soporte a la Alta Cohesión.

- **Controlador**: Un controlador es un objeto de interfaz no destinada al usuario que se encarga de manejar un evento del sistema. Define además el método de su operación. El hecho de delegar a un controlador la responsabilidad de la operación de un sistema entre las clases del dominio soporta la reutilización de la lógica para manejar los procesos afines del negocio en aplicaciones futuras (22). Este patrón es utilizado al manejar todos los eventos de las interfaces en implementaciones separadas de la vista, logrando la reutilización de las interfaces gráficas en más de una funcionalidad.

```
public class ImpUIPlugin extends FrmPlugin implements IUIPlugin{

    EPlugin entorno;

    @Override
    public void actionPerformed(ActionEvent arg0) {
        if(arg0.getSource()==this.jbrefrescar){
            refresh();
        }
        if(arg0.getSource()==this.jbdesintalar){
            if(entorno.DesintalarPlugin(this.jTable0.getSelectedRow()))
            {
                JOptionPane.showMessageDialog(null, "El plugin será desinstalado en el próximo inicio de la
            }
            else{
                JOptionPane.showMessageDialog(null, "Plugin no desintalado","Mensaje", 0);
            }
        }

    }
```

Figura No. 8 Uso del patrón Controlador

3.2.2.2 Patrones GOF[31] usados:

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 (22).

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 reutilizable, lo que significa que es aplicable a diferentes problemas de diseño en distintas circunstancias (22). Los patrones usados en la aplicación son los siguientes:

- ***Facade* o Fachada**: Proporciona una interfaz unificada de alto nivel que agrupa las funcionalidades de un subsistema facilitando su uso. Simplifica el acceso a un conjunto de clases debido que el cliente se comunica con ellas a través de una sola interfaz. La fachada sabe que clase es responsable de cada petición, pero es completamente

[31] El grupo Gang of Four (GoF) escritores del libro Design Patterns compuesto por Erich Gamma, Richard Helm, Ralph Johnson y John Vlisides.

transparente para ellas (23). La utilización de este patrón se evidencia en el paquete Acceso a Datos donde en cada clase del negocio que necesite una implementación del acceso a datos lo hace a través de su interfaz IADModuloBD<T>.

```
import Dominio.TbNnivelEvaluacion;
import Dominio.TbRversionConsulta;
import Dominio.TbRversionConsultaId;
import Negocio.Interfaces.INegocioGestionarConsulta;
import AD.Interfaces.IADModuloBD;

public class NegocioGestionarConsulta implements INegocioGestionarConsulta {


    public IADModuloBD<TbDconsulta> adconsulta;
    public IADModuloBD<TbDformaEvaluar> adformaevaluar;
    public IADModuloBD<TbDvalorEncontrado> advalorencontrado;
    public IADModuloBD<TbRversionConsulta> adversionconsulta;
    public IADModuloBD<TbDfeIntervalo> adfeintervalo;
    public IADModuloBD<TbDfeLista> adfelista;
    public IADModuloBD<TbDfeVariante> adfevariante;
    public IADModuloBD<TbDvalorPredeterminado> advalorpredeterminado;
    public IADModuloBD<TbDintervalo> adintervalo;
    public IADModuloBD<TbDvariante> advariante;
    public IADModuloBD<TbDvalorVariante> advalorvariante;
    public IADModuloBD<TbDvalorVariante> adversion;
    public IADModuloBD<TbNgestor> adgestor;
```

Figura No. 9 Uso del patrón Fachada

- **Inversión de control (IoC) Inyección de dependencias**: La inversión de control es un principio que permite asignar la responsabilidad del control en la creación de objetos a un agente externo como un contenedor, en lugar de hacerlo las clases de la aplicación. *Spring* proporciona un contenedor que maneja el ciclo de vida de los objetos y las dependencias entre ellos. Mientras que la inversión de control es un principio de diseño general, la inyección de dependencias es un patrón de diseño concreto que rige este principio. El patrón inyección de dependencias consiste en inyectar objetos a una clase en vez de hacerlo la misma clase (24). Este patrón se implementa mediante un contenedor que inyecta a cada objeto los objetos necesarios según las relaciones plasmadas en un fichero de configuración y típicamente es implementado por un *framework* externo a la aplicación como *Spring*.

```
<bean id="negocioconsulta" class="Negocio.Implementacion.NegocioGestionarConsulta">
    <property name="adconsulta" ref="adconsulta"></property>
    <property name="adformaevaluar" ref="adformaevaluar"></property>
    <property name="advalorencontrado" ref="advalorencontrado"></property>
    <property name="adversionconsulta" ref="adversionconsulta"></property>
    <property name="adfeintervalo" ref="adfeintervalo"></property>
    <property name="adfelista" ref="adfelista"></property>
    <property name="adfevariante" ref="adfevariante"></property>
    <property name="advalorpredeterminado" ref="advalorpredeterminado"></property>
    <property name="adintervalo" ref="adintervalo"></property>
    <property name="advariante" ref="advariante"></property>
    <property name="advalorvariante" ref="advalorvariante"></property>
    <property name="adversion" ref="adversion"></property>
    <property name="adgestor" ref="adgestor"></property>    </bean>
    <bean id="negocioindicador" class="Negocio.Implementacion.NegocioGestionarIndicador">
    <property name="adindicador" ref="adindicador"></property>
</bean>
```

Figura No. 10 Inversión del Control

- **Singleton**: Garantiza la existencia de una única instancia para una clase y la creación de un mecanismo de acceso global a dicha instancia. Este patrón es utilizado en la clase ContextoApp la cual garantiza la utilización de un único contexto en la aplicación.

```
public class ContextoApp {

    String context="context.xml";
    ApplicationContext app;
    private static ContextoApp principal;

    private ContextoApp() {

    public static ContextoApp getInstace(){
        if (principal == null) {
            principal = new ContextoApp();
            }
            return principal;
    }

    public Object getBean(String nombre) {

    public void CargarContexto() {


}
```

Figura No. 11 Uso del patrón Singleton

- **Localizador de Servicios**: La idea básica detrás del Localizador de Servicios es tener un objeto que sabe cómo encontrar todos los servicios que una aplicación pueda necesitar; o sea los *plugins* del SASGBD que serán los encargados de realizar las evaluaciones de los resultados de auditorías a los SGBD.

```
public final void LoadPlugins() throws Exception{
    ServiceLoader< IPluginModuloBD > sl =  ServiceLoader.load(IPluginModuloBD.class);
    sl.reload();
    plugins = new HashMap<String, IPluginModuloBD>();
    for (Iterator< IPluginModuloBD  > it = sl.iterator(); it.hasNext();) {
        try {
            IPluginModuloBD pl = it.next();
            plugins.put(pl.getGestor(), pl);
        } catch (Exception ex) {
            throw ex;
        }
    }
}
```

Figura No. 12 Uso del patrón Localizador de Servicios

- **Modelo-Vista-Controlador:** el patrón Modelo-Vista-Controlador (MVC) separa la lógica de negocio de la interfaz de usuario, lo cual posibilita la evolución de estos dos aspectos por separado. Incrementa la reutilización y la flexibilidad. Este patrón propone el uso de un modelo (clases del paquete *Entorno*), muchas vistas (clases del paquete UI del paquete *GUI*) y muchos controladores (clases del paquete *Implementacion* del paquete *GUI*). El MVC en Swing tiene como Vista a las clases de interfaz gráfica las cuales heredan de *java.awt.Component*, el Controlador es una clase dedicada a capturar los eventos que sean lanzados por una vista, el Modelo se encarga de preparar los datos a mostrar en las vistas y a llamar los métodos del Negocio.

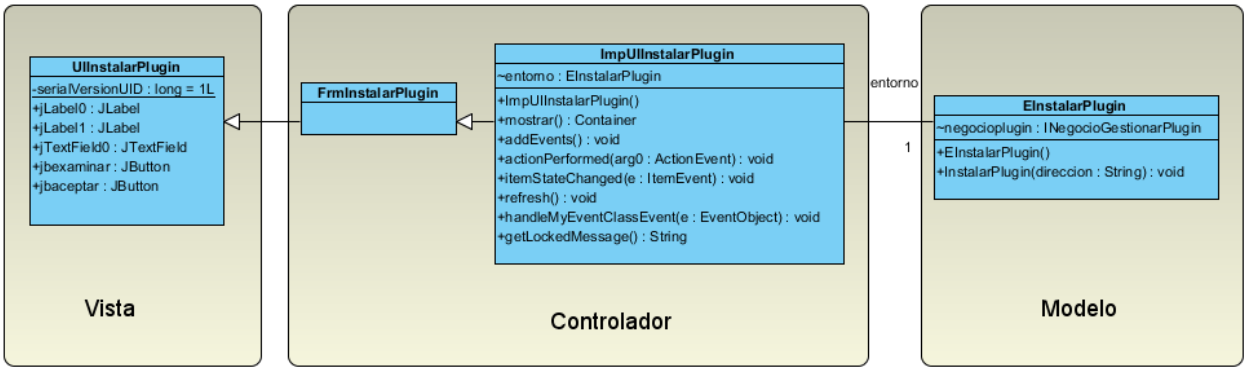


Figura No. 13 Uso del patrón MVC.

3.3 Diagrama de Paquetes

En ocasiones se hace necesario agrupar por paquetes aquellas funcionalidades semejantes, teniendo en cuenta las características del modelo, ya que brinda una mejor organización. En el caso del Sistema para la realización de auditorías a SGBD la representación queda de la siguiente forma:

- Administrar auditoría: Se agruparán las funcionalidades para la gestión de matrices de resultado e informes de auditorías.
- Administrar *Plugins*: Se agruparán las funcionalidades de administración de *plugins* y la comunicación de estos con el módulo.
- Administrar Consultas: Se agruparán las funcionalidades de gestión de consultas e indicadores.
- *Plugins*: Se agruparán las implementaciones de los *plugins* que cargará la aplicación para la evaluación de las auditorías.

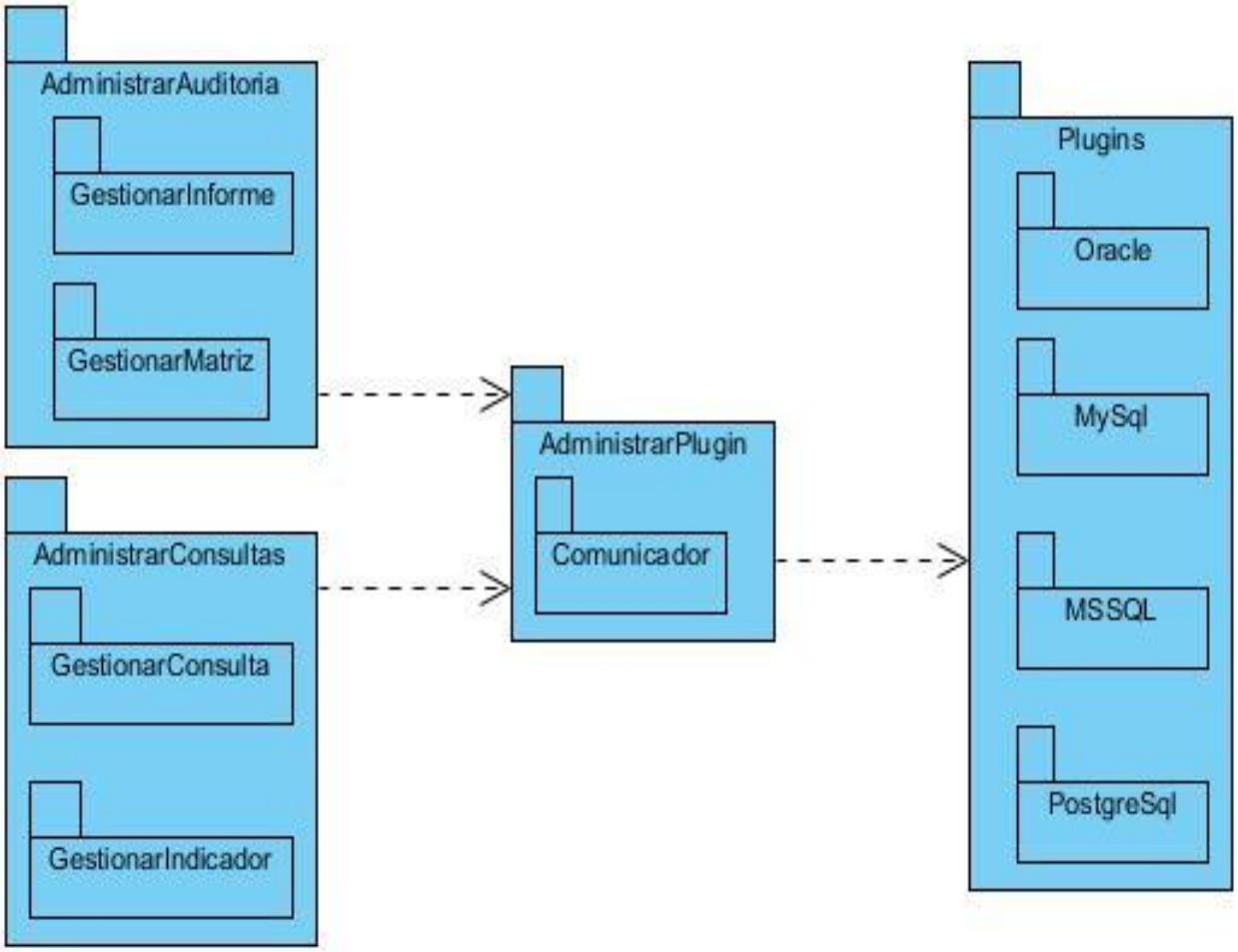


Figura No. 14 Diagrama de Paquetes

3.4 Diagramas de clases del diseño

Los diagramas de clases son diagramas de estructura estática que muestran las clases del sistema y sus interrelaciones. Los diagramas de clase son el pilar básico del modelado con

UML, siendo utilizados tanto para mostrar lo que el sistema puede hacer (análisis), como para mostrar cómo puede ser construido (diseño) (10). Para una mejor comprensión de los diagramas de clases del diseño se diferenciaron las capas por colores.

Figura No. 15 Leyenda

DCD Generar Informe General

En la siguiente figura se muestra el diagrama de clases del diseño correspondiente al CU Generar Informe Generar. Los demás diagramas se encuentran en los Anexos.

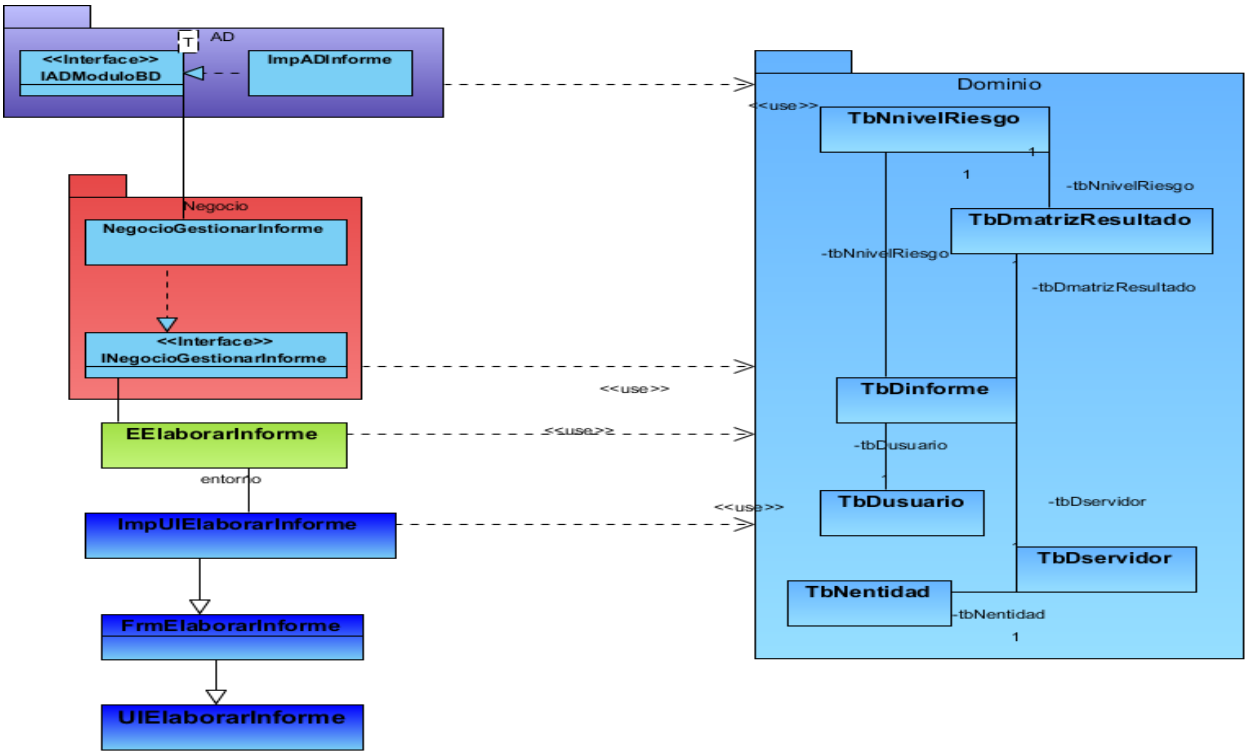


Figura No. 16 DCD Generar Informe General

3.5 Diagramas de interacción

Un diagrama de secuencia (DS) muestra la interacción de un conjunto de objetos en una aplicación a través del tiempo y se modela para cada caso de uso. Mientras que el diagrama de casos de uso permite el modelado de una vista del negocio, el diagrama de secuencia contiene detalles de implementación del escenario, incluyendo los objetos y clases que se usan para implementar el escenario, y mensajes intercambiados entre los objetos (10). A continuación se presentan el diagrama de secuencia corresponde al CU Generar Informe General los demás diagramas se encuentran en los Anexos.

DS Generar Informe General

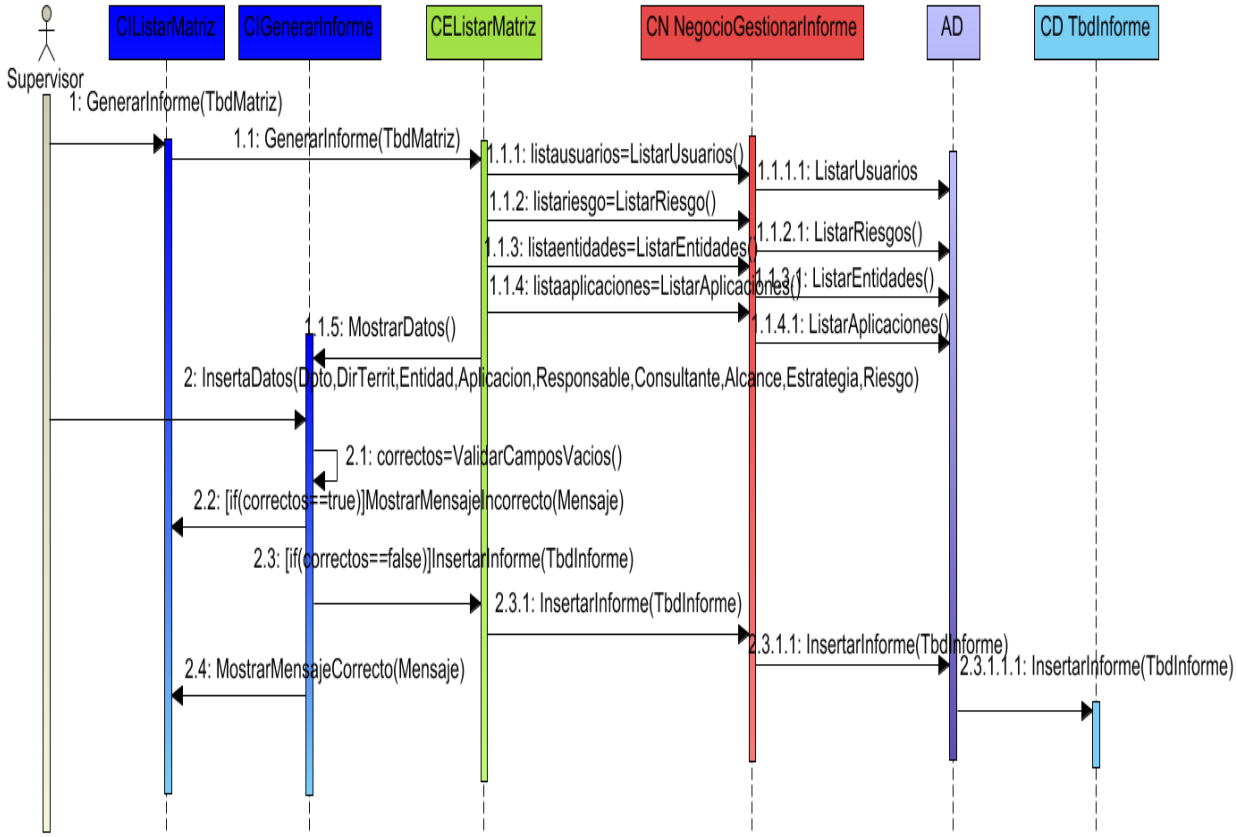


Figura No. 17 DS Generar Informe General

3.6 Diseño de la Bases de Datos

El Modelo de Datos describe las representaciones lógicas y físicas de datos persistentes utilizados por una aplicación. El modelo físico de datos se desarrolló a partir de la base del conjunto de clases persistentes y sus asociaciones en el modelo de diseño (10). La figura 18 muestra las entidades del SASGBD y las relaciones entre ellas, en la tabla 8 se describen las tablas utilizadas en la aplicación y sus atributos.

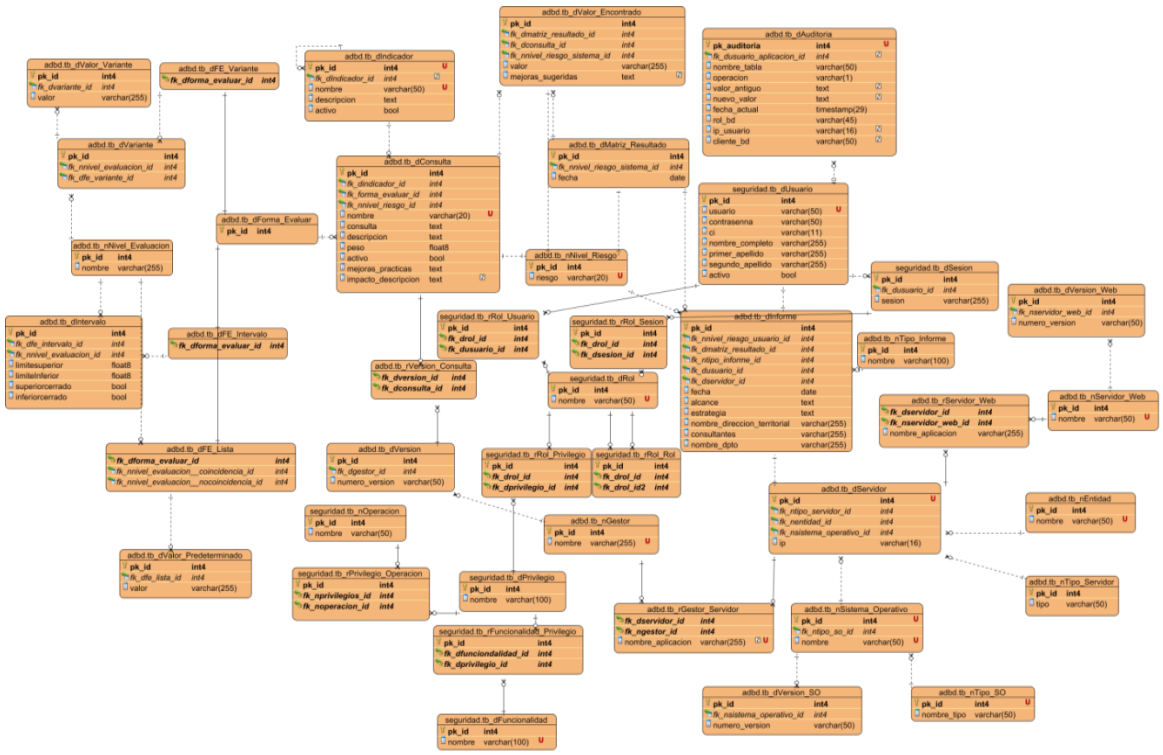


Figura No. 18 Diseño de la Bases de Datos

Nombre: adbd.tb_dversion				
Descripción: **Almacena todas la versiones de cada gestor.**				
Nombre:	Tipo de Dato:	PK/FK	¿Nulo?	**Descripción:**
pk_id	Int4	PK	No	**Identificador de la tabla adbd.tb_dversion.**
numero_version	Char(20)		No	**Almacena la versión de cada gestor.**
fk_dgestor_id	Int4	FK	No	**Identificador de la tabla adbd.tb_dGestor**
Nombre: adbd.tb_dIndicador				
Descripción: **Almacena los datos de los indicadores que se utilizan para clasificar las consultas.**				
Nombre:	Tipo de Dato:	PK/FK	¿Nulo?	**Descripción:**
pk_id	Int4	PK	No	**Identificador de la tabla adbd.tb_dIndicador.**
fk_dversion_id	Int4	FK	No	**Identificador de la tabla Versión.**
fk_dindicador_id	Int4	FK	No	**Un indicador puede pertenecer a**

				otro indicador.
nombre	Char(20)		No	**Almacena el nombre del Indicador.**
peso	Int4		No	**Almacena el peso de cada indicador.**
descripcion	text			**Almacena una pequeña descripción del Indicador.**
estado	bool		No	**Almacena el estado en que se encuentra el Indicador.**
Nombre: adbd.tb_dconsulta				
Descripción: **Almacena los datos de las consultas que se les van a agregar al *script* que se genera.**				
Nombre:	Tipo de Dato:	PK/FK	¿Nulo?	**Descripción:**
pk_id	Int4	PK	No	**Identificador de la tabla adbd.tb_dConsulta.**
fk_dindicador_id	Int4	FK	No	**Identificador de la tabla adbd.tb_dIndicador.**
nombre	Char(20)		No	**Almacena el nombre de la consulta.**
consulta	text		No	**Almacena las sentencias sql de las consulta que se le harán a los gestores.**
descripcion	text			**Almacena una pequeña descripción de la consulta.**
peso	Int4		No	**Almacena el peso de cada consulta.**
fk_forma_evaluiar_id	Int4	fk	No	**Identificador de la tabla adbd.tb_dforma_evaluar.**
fk_nnivel_riesgo_id	Int4	Fk	No	**Identificador de la tabla adbd.tb_ nnivel_riesgo.**
activo	Bool		No	**Almacena el estado de la consulta.**
mejoras_practicas	Text		No	**Almacena las mejores prácticas que utiliza la consulta.**
Impacto_descripcion	Text		No	**Almacena el impacto que tienen las consultas.**

Nombre: adbd.tb_dInforme				
Descripción: **Almacena los datos de los informes generados por la aplicación.**				
Nombre:	Tipo de Dato:	PK/FK	¿Nulo?	**Descripción:**
pk_id	Int4	PK	No	**Identificador de la tabla adbd.tb_dInforme.**
fk_nnivel_riesgo_usuario_id	Int4	Fk	No	**Identificador de la tabla adbd.tb_nnivel_riesgo**
fk_dmatriz_resultado_id	Int4	Fk	No	**Identificador de la tabla adbd.tb_dmatriz_resultado**
fk_ntipo_informe_id	Int4	Fk	No	**Identificador de la tabla adbd.tb_ntipo_informe**
fk_dusuario_id	Int4	Fk	No	**Identificador de la tabla adbd.tb_dusuario**
fk_dservidor_id	Int4	Fk	No	**Identificador de la tabla adbd.tb_dservidor**
fecha	date		No	**Almacena la fecha de cuando fue generado el informe.**
nombre_direccion_territorial	Char(20)		No	**Almacena el nombre de la dirección territorial a la que se le realizo la auditoría.**
alcance	Text		No	**Almacena el alcance de la auditoría.**
estrategia	Text		No	**Almacena la estrategia que sigue la auditoría**
impacto	Text		No	**Almacena el impacto de la auditoría**
consultante	Text		No	**Almacena el nombre del consultante que realiza la auditoría.**
Nombre: adbd.tb_nnivel_riesgo				
Descripción: **Almacena los datos de los niveles de riesgo.**				
Nombre:	Tipo de Dato:	PK/FK	¿Nulo?	**Descripción:**
pk_id	Int4	PK	No	**Identificador de la tabla adbd.tab_nnivel_riesgo.**

riesgo	Char(20)		No	
Nombre: adbd. tb_nentidad				
Descripción: **Almacena los datos de las entidades de ETECSA.**				
Nombre:	Tipo de Dato:	PK/FK	¿Nulo?	**Descripción:**
id	Int4	PK	No	**Identificador de la tabla adbd.tab_ nEntidades.**
nombre	Char(50)		No	
Nombre: adbd.tab_dservidor				
Descripción: **Almacena los datos de servidores a auditar.**				
Nombre:	Tipo de Dato:	PK/FK	¿Nulo?	**Descripción:**
id	Int4	PK	No	**Identificador de la tabla adbd.tab_nservidor.**
fk_ntipo_servidor_id	Int4	FK	No	**Identificador de la tabla adbd.tb_ ntipo_servidor.**
fk_nentidad_id	Int4	FK	No	**Identificador de la tabla adbd.tab_nentidad.**
Nombre	Char(20)		No	**Nombre del servidor.**
ip	Char(16)		No	**Ip del servidor.**
vulnerabilidades	text		No	
Nombre: adbd.tb_ ntipo_servidor				
Descripción: **Almacena los datos de los tipos de servidores a auditar.**				
Nombre:	Tipo de Dato:	PK/FK	¿Nulo?	**Descripción:**
id	Int4	PK	No	**Identificador de la tabla adbd.tb_ ntipo_servidor.**
tipo	Char(20)		No	**Almacena los tipos de servidores.**
Nombre: adbd. tb_dforma_evaluar				
Descripción: **Almacena los datos de las formas de evaluación de la consulta.**				
Nombre:	Tipo de Dato:	PK/FK	¿Nulo?	**Descripción:**
pk_id	Int4	PK	No	**Identificador de la tabla adbd. tb_dforma_evaluar.**
Nombre: adbd.tb_ dmatriz_resultado				

Descripción: **Almacena los datos de los tipos de servidores a auditar.**				
Nombre:	Tipo de Dato:	PK/FK	¿Nulo?	**Descripción:**
pk_id	Int4	PK	No	**Identificador de la tabla adbd.tb_dmatriz_resultado.**
tb_nnivel_evaluacion_sistema	**Int4**	**FK**	**No**	**Identificador de la tabla adbd.tb_nnivel_evaluacion.**

Tabla 8 Descripción de las tablas y atributos

3.7 Conclusiones

En este capítulo se decidió la utilización de una arquitectura multi capas la cual reduce la dependencia entre ellas, guiando el acceso solamente de capas superiores a capas inferiores, además el uso de patrones de diseño como el MVC aumentan la reutilización de los componentes para una posterior implementación web del sistema. Se demostró que con la utilización del patrón Localizador de Servicios se cumple el objetivo de manejar dinámicamente los componentes externos a la aplicación, los cuales serán los encargados de la evaluación de los resultados de auditoria. Se desarrollaron los diagramas de clases del diseño y de secuencia, y una descripción detallada de la arquitectura que guiará el proceso de desarrollo, con lo cual se cuentan con los artefactos necesarios para dar paso a la fase de Implementación del sistema.

CAPÍTULO 4: IMPLEMENTACIÓN Y PRUEBA

4.1 Introducción

En este capítulo se presentará el modelo de implementación, donde se describe cómo los elementos del modelo de diseño se implementarán en términos de componentes. Se expondrá además el modelo de despliegue de la aplicación y por último se realizarán pruebas al resultado de la implementación para comprobar el cumplimiento de los requisitos del sistema y disminuir los errores de la aplicación ejecutable.

4.2 Modelo de Implementación

El modelo de implementación describe cómo los elementos del modelo de diseño y las clases, se implementan en términos de componentes, ficheros de código fuente, ejecutables. El modelo de implementación describe también como se organizan los componentes de acuerdo con los mecanismos de estructuración y modularidad disponibles en el entorno de implementación y en el lenguaje de programación utilizado, y como depende los componentes unos de otros. (22)

4.3 Diagramas de implementación

El diagrama de componentes muestra las dependencias lógicas entre componentes de *software*. Describe la vista de implementación estática de cualquier sistema, sin importar tamaño o complejidad. Los componentes son unidades autónomas dentro de un sistema o subsistema, que representan tipos de elementos de *software* que forman parte del desarrollo de aplicaciones informáticas (22). A continuación de muestra el diagrama de componente del CU Generar Informe General, los demás diagramas de componentes se encuentran en los Anexos.

Diagrama de componente CU Generar Informe General

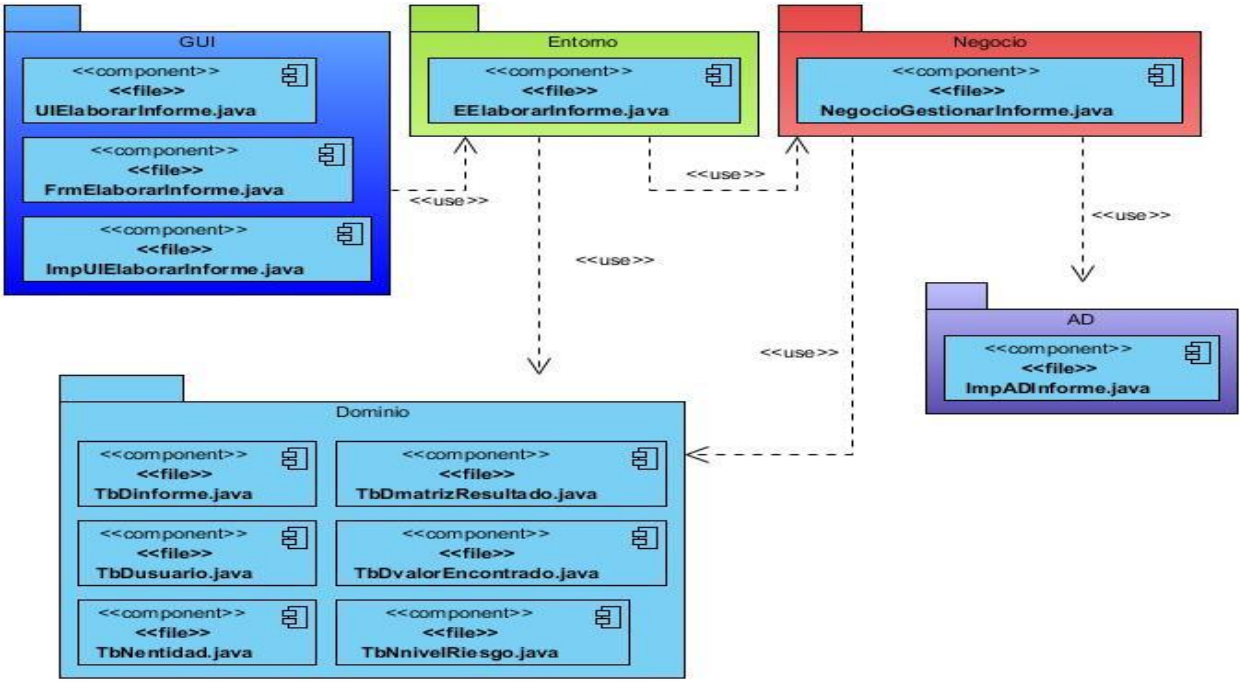


Figura No. 19 DC Generar Informe General

4.4 Diagrama de Despliegue

Un diagrama de despliegue muestra las relaciones físicas de los distintos nodos que componen un sistema. Un nodo constituye un recurso de ejecución que puede contener instancias de componentes de *software*, objetos o procesos (caso particular de un objeto). De forma general, un diagrama de despliegue es un grafo de nodos unidos por conexiones de comunicación que muestra las relaciones físicas entre los componentes hardware y *software* en el sistema final, es decir, presenta cómo están configurados los elementos de procesamiento en tiempo de ejecución y los componentes *software* (25). A continuación se muestra el diagrama de despliegue de la herramienta.

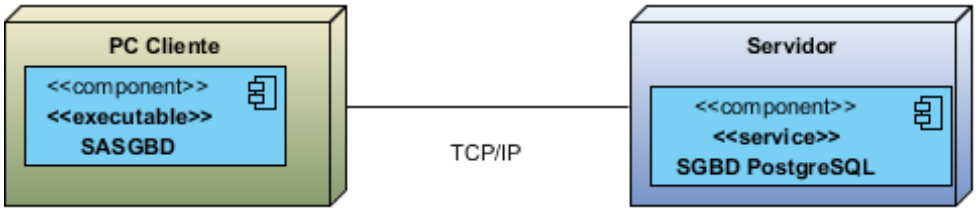


Figura No. 20 Diagrama de despliegue

4.4.1 Descripción de los nodos

PC Cliente: Ordenador desde donde se ejecutará la aplicación SASGBD.

Servidor: Ordenador que tendrá instalado el SGBD PostgreSQL al cual se conecta la aplicación SASGBD.

4.4.2 Protocolos utilizados

TCP/IP: Protocolo de red utilizado para realizar la comunicación entre el servidor de Bases de Datos y el SASGBD.

4.5 Diseño de casos de prueba

En el flujo de trabajo de prueba se verifican los resultados de la implementación probando cada construcción, incluyendo tanto construcciones internas como intermedias, así como versiones finales del sistema a ser entregadas a los clientes (10). Los objetivos de esta sección son diseñar e implementar las pruebas mediante la creación de casos de prueba, especificando qué probar y creando los procedimientos de prueba. Se realizarán las pruebas y presentación de los resultados de cada una de estas, con el objetivo de eliminar defectos en nuevas construcciones del sistema.

4.5.1 Tipos de Pruebas

- **Caja Negra**: Se le aplicaron las pruebas de caja negra a las interfaces del *software*, las cuales demostraron que las funcionalidades del sistema son óptimas. Los casos de prueba demostraron que todas las entradas fueron aceptadas adecuadamente y daban una salida correcta, así como que la integridad de la información externa se mantiene. Los casos de prueba de caja negra realizados se encuentran en los Anexos.

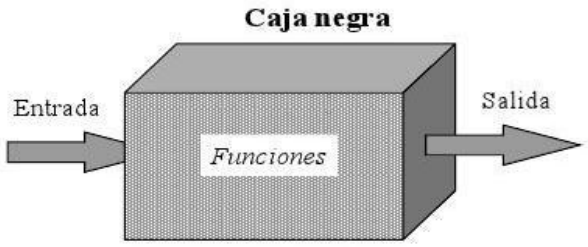


Figura No. 21 Pruebas de Caja negra

Caja Blanca: Las pruebas de caja blanca se le realizaron a las funcionalidades del *software* con el *framework* JUnit[32] para verificar el correcto funcionamiento del flujo de los datos entre los componentes relacionados así como el tiempo de respuesta. Los casos de pruebas de caja blanca realizados se encuentran en los Anexos.

[32] JUnit es un conjunto de bibliotecas creadas por Erich Gamma y Kent Beck que son utilizadas en programación para hacer pruebas unitarias de aplicaciones Java.

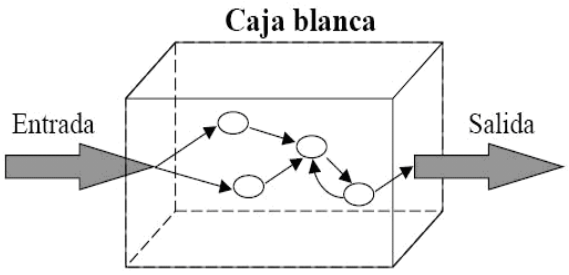


Figura No. 22 Pruebas de Caja blanca

4.5.2 Resultados de las pruebas Caja negra

De los casos de prueba realizados a las funcionalidades del sistema se obtuvieron una lista de no conformidades (ver Tabla 9) de ellas ninguna significativa, las cuales fueron resueltas, en la segunda iteración se comprobó el correcto funcionamiento de todas las funcionalidades del sistema.

Elemento	No	Ubicación de la No Conformidad	Etapa de Detección	Significativa	Estado NC	Respuesta del Equipo de Desarrollo
Alcance	1	Generar Informe General	Prueba	NO	-3/5/2012 pendiente -5/5/2012 resuelta	-
Estrategia	2	Exportar Informe General	Prueba	NO	-3/5/2012 pendiente -5/5/2012 resuelta	-
Fecha	3	Generar Informe General	Prueba	NO	-3/5/2012 pendiente -5/5/2012 resuelta	-
Descripción	4	Modificar Consulta	Prueba	NO	-3/5/2012 pendiente -5/5/2012 resuelta	-

Forma evaluación	**5**	Insertar Consulta	Prueba	NO	-3/5/2012 pendiente -5/5/2012 resuelta	-

Tabla 9 Lista de no conformidades

4.5.3 Resultado de las pruebas de Caja blanca

Los casos de prueba de caja blanca no arrojaron ningún error significativo. A continuación se muestran los resultados obtenidos del tiempo de ejecución de los CU críticos:

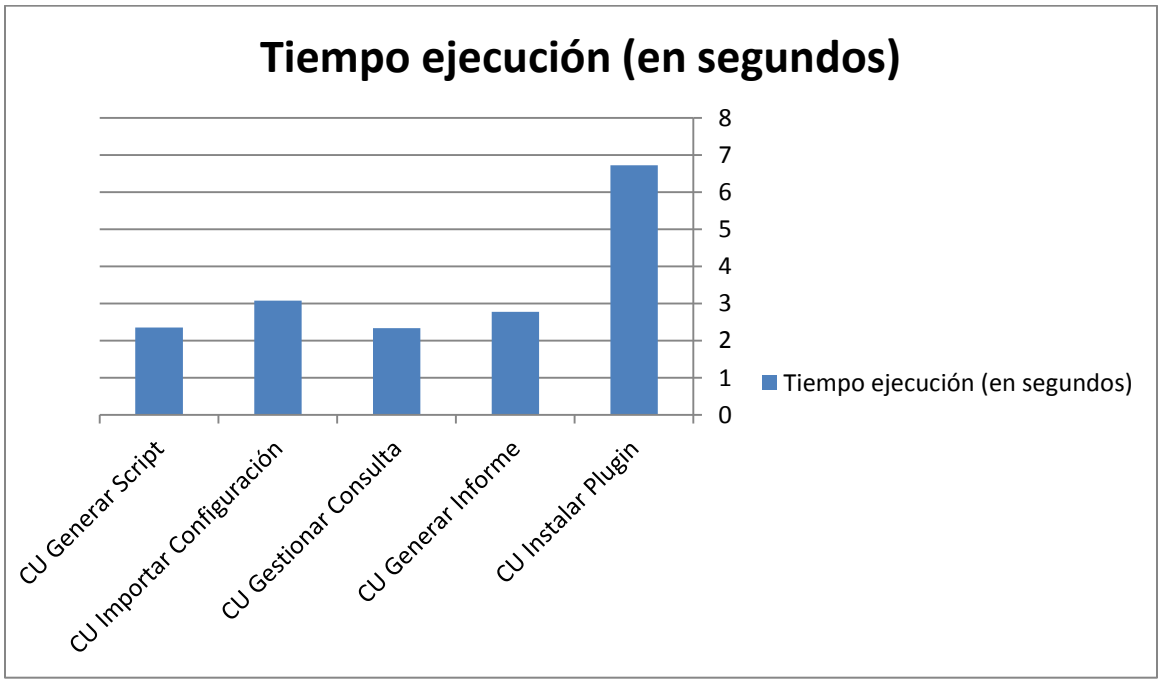


Figura No. 23 Resultado de las pruebas

4.6 Métricas para la comprobación de la calidad del software

Para la evaluación del software mediante métricas de calidad del software se utilizaron dos herramientas Checkstyle y Findbugs, ambos como plugins para el IDE seleccionado. Todas las deficiencias detectadas por estas herramientas fueron corregidas a continuación se muestran las deficiencias arrojadas por cada una de estas herramientas:

Checkstyle:

- Espacios innecesarios entre símbolos.
- Falta de los comentarios de documentación.

- Errores de indentación.
- Falta de espacios entre operadores.
- Errores de convención de nombres de paquetes, clases, variables y métodos.
- Errores en la asignación de los métodos de acceso a las variables, métodos y clases.

FindBugs:

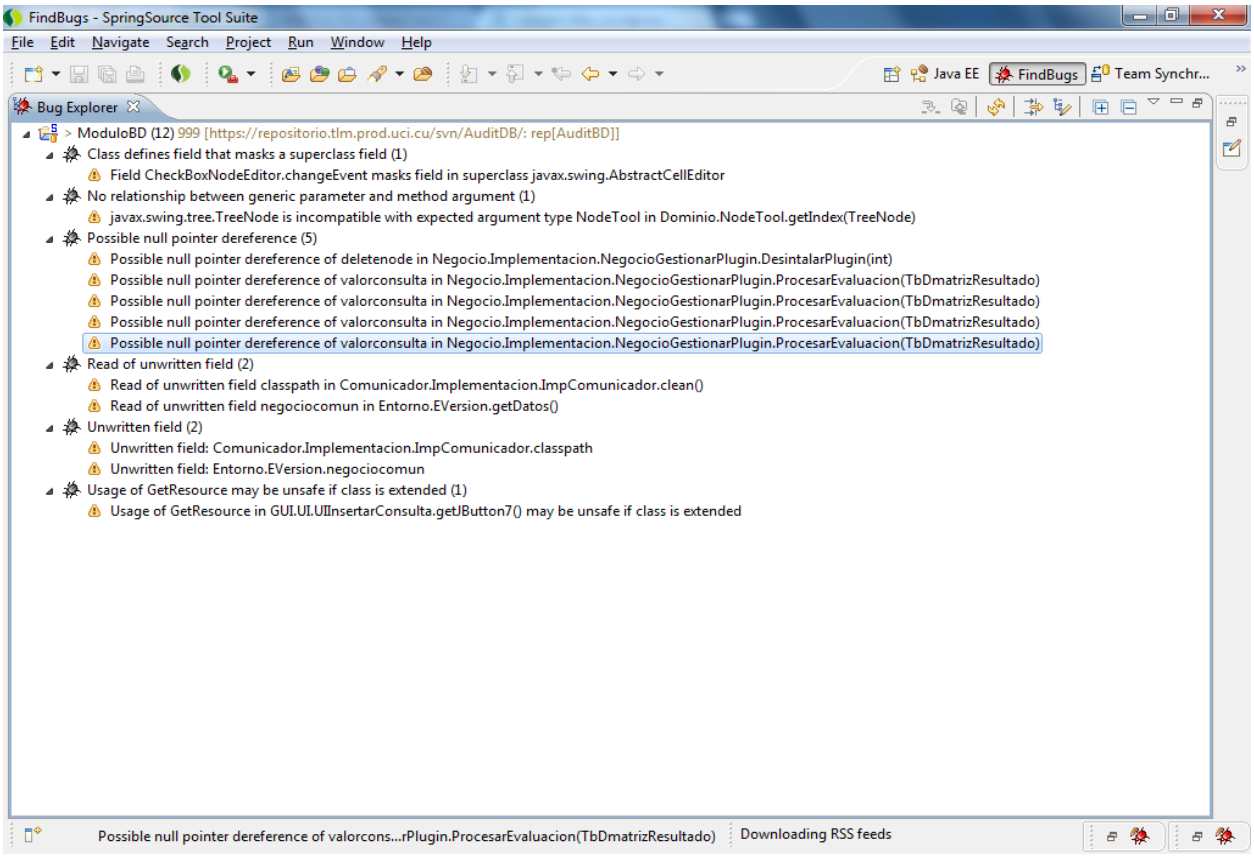


Figura No. 24 Resultados obtenidos con FindBugs

4.7 Conclusiones

El presente capitulo permitió la correcta implementación del sistema propuesto, en el cual se obtuvo un sistema que está compuesto por un componente ejecutable. Este componente será desplegado en un nodo y en un segundo nodo se desplegará un SGBD PostgreSQL, los cuales se comunicarán mediante protocolo TCP/IP. Además la realización de pruebas logró corregir errores de la implementación que atentaban contra el cumplimiento de los CU definidos anteriormente, los cuales daban respuesta a los requisitos funcionales de la aplicación. Se realizaron pruebas unitarias que permitieron comprobar que las funcionalidades

implementadas se ejecutaran correctamente y devolvieran los resultados esperados. Además se obtuvieron los tiempos de ejecución de cada uno de esas funcionalidades los cuales son todos menores de 7 segundos. Con la comprobación de métricas de calidad de software se corrigieron muchos errores que atentan con la calidad del producto final, eliminando posibles brechas de seguridad y logrando el cumplimiento de las pautas de codificación del centro de Telemática de la Facultad 2.

CONCLUSIONES GENERALES

Mediante el presente documento se abordó acerca del proceso de desarrollo del Sistema de realización de auditorías a SGBD. Dando cumplimiento el problema a resolver planteado y mejorando el proceso de la realización de auditorías a los SGBD a los cuales el departamento de SI supervisa.

El resultado de este trabajo constituye un ahorro al país de divisas en materia de pago de licencias para aplicaciones extranjeras que eran necesitadas por los clientes. Culminado el desarrollo del sistema se dio cumplimiento, de forma satisfactoria, a los objetivos trazados para el desarrollo de este trabajo:

- Se definió una arquitectura que permite la instalación de *plugins* para la evaluación de resultados de cualquier SGBD.
- Se diseñó e implementó una herramienta que domina todos los procesos básicos de realización de auditorías que realiza el departamento de SI de ETECSA.

Como producto final se obtuvo una aplicación de escritorio para la realización de auditorías en ETECSA. El país no cuenta con una aplicación de este tipo siendo la creación de la misma un avance tecnológico importante además de dar un paso más a la soberanía tecnología del país.

RECOMENDACIONES

Como recomendaciones para futuras versiones del sistema se propone:

- Implementar la versión web del sistema para multiplicar las posibilidades de desplegar en ambientes variados.
- Agregar funcionalidades para permitir añadir nuevas formas evaluativas.
- Añadir un sistema de reportes avanzados para proveer mayor información sobre los datos almacenados.
- Integrar el módulo “Herramienta colaborativa para la realización de auditorías a SGBD” para realizar las auditorías directamente desde el SASGB.

REFERENCIA BIBLIOGRAFÍA

1. **Antúnez Sánchez, Alcides Francisco and Odoardo Hernández, Noria.** *Auditoría y Seguridad Informática. Realidades y Perspectivas en Cuba, una mirada del escenario actual.*

2. **Empresa de Telecomunicaciones de Cuba S. A. (ETECSA).** Empresa de Telecomunicaciones de Cuba S.A. [Online] 2012. [Cited: 06 12, 2012.] http://www.etecsa.cu/?page=inicio.

3. **Hernández León, Rolando Alfredo and Coello González, Sayda .** *El paradigma cuantitativo de la investigación científica.* Cuidad de la Habana : Editorial Universitaria, 2002. ISBN 978-959-16-0343-2.

4. Sitio - Hoy Digital. [Online] [Cited: Enero 17, 2012.] http://hoy.com.do/investigacion/2009/1/5/261742/TecnologiaQue-es-auditoria-informatica.

5. **Ramírez Rodríguez, Germán.** *SOBRE LA AUDITORÍA INFORMÁTICA Y LOPD DESDE DE EXPERIANCIA PERSONAL.* Madrid : s.n., 2009.

6. **SoftTree Technologies, Inc.** DB Audit Home Page. [Online] 2011. [Cited: 01 15, 2012.] http://www.softtreetech.com/dbaudit/.

7. **dbWatch Software.** Database Monitoring. [Online] 2011. [Cited: 01 15, 2012.] http://www.dbwatchsoftware.com/.

8. **Application Security, Inc.** DbProtect - Database Security | Vulnerability Assessment | Activity Monitoring and Auditing. [Online] 2012. [Cited: 01 15, 2012.] http://www.appsecinc.com/products/dbprotect/.

9. —. Application Security Inc. DataBase Security Assesment. [Online] 2012. [Cited: 01 16, 2012.] http://www.appsecinc.com/products/appdetective/.

10. **Jacobson, Ivar, Booch, Grady y Rumbaugh.** *El proceso Unificado de Desarrollo de Software.* 1era. Madrir : Addison Wesley, 2000.

11. **Visual Paradigm.** UML, BPMN and Database Tool for Software Development. [Online] 2011. [Cited: 01 13, 2012.] www.visual-paradigm.com.

12. **Sánchez, María A. Mendoza.** *Metodologías De Desarrollo De Software.* 2004.

13. **Jeffrey Savit, Sean Wilcox & Sean Wilcox, Bhuvana Jayaraman.** *Java para la empresa.* s.l. : McGraw-Hill, 1999.

14. **Walls, Craig and Breidenbach, Ryan.** *Spring in action.* s.l. : Manning Publications, 2008. 1-933988-13-4.

15. **Meszaros, Gerard.** *xUnit Test Patterns - Refactoring Test Code.* s.l. : Addison Wesley, 2007. 0131495054.

16. **Nathaniel Ayewah, William Pugh and Penix, John.** *Evaluating Static Analysis Defect Warnings.* University of Maryland : s.n., 2007. 978-1-59593-595.

17. **Oracle.** Code Conventions for the Java Programming Language. [Online] 2012. [Cited: 06 12, 2012.] http://www.oracle.com/technetwork/java/codeconv-138413.html.

18. **Bauer, Christian and King, Gavin.** *Hibernate in action.* s.l. : Manning publications Co., 2005. 1932394-15-X.

19. **PostgreSQL Global Development Group .** PostgreSQL Advantages. [Online] 2012. [Cited: 01 19, 2012.] http://www.postgresql.org/about/advantages/.

20. **Martínez, Alejandro ; Martínez, Raúl.** Guía a Rational Unified Process. [Online] [Cited: Enero 19, 2012.] http://www.dsi.uclm.es/asignaturas/42551/trabajosAnteriores/Trabajo-Guia RUP.PDF.

21. **Pressman, Roger S.** *Software Engineering: A Practitioner's, Approach, Seventh Edition.* New York : McGraw-Hil Companies, 2010. 978-0-07-337 597 -7.

22. **Visconti, Marcello and Astudillo, Hernán.** Fundamentos de Ingeniería de Software. [Online] [Cited: Abril 29, 2012.] http://www.inf.utfsm.cl/~visconti/ili236/Documentos/08-Patrones.PDF.

23. **Lago, Ramiro.** Patrones de diseño software. [Online] Abril 2007. [Cited: Marzo 9, 2012.] http://www.proactiva-calidad.com/java/patrones/index.html.

24. **Guillerón, Gastón.** Inversion de Control « Gln Blog. [Online] [Cited: Abril 10, 2012.] http://glnconsultora.com/blog/?tag=inversion-de-control.

25. Guía de Usuario de Enterprise Architect 7.0. [Online] [Cited: 4 10, 2012.] http://www.sparxsystems.com.ar/download/ayuda/index.html?componentdiagram.htm.

BIBLIOGRAFÍA

1. **Antúnez Sánchez, Alcides Francisco and Odoardo Hernández, Noria.** *Auditoría y Seguridad Informática. Realidades y Perspectivas en Cuba, una mirada del escenario actual.*

2. **Empresa de Telecomunicaciones de Cuba S. A. (ETECSA).** Empresa de Telecomunicaciones de Cuba S.A. [Online] 2012. [Cited: 06 12, 2012.] http://www.etecsa.cu/?page=inicio.

3. **Hernández León, Rolando Alfredo and Coello González, Sayda .** *El paradigma cuantitativo de la investigación científica.* Cuidad de la Habana : Editorial Universitaria, 2002. ISBN 978-959-16-0343-2.

4. Sitio - Hoy Digital. [Online] [Cited: Enero 17, 2012.] http://hoy.com.do/investigacion/2009/1/5/261742/TecnologiaQue-es-auditoria-informatica.

5. **Ramírez Rodríguez, Germán.** *SOBRE LA AUDITORÍA INFORMÁTICA Y LOPD DESDE DE EXPERIANCIA PERSONAL.* Madrid : s.n., 2009.

6. **SoftTree Technologies, Inc.** DB Audit Home Page. [Online] 2011. [Cited: 01 15, 2012.] http://www.softtreetech.com/dbaudit/.

7. **dbWatch Software.** Database Monitoring. [Online] 2011. [Cited: 01 15, 2012.] http://www.dbwatchsoftware.com/.

8. **Application Security, Inc.** DbProtect - Database Security | Vulnerability Assessment | Activity Monitoring and Auditing. [Online] 2012. [Cited: 01 15, 2012.] http://www.appsecinc.com/products/dbprotect/.

9. —. Application Security Inc. DataBase Security Assesment. [Online] 2012. [Cited: 01 16, 2012.] http://www.appsecinc.com/products/appdetective/.

10. **Jacobson, Ivar, Booch, Grady y Rumbaugh.** *El proceso Unificado de Desarrollo de Software.* 1era. Madrir : Addison Wesley, 2000.

11. **Visual Paradigm.** UML, BPMN and Database Tool for Software Development. [Online] 2011. [Cited: 01 13, 2012.] www.visual-paradigm.com.

12. **Sánchez, María A. Mendoza.** *Metodologías De Desarrollo De Software.* 2004.

13. **Jeffrey Savit, Sean Wilcox & Sean Wilcox, Bhuvana Jayaraman.** *Java para la empresa.* s.l. : McGraw-Hill, 1999.

14. **Walls, Craig and Breidenbach, Ryan.** *Spring in action.* s.l. : Manning Publications, 2008. 1-933988-13-4.

15. **Meszaros, Gerard.** *xUnit Test Patterns - Refactoring Test Code.* s.l. : Addison Wesley, 2007. 0131495054.

16. **Nathaniel Ayewah, William Pugh and Penix, John.** *Evaluating Static Analysis Defect Warnings.* University of Maryland : s.n., 2007. 978-1-59593-595.

17. **Oracle.** Code Conventions for the Java Programming Language. [Online] 2012. [Cited: 06 12, 2012.] http://www.oracle.com/technetwork/java/codeconv-138413.html.

18. **Bauer, Christian and King, Gavin.** *Hibernate in action.* s.l. : Manning publications Co., 2005. 1932394-15-X.

19. **PostgreSQL Global Development Group .** PostgreSQL Advantages. [Online] 2012. [Cited: 01 19, 2012.] http://www.postgresql.org/about/advantages/.

20. **Martínez, Alejandro ; Martínez, Raúl.** Guía a Rational Unified Process. [Online] [Cited: Enero 19, 2012.] http://www.dsi.uclm.es/asignaturas/42551/trabajosAnteriores/Trabajo-Guia RUP.PDF.

21. **Pressman, Roger S.** *Software Engineering: A Practitioner's, Approach, Seventh Edition.* New York : McGraw-Hil Companies, 2010. 978-0-07-337 597 -7.

22. **Visconti, Marcello and Astudillo, Hernán.** Fundamentos de Ingeniería de Software. [Online] [Cited: Abril 29, 2012.] http://www.inf.utfsm.cl/~visconti/ili236/Documentos/08-Patrones.PDF.

23. **Lago, Ramiro.** Patrones de diseño software. [Online] Abril 2007. [Cited: Marzo 9, 2012.] http://www.proactiva-calidad.com/java/patrones/index.html.

24. **Guillerón, Gastón.** Inversion de Control « Gln Blog. [Online] [Cited: Abril 10, 2012.] http://glnconsultora.com/blog/?tag=inversion-de-control.

25. Guía de Usuario de Enterprise Architect 7.0. [Online] [Cited: 4 10, 2012.] http://www.sparxsystems.com.ar/download/ayuda/index.html?componentdiagram.htm.

26. Sitio de la Universidad de Palermo. *Sitio de la Universidad de Palermo.* [Online] [Cited: Enero 20, 2012.] http://www.palermo.edu/ingenieria/downloads/introduccion_spring_framework_v1.0.PDF.

27. **Fajardo , Ivan.** Service Locator Design pattern. *http://ivanfajardo.blogspot.com/2006/10/service-locator-design-pattern.html.* [Online] 2006.

28. **SpringSource.** About Spring. [Online] SpringSource, a division of VMware, 2012. [Cited: 01 19, 2012.] http://www.springsource.org/about.

29. **Eclipse Foundation.** *Eclipse Platform Technical Overview.* s.l. : Object Technology International, Inc., 2003.

30. *Migración a plataformas de aplicaciones informáticas libres y códigos fuente abiertos.* **Antich, Arnaldo Coro.** La Habana,UCI : s.n., 2011.

31. **Geetanjali Arora, Balasubramaniam Aiaswamy & Nitin Pandey.** *Programación en C#.* España : Anaya Multimedia, 2002.

32. **White, Stephen A.** *Introduction to BPMN.* s.l. : IBM, 1994.

GLOSARIO DE TÉRMINOS

Acoplamiento: Grado de dependencia de los componentes entre sí. Existen dos tipos de acoplamiento: “estrecho” y “holgado”. El acoplamiento holgado es recomendable en arquitectura de *software* ampliable, pero se aconseja el acoplamiento estrecho para obtener el máximo rendimiento. El acoplamiento aumenta a medida que los datos que intercambian los componentes crecen y se hacen más complejos.

Auditoría. Técnica de control, dirigida a valorar, el control interno y la observancia de las Normas Generales de Contabilidad. Comprende un examen independiente de los registros de contabilidad y otra evidencia relacionada con una entidad para apoyar la opinión experta imparcial sobre la confiabilidad de los estados financieros.

Aplicación: Sistema que ofrece a un usuario final un conjunto coherente de casos de uso.

Arquitectura: Conjunto de decisiones significativas acerca de la organización de un sistema *software*, la selección de los elementos estructurales a partir de los cuales se compone el sistema, y las interfaces entre ellos, junto con su comportamiento, tal y como se especifica en las colaboraciones entre esos elementos, la composición de estos elementos estructurales y de comportamiento en subsistemas progresivamente mayores, y el estilo arquitectónico que guía está organización: estos elementos y sus interfaces, sus colaboraciones y su composición. La arquitectura del *software* se interesa no solo por la estructura y el comportamiento, sino también por las restricciones y compromisos de uso, funcionalidad, flexibilidad al cambio, funcionamiento, reutilización, comprensión, economía y tecnología, así como por aspectos estéticos.

Artefacto: Pieza de información tangible que es creada, modificada y usada por los trabajadores al realizar actividades; representa un área de responsabilidad, y es candidata a ser tenida en cuenta para el control de la configuración. Un artefacto puede ser un modelo, un elemento de un modelo, o un documento.

Base de Datos: Conjunto no redundante de información almacenada en memoria organizada independientemente de su utilización y su implementación en máquinas accesibles en tiempo real y compatible con usuarios concurrentes con necesidad de información diferente y no predicable en tiempo.

Clase: Descripción de un conjunto de objetos que comparten los mismos atributos, operaciones, métodos, relaciones y semánticas. Una clase puede utilizar un conjunto de interfaces para especificar recopilaciones de operaciones que proporciona a su entorno.

Cohesión: Unión innata de componentes del mismo tipo que dependen uno del otro. Acción o estado de permanecer juntos; unión cercana.

Diagrama: Representación gráfica de todo el modelo o sólo de una parte.

Entidad (empresarial): Una unidad económica que realiza transacciones comerciales que se deben registrar, resumir y reportar. Se considera la entidad separada de su propietario o propietarios.

Eficacia: El grado en que se cumplen los objetivos y la relación entre el efecto deseado en una actividad y su efecto real.

Eficiencia: La relación que existe entre el producto (en término de bienes, servicios u otros resultados) y los recursos empleados en su producción.

Framework: Es una estructura de soporte definida, en la cual otro proyecto de *software* puede ser organizado y desarrollado. Puede incluir soporte de programas, bibliotecas y un lenguaje de *scripting* entre otros *software* para ayudar a desarrollar y unir los diferentes componentes de un proyecto.

GUI:(en inglés Graphical User Interface): Interfaz Gráfica de Usuario es un tipo de interfaz de usuario que utiliza un conjunto de imágenes y objetos gráficos para representar la información y acciones disponibles en la interfaz. Habitualmente las acciones se realizan mediante manipulación directa para facilitar la interacción del usuario con la computadora.

Informe de auditoría: Expresión escrita por el auditor respecto a los resultados de las verificaciones realizadas durante la ejecución de la auditoría, manifestando sus criterios y comentarios respecto a los estados financieros y otros hechos económicos.

Modelo: Abstracción semánticamente cerrada de un sistema; descripción completa de un sistema desde una perspectiva determinada (completa significa que no es necesario añadir información adicional para comprender el sistema desde dicha perspectiva); conjunto de elementos de modelo. Dos modelos no se pueden solapar.

Nodo: Un nodo es un clasificador que representa un recurso computacional de tiempo de ejecución, que generalmente tiene al menos una memoria y a menudo capacidad de proceso. Los componentes y objetos de tiempo de ejecución pueden residir en los nodos.

Plataforma: En informática, una plataforma es un sistema que sirve como base para hacer funcionar determinados módulos de hardware o de *software* con los que es compatible.

Proceso de auditoría. Las acciones que realiza el auditor para llevar a cabo sus labores de revisión.

Control de versiones: Es la gestión de los diversos cambios que se realizan sobre los elementos de algún producto o una configuración del mismo. Una versión, revisión o edición de un producto, es el estado en el que se encuentra dicho producto en un momento dado de su desarrollo o modificación.

Supervisión: Requisito esencial en la auditoría que conduce a un adecuado control y dirección en todas las etapas, para que asegure que los procedimientos, técnicas y pruebas que se realizan, se vinculen en forma competente y eficaz con los objetivos que se persiguen.

Seguridad Informática: La seguridad informática, es el área de la informática que se enfoca en la protección de la infraestructura computacional y todo lo relacionado con esta (incluyendo la información contenida.

Sistemas informáticos: Un sistema informático como todo sistema, es el conjunto de partes interrelacionadas, hardware, *software* y de recurso humano que permite almacenar y procesar información.

***Software* libre:** El *software* libre (en inglés *free software*, aunque esta denominación también se confunde a veces con “gratis” por la ambigüedad del término “free” en el idioma inglés, por lo que también se usa “libre *software*” y “*logical* libre”) es la denominación del *software* que respeta la libertad de los usuarios sobre su producto adquirido y, por tanto, una vez obtenido puede ser usado, copiado, estudiado, modificado, y redistribuido libremente.

Tecnologías libres: Las tecnologías libres son aquellas que no precisan de autorización o licencia para su uso. Más bien, pertenecen a la sabiduría y cultura popular, propias de la ciudadanía, que es quien las utiliza y explota en su propio beneficio.

Vulnerabilidades: Son errores que permiten realizar desde afuera actos sin permiso del administrador del equipo.