Universidad de las Ciencias Informáticas

Facultad 3

Título: "Componente de gestión de políticas de control de acceso a nivel de base de datos para el sistema Acaxia"

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

Autor(es): José Eduardo Vázquez Hernández
Katia Saria Preval

Tutor(es): Ing. Mileidy Magaly Sarduy Pérez.
Ing. Andrés Reynaldo Milord

2013 junio

Declaración de autoría

Declaramos ser autores de la presente tesis y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste firmo la presente a los 12 días del mes de junio del año 2013.

_____ _____

José Eduardo Vázquez Hernández **Katia Saria Preval**

_____ _____

Ing. Mileidy M. Sarduy Pérez **Ing. Andrés Reynaldo Milord**

Síntesis de los tutores:

Ing. Mileidy M. Sarduy Pérez.

Ingeniera en Ciencias Informáticas, graduada en el año 2009. Actualmente se desempeña como Jefa del proyecto y ocupa el rol de analista principal del proyecto de Gestión Integral de Seguridad (Acaxia).

Ing. Andrés Reynaldo Milord.

Ingeniero en Ciencias Informáticas, graduado en el 2012. Actualmente se desempeña como desarrollador en el proyecto Gestión Integral de Seguridad (Acaxia).

De Katia:

Infinitas gracias doy a Dios, el dador de la vida, porque gracias a Él existo. Por los cuidados que ha tenido para conmigo y mi familia, porque cada día me da una nueva oportunidad de ser mejor.

Gracias a la Revolución cubana por darme la oportunidad de obtener un título universitario.

No puedo decirle a Maíta "Gracias", sino "Te lo debo todo". Has sido esa fuerza que me ha impulsado cada día a seguir, a no detenerme, siempre recordando que "el que con lágrimas cosecha, con gozo recogerá". A ti doy las gracias por ser mi madre.

A mi Claudiñita, mi hermanita, que con sólo 15 años ha enfrentado la vida con una madurez y responsabilidad envidiable. Gracias por tus sacrificios, no soy un ejemplo para ti, sino que tú lo eres para mí. Te quiero muchísimo.

A mi papá, por estar siempre presto a brindar su apoyo. Muchas gracias por todo.

A mi HD, por soportarme durante todo este tiempo. Por ayudarme a descubrir lo que soy capaz de lograr. Gracias porque desde que llegaste a mi vida por más pesados que fueran mis problemas, siempre me has ayudado a llevarlos. Gracias por ser tú, por tu apoyo incondicional pero sobre todo, muchas gracias por creer en mí. TKM.

A mis suegras Mimi y Noemí y a mi cuñi Daniela, por todo su cariño y comprensión. Porque me han hecho sentir que tengo otra familia.

A mis 11 hermanos dondequiera que estén. Gracias porque al llegar a esta universidad se convirtieron en mi inspiración y me hicieron conocer lo que es el verdadero amor de un hermano y un amigo cristiano. Los quiero inmenso.

A mis amigos Mendoza, Leidy, Piñeiris, Pedruco, Gladys, Annia, La Soto, El Dary. Gracias por todos aquellos momentos inolvidables, tienen un espacio reservado para ustedes en mi corazón.

A los profes Yalice, Sandor, Navarrete (El Nava), Ivelis, Yinett, porque me apoyaron en el momento que más lo necesitaba.

A mis tutores, Mile y Andrés, a Rene, a Yoel y a todos mis compañeros de proyecto.

A Jose, mi súper compañero de tesis. Gracias por estar siempre dispuesto a ayudarme, no solo como compañero de tesis, sino como amigo.

A mis tías, porque siempre me han apoyado y me estiman y me quieren mucho, gracias a todas por confiar en mi.

A una persona en especial, a la profe Hilda Ester Heredia, a ella ***GRACIAS*** *de todo corazón porque cuando todo parecía perdido ahí estuvo. A usted le debo este título universitario.*

Gracias a la UCI, porque me ha permitido crecer, como profesional y como persona, gracias por todas las oportunidades que me ha brindado.

A todos los que permitieron que hoy estuviera redactando estos agradecimientos para un trabajo de diploma. ***Muchas gracias.***

De Jose:

A mi novia Eliza por haber sido durante estos últimos 5 años la única persona que ha estado conmigo en mis momentos más difíciles y en los más alegres. Por haberme apoyado en muchos de mis proyectos de vida, por haber sido mí sostén, por preocuparse tanto por mí y por amarme tanto.

A mis padres por haber sido capaces de traerme hoy hasta aquí, con todos los esfuerzos, preocupaciones y dolores de cabeza que eso ha implicado.

A mis hermanas por evitarme la soledad.

A todos mis abuelos y tíos abuelos por la ayuda siempre que la he necesitado. Por la atención constante.

A mis tíos y en especial a Hano por ser otro padre.

A mi grupo porque en una escuela como esta no hay cabida para el individualismo y se necesita siempre de personas como ellos para triunfar.

A mi nuevo equipo de trabajo por las largas noches de lunes, por todo lo que con ellos he aprendido y por ser movidos por las mismas fuerzas que me mueven a mí.

A Yoel por la paciencia y la dedicación.

A los tutores por toda la ayuda.

A todos los presentes por el placer de tenerlos aquí.

Al Comandante y a Revolución por continuar forjándome y por ser la razón por la que no dejo momento al descanso.

De Katia:

A Maíta y Claudia por sus sacrificios de amor, perseverancia y apoyo incondicional.

De Jose:

De manera especial quisiera dedicarle esta tesis a mi abuela Beby por ser mi mentora, mi apoyo y estar conmigo en los momentos más trascendentales de mi vida.

Resumen

La seguridad de los datos que se manejan en una institución es una de las actividades esenciales que debe ser altamente considerada. El control de acceso a la información que se procesa en cualquier entidad debe estar respaldado por medidas que regulen la forma en que se manipulan los datos. El Sistema de Gestión Integral de Seguridad (Acaxia) se desarrolla sobre el marco de trabajo Sauxe, con el objetivo de gestionar la seguridad de los sistemas que se suscriban a él. La mayoría de la información sensible que manejan las empresas en el mundo está almacenada en los sistemas gestores de base de datos, es por ello que estos constituyen uno de los objetivos principales de los atacantes. Actualmente Sauxe establece una única conexión a la base de datos para todos los sistemas que se encuentran suscritos a Acaxia, lo que representa una brecha en la seguridad. El presente trabajo tiene como objetivo desarrollar un componente de gestión de políticas de control de acceso a nivel de base de datos para Acaxia. Dicho componente brindará la posibilidad de personalizar las transacciones a decisión de las entidades que utilicen Acaxia. Esto permitirá aumentar la granularidad a nivel de base de datos para que los usuarios posean solamente los permisos que les correspondan para desempeñar sus tareas.

PALABRAS CLAVES

base de datos, control de acceso, granularidad, seguridad.

Índice de contenidos

Índice de figuras

Índice de tablas

Introducción

En las últimas décadas del siglo XX las Tecnologías de la Informática y las Comunicaciones (TIC) alcanzaron un acelerado desarrollo, propiciando un considerable aumento en su utilización debido a las grandes ventajas que brindan. En la actualidad, la mayoría de las actividades industriales, comerciales, militares, investigativas y de servicios dependen de los Sistemas de Información (SI) para su funcionamiento. Estos nuevos sistemas se vuelven cada vez más complejos y necesarios en cualquier entidad, lo que hace más difícil una buena gestión de la seguridad sobre los datos que manejan. Este hecho provoca que éstos SI sean cada vez más susceptibles a ataques, lo cual implica la creación de múltiples herramientas y aplicaciones informáticas que garanticen un buen manejo de la seguridad. Una de las actividades más esenciales y complejas que existen actualmente, debido a la gran diversidad de amenazas a las que los SI se encuentran sometidos es la de fortalecer la seguridad. Un ataque sobre un SI en muchas ocasiones está causado por el aprovechamiento de brechas de seguridad que no son detectadas por los informáticos responsables de la construcción de este software, provocando pérdida de datos y la obstrucción parcial o total de los servicios que brinda. Con el objetivo de evitar la ocurrencia de ataques sobre los sistemas informáticos surge una ciencia denominada Seguridad Informática, la cual refleja algunos de los principios que debe cumplir un software para considerarse seguro: (Pfleeger, 2006)

- Eslabón más débil.
- Mínimo privilegio.
- Dinamismo.
- Proporcionalidad.
- Participación universal.

Dos de los principios de la Seguridad Informática que son violados con regularidad son el de mínimo privilegio, el cual consiste en *"otorgar sólo los permisos estrictamente necesarios para efectuar las acciones que se requieran, ni más ni menos de los solicitados"* (Pfleeger, 2006) y el de proporcionalidad que consiste en que "*las medidas de seguridad deben estar en correspondencia con lo que se protege y con el nivel de riesgo existente. No sería lógico proteger con múltiples recursos un activo informático que no posee valor o que la probabilidad de ocurrencia de un ataque sobre el mismo es muy baja"*. (Pfleeger, 2006)

En los últimos años ha proliferado el uso de las aplicaciones web, las cuales son herramientas que los usuarios pueden utilizar accediendo a un servidor web a través de Internet o de una intranet

mediante un navegador. Estas herramientas son muy utilizadas por las instituciones para la gestión de sus procesos empresariales.

La existencia de brechas de seguridad en estas aplicaciones ha propiciado un aumento de los ataques informáticos que son ejecutados contra estos sistemas. Consecuentemente con el aumento en los últimos años de pérdidas económicas ocasionadas por estos tipos de ataques, la Universidad de las Ciencias Informáticas asume la tarea de fortalecer la seguridad en los SI. Ante esta necesidad, en el departamento de Tecnología del Centro para la Informatización de la Gestión de Entidades (CEIGE) se decide desarrollar el Sistema de Gestión Integral de Seguridad (Acaxia), el cual brindará sus servicios a todos los SI que se suscriban a él. Para ello implementa cinco procesos que debe cumplir un sistema de este tipo, autenticación, autorización, auditoría, administración de perfiles y la administración de conexiones. Acaxia se desarrolla sobre el marco de trabajo para el desarrollo de aplicaciones web de gestión (Sauxe).

En la actualidad Sauxe registra las conexiones de los sistemas suscritos a Acaxia en un fichero; de manera que existe una única conexión a la base de datos para todos los sistemas registrados. Esto trae consigo que los usuarios pertenecientes a los sistemas realicen todas las operaciones en la aplicación con una única conexión que tiene privilegios sobre objetos de la base de datos a los cuales no deberían tener acceso. El exceso de privilegios en la capa de datos representa una brecha de seguridad alarmante existente en Acaxia. Una persona con conocimientos básicos de informática, con posibilidades de acceder al fichero de texto donde se almacenan las credenciales de conexión y con acceso al servidor de base de datos, tiene la posibilidad de alterar la información contenida en el mismo (adición, modificación, eliminación). Puede ser muy difícil y hasta imposible determinar el responsable de esta alteración. Un ataque de esta magnitud puede representar pérdidas incalculables para los sistemas que Acaxia se encarga de proteger.

Por otra parte, Sauxe define una configuración estándar en la gestión de conexiones a la base de datos para los sistemas que utilicen Acaxia. Este hecho provoca que no se tenga en cuenta que a algunos de estos sistemas les interese más que a otros aumentar la seguridad teniendo en cuenta el tipo de información que manejan. Lo anteriormente planteado atenta contra dos de los principios de la Seguridad Informática descritos: mínimo privilegio y proporcionalidad. Partiendo de lo antes expuesto se tiene como **problema a resolver**: ¿Cómo aumentar la granularidad a nivel de base de datos en el establecimiento de políticas de control de acceso en el sistema Acaxia? Para darle respuesta a esta problemática se define como **objeto de estudio:** el proceso de administración de conexiones a base de datos en los frameworks de desarrollo y sistemas de seguridad. Centrado en el **campo de acción:**

proceso de administración de las conexiones a base de datos en Acaxia orientado al gestor PostgresSQL. Basado en el problema planteado se determina como **objetivo general**: desarrollar un componente de gestión de políticas de control de acceso para aumentar la granularidad a nivel de base de datos en el sistema Acaxia. Para lograr el cumplimiento del objetivo general se trazaron los siguientes **objetivos específicos:**

- Construir el marco teórico conceptual de la investigación sobre la gestión de conexiones en los frameworks de desarrollo y sistemas de seguridad.
- Realizar el análisis y diseño del componente para la gestión de políticas de control de acceso a nivel de base de datos en el sistema Acaxia.
- Implementar el componente para la gestión de políticas de control de acceso a nivel de base de datos en el sistema Acaxia.
- Validar el componente para la gestión de políticas de control de acceso a nivel de base de datos en el sistema Acaxia a partir de pruebas de caja blanca y caja negra.

Posibles resultados: componente de gestión de políticas de control de acceso a nivel de base de datos para el sistema Acaxia.

Para el desarrollo del presente trabajo se parte de la **Idea a defender:** si se desarrolla un componente de gestión de políticas de control de acceso se logrará aumentar la granularidad a nivel de base de datos en el sistema Acaxia.

Los siguientes métodos teóricos sustentan el trabajo de investigación:

- **Histórico – Lógico:** permitirá realizar un estudio de las principales soluciones relacionadas con la investigación a nivel mundial.
- **Analítico – Sintético:** el cual permitirá determinar si es factible o no utilizar alguna de las características que aporten los sistemas estudiados para gestionar políticas de control de acceso a base de datos en el ambiente en el que se desarrolla el sistema de seguridad Acaxia. Resultando muy importante para enfocar la investigación en los elementos más significativos, arrojando ideas y conclusiones mucho más prácticas y concretas.

El método empírico a utilizar es:

- **Entrevista:** permitirá la obtención de información relacionada con la investigación a través de entrevistas a los clientes del sistema Acaxia y a expertos en el tema.

La presente investigación está estructurada en tres capítulos:

Capítulo 1: Fundamentación teórica.

Se hace referencia a los principales conceptos claves para la investigación. Se realiza un estudio al Modelo de Control de Acceso Basado en Roles. Se hace un análisis del estado del arte de algunos de los marcos de trabajo existentes y sistemas de seguridad enfocado a la forma en que gestionan las conexiones a base de datos. Se describen las herramientas, metodologías y tecnologías definidas por el CEIGE para el desarrollo de la solución propuesta.

Capítulo 2: Propuesta de solución.

En este capítulo se exponen los requisitos funcionales y no funcionales identificados con que contará el componente de gestión de políticas de control de acceso a base de datos en el sistema Acaxia. Se muestra el modelo conceptual, prototipos de interfaces así como el modelo de datos y diagramas de clases. Se identifican los patrones arquitectónicos y de diseño a utilizar para el desarrollo del componente.

Capítulo 3: Implementación.

Se aplican las métricas Tamaño Operacional de Clases y Relaciones entre Clases al diseño elaborado. Se exponen los estándares de codificación utilizados para lograr un buen entendimiento y legibilidad del código. Se describen los artefactos generados como el diagrama de despliegue y el de componentes. Se aplica la prueba de caja blanca utilizando la técnica del camino básico a la implementación. Se muestran los resultados de las pruebas funcionales realizadas a la solución para demostrar su correcto funcionamiento. Se evidencia el cumplimiento de la idea a defender planteada; a través de un cuestionario respondido por los clientes.

Capítulo 1: Fundamentación teórica

Introducción

La mayoría de la información que es generada hoy en día se almacena en los gestores de base de datos relacionales. Se hace muy importante que los frameworks de desarrollo y los sistemas de seguridad realicen un buen manejo de las transacciones hacia estos gestores. Esto posibilitará mantener una mayor confiabilidad sobre los datos que son consultados. Por ello en el presente capítulo se realiza el marco conceptual sobre los principales temas que se abordarán alrededor de la presente investigación. Se estudiarán varios marcos de trabajo y un sistema de seguridad, teniendo en cuenta la forma en que estos gestionan las transacciones a la base de datos.

1.1 La seguridad en los Sistemas de Información

La **Seguridad Informática** es "*un conjunto de métodos y herramientas destinados a proteger los bienes informáticos de una institución*". (Pfleeger, 2006) Por ello las entidades que utilizan SI para la gestión de sus procesos esenciales hacen uso de los sistemas de seguridad que les resultan más factibles teniendo en cuenta el ambiente y las necesidades de la institución, con el objetivo de evitar ataques contra la información más sensible que se maneja en la entidad.

La Seguridad Informática tiene el objetivo de garantizar:

- **Confidencialidad:** la información o los activos informáticos son accedidos solo por las personas autorizadas. (Pfleeger, 2006)
- **Integridad:** los activos o la información solo pueden ser modificados por las personas autorizadas y de la forma autorizada. (Pfleeger, 2006)
- **Disponibilidad:** los activos informáticos son accedidos por las personas autorizadas en el momento requerido. (Pfleeger, 2006)

Se define como **sistema de información** a la colección integrada de componentes que incluyen procesos computacionales, procedimientos administrativos, equipos/hardware (servidores y redes), aplicaciones/software, base de datos y recursos humanos, con un propósito y contexto social. Estos sistemas permiten almacenar, procesar, analizar y diseminar información de apoyo a las operaciones, administración y toma de decisiones en una organización. (Casa, 2010)

Teniendo en cuenta las definiciones de Seguridad Informática y SI. En la presente investigación se definen a los **sistemas de seguridad informática** como herramientas que permiten gestionar la Seguridad Informática en SI.

Los sistemas de seguridad informática están destinados a la protección de los recursos lógicos; estableciendo las medidas necesarias para garantizar la confidencialidad, integridad y disponibilidad de la información. También se encargan de la gestión del control de acceso a la información y recursos que se gestionan en las instituciones. (Thais Figueras, 2012)

Sumado a los objetivos anteriormente citados, otro aspecto esencial para fortalecer la confidencialidad, disponibilidad e integridad de la información es el control de acceso. Según Charles P. Pflegger el **control de acceso** se encarga de limitar la actividad que los usuarios legítimos tienen en el sistema. Por tanto, es el proceso por el que se asegura que todos los accesos a sistemas y a sus recursos se realizan de forma controlada, y que sólo se pueden realizar aquellos accesos que hayan sido autorizados. (Pfleeger, 2006)

El **control de acceso** también se define como el proceso por el cual la entrada a los recursos del sistema, así como, a la información en el flujo de trabajo son regulados según determinadas políticas de seguridad y permitiendo el acceso solamente a entidades autorizadas. (Miguel Sánchez, 2006)

Teniendo en cuenta los conceptos citados anteriormente, en el marco de esta investigación el control de acceso a nivel de base de datos es el proceso encargado de limitar la actividad que los usuarios pueden realizar sobre los recursos (esquemas, tablas, funciones, triggers, secuencias, vistas). Para ello se establecen un conjunto de políticas de seguridad.

Para lograr un mejor establecimento en la gestión de políticas de control de acceso a base de datos se hace necesario granular los permisos sobre los recursos. Por ello se define como **granularidad** al conjunto de políticas de control que dictan a qué datos pueden tener acceso los usuarios. Cada elemento de la base de datos está marcado para mostrar sus limitaciones de acceso. Lo cual garantiza que el valor no sea modificado por un usuario no autorizado. (Pfleeger, 2006)

Teniendo en cuenta la definición anterior, se entiende como **granularidad en el establecimiento de políticas de control de acceso**, a la posibilidad de seccionar en partes más pequeñas los permisos sobre los recursos a nivel de datos.

Existen varios modelos de control de acceso en la actualidad, el presente trabajo se centra en el estudio del Modelo de Control de Acceso Basado en Roles (RBAC) pues es el que se implementa en el sistema Acaxia.

1.1.1 Modelo de Control de Acceso Basado en Roles

El Modelo de Control de Acceso Basado en Roles (RBAC[1]) está basado en la definición de un conjunto de elementos y de relaciones entre ellos. A nivel general describe un grupo de usuarios que pueden estar actuando bajo un conjunto de roles y realizando operaciones en las que utilizan un conjunto de recursos. RBAC añade la posibilidad de modelar una jerarquía de roles de forma que se puedan realizar generalizaciones y especializaciones en los controles de acceso y se facilite la modelización de la seguridad en sistemas complejos.

RBAC incluye un conjunto de sesiones donde cada sesión es la relación entre un usuario y un sub-conjunto de roles establecidos en el momento de activar la sesión. Cada usuario tiene asociada una o más sesiones. Los permisos disponibles para un usuario son el conjunto de permisos asignados a los roles que están activados en todas las sesiones del usuario. (Miguel Sánchez, 2006)

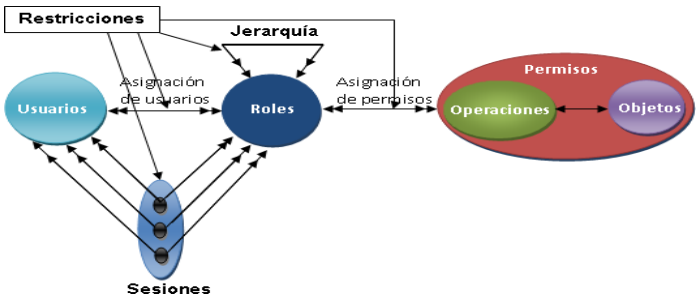


Figura 1. Elementos básicos de RBAC.
Fuente: (Miguel Sánchez, 2006)

RBAC permite expresar de forma sencilla y natural las políticas de accesos a los recursos. Posibilita la construcción jerárquica de estas políticas de acceso, por herencia o especialización. El modelo RBAC está apto para el desarrollo de aplicaciones de seguridad, pues tiene el potencial de reducir la complejidad y el coste de la administración en un entorno de seguridad.

El control de acceso es ampliamente utilizado en los marcos de trabajo o frameworks de desarrollo de software y en los sistemas informáticos encargados de la gestión de la seguridad. En la presente investigación se hace un estudio de algunos de los frameworks más utilizados actualmente a nivel mundial.

[1] **RBAC** por las siglas en inglés de **R**ole **B**ased **A**ccess **C**ontrol.

1.2 Las conexiones a base de datos en los marcos de trabajo y el sistema de seguridad Spring Security

En la presente investigación se hace necesario el estudio de varios frameworks de desarrollo y sistemas de seguridad informática, con el objetivo de analizar la forma en que cada uno de ellos administra las conexiones hacia la base de datos. Para ello se entiende por administración de conexiones a sustituir la lógica estática del acceso a datos, por una que permita mayor operatividad en cuanto a la forma de acceder a los datos. (Cuba, 2008)

Un framework es un conjunto de bibliotecas, herramientas y normas a seguir que ayudan a desarrollar aplicaciones. Está compuesto por varios segmentos/componentes que interactúan los unos con los otros. También permiten la reutilización de código y la estandarización del desarrollo. Esto posibilita que las aplicaciones puedan escribirse de manera más eficaz si utilizamos un framework adaptado al proyecto. (Lafosse, 2010)

Según la revista española Todo Linux, dos de los frameworks para el lenguaje PHP[2] más usandos son Symfony y Zend Framework. (linux, 2009)

En septiembre del 2012 el sitio web symfony.es realizó un estudio para analizar la popularidad relativa de los tres frameworks para PHP más importantes en diferentes regiones del mundo. Este estudio abarcó varios países como Alemania, Francia, Estados Unidos entre otros demostrando que dos de los frameworks más usados actualmente son Symfony y Zend Framework. (symfony.es, 2012)

Estos frameworks son soluciones muy robustas al alcance de todos los usuarios para facilitar el desarrollo de aplicaciones informáticas. Están construidos sobre tecnologías libres y presentan gran facilidad tanto de aprendizaje como de uso.

Otro de los marcos de trabajo más usados para el lenguaje Java es Hibernate por las grandes ventajas que brinda pues es 100% orientado a objetos, no necesita escribirse códigos en líneas de comando y está respaldado por una comunidad Open Source activa. (Martín, 2009)

Teniendo en cuenta estos planteamientos se decide analizar en la presente investigación:

Para el lenguaje PHP:

- Symfony.
- Zend Framework.

Para el lenguaje Java:

- Hibernate.

[2] **PHP:** por las siglas en inglés de **H**ypertext **P**re-processor.

El sistema de seguridad a estudiar es Spring Security por ser uno de los proyectos más maduros y ampliamente utilizado. Mantenido activamente, ya que, hoy en día se utiliza para asegurar numerosos entornos exigentes, incluyendo agencias gubernamentales, aplicaciones militares y de los bancos centrales. (SpringSource, 2013)

1.2.1 Symfony y la conexión a base de datos

Symfony es un framework diseñado para optimizar, gracias a sus características, el desarrollo de las aplicaciones web. Separa la lógica de negocio, la lógica del servidor y la presentación de la aplicación web. Proporciona varias herramientas y clases encaminadas a reducir el tiempo de desarrollo de una aplicación web compleja. Además, automatiza las tareas más comunes, permitiendo al desarrollador dedicarse por completo a los aspectos específicos de cada aplicación. (Web, 2013)

Symfony admite el uso de los gestores de base de datos SQLite, MySQL, PostgreSQL, Oracle y SQLServer de manera independiente. La información mínima que necesita Symfony para realizar peticiones a la base de datos es su nombre, las credenciales para acceder (usuario, contraseña) y el tipo de base de datos.

Los datos de conexión dependen del entorno[3] de ejecución. Cada aplicación también puede redefinir esta configuración, lo que es útil cuando se quiere disponer de diferentes políticas de seguridad. De esta forma, es posible poseer diferentes usuarios de base de datos con privilegios diferentes para cada aplicación.

En cada entorno también es posible definir varias conexiones diferentes. Cada conexión siempre hace referencia al esquema de datos del mismo nombre. Las opciones de conexión a la base de datos también se pueden establecer manualmente.

1.2.2 Zend Framework y la conexión a base de datos

Zend Framework es un marco de código abierto para desarrollar aplicaciones web con PHP5. Zend Framework es una implementación que usa el paradigma orientado a objetos. La estructura de sus componentes es algo único; cada componente está construido con una baja dependencia de otros componentes. (ZendFramework, 2010)

[3] **Entorno de Ejecución:** las aplicaciones de Symfony se pueden ejecutar en diferentes entornos. Todos los entornos comparten el mismo código pero pueden tener configuraciones completamente diferentes. Cuando se crea una aplicación, Symfony crea por defecto tres entornos: producción (prod), pruebas (test) y desarrollo (dev). También es posible añadir cualquier nuevo entorno que se considere necesario. En cierta forma, un entorno y una configuración son sinónimos.

Zend Framework admite entre otros los gestores MicrosoftSQL Server, MySQL, PostgreSQL y SQLite. Los parámetros de conexión[4] a base de datos son especificados en el archivo Zend_Config_Ini. El framework es capaz de soportar múltiples conexiones y no es responsable de los roles que existen en la base de datos, ni de los privilegios que estos tienen sobre los objetos de la misma.

1.2.3 Hibernate y la conexión a base de datos

Hibernate es un framework robusto para el mapeo de objetos en una base de datos relacional para ambientes de Java. Esta herramienta provee utilidades para consultas y captura de datos, reduciendo el tiempo que habría que emplear para el manejo de los mismos mediante SQL[5] y JDBC[6]. El framework es útil con modelos orientados a objetos cuya lógica de negocios reside en la capa intermedia. (Hibernate, 2009)

Hibernate posee un pool de conexiones integrado que se encarga de asignar las conexiones a la base de datos, además el framework tiene soporte para otros dos más desarrollados por terceros. Para la configuración de Hibernate se puede utilizar el archivo ***hibernate.properties*** o ***hibernate.cfg.xml***, aquí es donde se van a introducir los datos de conexión. Al configurar estos archivos se ha establecido un ***SesionFactory***; el cual es una factoría global responsable de una base de datos en particular. Para realizar conexiones distintas es necesario configurar otros ***SessionFactorys***. Hibernate no se encarga de gestionar los permisos de los roles sobre la base de datos.

1.2.4 Spring Security y la conexión a base de datos

Spring Security es un sistema de Seguridad Informática elaborado en Java. Este sistema de seguridad no se encarga de gestionar las conexiones hacia la base de datos, desde el mismo se pueden abrir conexiones hacia el gestor pero incrustando la sentencia directamente en el código. No maneja ningún fichero desde el cual se pueda acceder a los parámetros de conexión. Es recomendable para realizar el acceso hacia la base de datos cuando se utilice Spring Security emplear un framework de acceso a datos como Hibernate. Este sistema de seguridad almacena por defecto los roles en un XML[7], aunque puede ser modificado para almacenarlos en una tabla del

[4] **Parámetros de conexión:** dirección ip, gestor, puerto, base de datos, usuario y contraseña de rol de base de datos.
[5] **SQL** por las siglas en inglés de **S**tructured **Q**uery **L**anguage.
[6] **JDBC** por las siglas en inglés de **J**ava **D**ata**b**ase **C**onnectivity.
[7] **XML** por las siglas en inglés de E**x**tensible **M**arkup **L**anguage.

gestor. A estos roles les establece privilegios a nivel de aplicación, pero no se encarga de gestionar la seguridad en la base de datos. (Mularien, 2010)

A continuación se muestra la Tabla 1 con los indicadores de comparación seleccionados:

Tabla 1. Tabla de indicadores de comparación.
Fuente: elaboración propia.

Software	Soporta varias conexiones	Forma de establecer conexiones	Gestionar privilegios del rol en el gestor de base de datos
Symfony	Soporta múltiples conexiones.	Los parámetros de conexión se especifican en el archivo database.yaml y también pueden especificarse en una tabla de la base de datos. Es posible disponer de diferentes usuarios de base de datos con privilegios diferentes para cada aplicación.	No gestiona privilegios del rol en el gestor de base de datos.
Zend Framework	Soporta múltiples conexiones.	Los parámetros de conexión a la base de datos se especifican en el fichero Zend_Config_Ini.	No gestiona privilegios del rol en el gestor de base de datos.
Hibernate	Soporta múltiples conexiones.	Los parámetros de conexión a la base de datos se especifican en los ficheros *hibernate.properties* o *hibernate.cfg.xml.*	No gestiona privilegios del rol en el gestor de base de datos.
Spring Security	No maneja las conexiones a la base de datos.	Se recomienda utilizar un framework de acceso a datos.	No gestiona privilegios del rol en el gestor de base de datos.

1.2.5 Valoración del estudio realizado

El estudio de los marcos de trabajo anteriormente mencionados: Symfony, Zend Framework e Hibernate y el sistema de seguridad Spring Security arroja que los frameworks poseen características comunes con Sauxe en cuanto a la forma en que gestionan las conexiones a la base de datos teniendo en cuenta que:

- Todos ellos configuran un archivo de conexión en el que se especifican los parámetros de conexión para acceder a la base de datos.
- No se responsabilizan de gestionar los privilegios de los roles sobre los recursos de la base de datos.
- No se establecen niveles de conexión que fortalezcan el control de acceso a la base de datos.
- El sistema de seguridad Spring Security no maneja las conexiones a la base de datos por defecto, para ello es más recomendable integrarle un framework de acceso a datos como Hibernate. Este sistema no se encarga de gestionar la seguridad en la capa de datos.

Estos sistemas aunque tienen puntos comunes con Sauxe, gestionan el control de acceso a la base de datos de acuerdo a sus características propias. Son insuficientes de forma individual para responder a la problemática planteada en la presente investigación.

1.3 Modelo de desarrollo de software

El modelo de desarrollo de software que se utiliza en el departamento de Tecnología es el definido para el CEIGE. Este modelo posee las siguientes características:

Orientado a componentes: sistema compuesto por varios componentes desarrollados de manera independiente pero que al unirlos constituyen dicho sistema.

Iterativo e incremental: plantea que ese mismo sistema puede tener tantas iteraciones como sean necesarias y que en cada versión se obtendrá un incremento y mejoramiento de las funcionalidades del sistema.

Incluye la especificación de las actividades de cada una de las fases del ciclo de vida de los proyectos del CEIGE teniendo en cuenta los procesos de CMMI[8] nivel dos para la Universidad de las Ciencias Informáticas. (CEIGE, 2013)

1.3.1. Flujo de trabajo

En el marco de la presente investigación el flujo de trabajo comprende las fases que se muestran en la Figura 2.

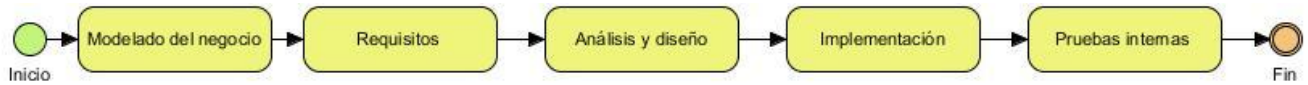


Figura 2. Flujo de trabajo.
Fuente: (CEIGE, 2013)

[8] **CMMI**: por las siglas en inglés de **C**apability **M**aturity **M**odel **I**ntegration.

Durante el flujo de trabajo en la fase modelado del negocio se emplea modelo de dominio, no se utiliza modelo de procesos debido a que no existen en el departamento procesos de negocio definidos. En el modelo de dominio el artefacto que se genera es el modelo conceptual, el cual describe los aspectos del dominio, identificándose los principales elementos físicos o lógicos del negocio que ayuden a entender el problema y que generalmente se presentan como clases. El modelo conceptual explica cuáles son y cómo se relacionan los conceptos relevantes en la descripción del dominio de un problema, identificando atributos y asociaciones existentes entre ellos. En la fase de requisitos se identifican los requisitos funcionales, se describen los mismos y se elaboran los prototipos de interfaces de usuario asociados a ellos. Se identifican también en esta fase los requisitos no funcionales con los que contará el componente. Durante la fase de análisis y diseño se define el estilo arquitectónico y el patrón arquitectónico que determinan la estructura fundamental del componente. En esta fase se genera el modelo de diseño, artefacto conformado por los diagramas de clases del diseño con estereotipos web y los diagramas de secuencia. Por último se realiza el modelo de datos, se elaboran los diseños de casos de prueba y el diagrama de despliegue. En la fase de implementación se elabora el diagrama de componentes, se definen los estándares de codificación a utilizar y se implementan los requisitos funcionales identificados con sus correspondientes interfaces. En las pruebas internas se realizan pruebas de caja negra mediante la técnica de partición equivalente y las pruebas de caja blanca utilizando la técnica del camino básico, también se resuelven las no conformidades detectadas durante la ejecución de estas pruebas.

1.4 Ingeniería de Requisitos

La **Ingeniería de Requisitos** se define como "*el uso sistemático de procedimientos, técnicas, lenguajes y herramientas para obtener el análisis, documentación, evolución continua de las necesidades del usuario y la especificación del comportamiento externo que satisfaga las necesidades del usuario."* (Pressman, 2005)

La Ingeniería de Requisitos permite garantizar la correcta descripción de los requisitos para evitar futuros errores y reducir el tiempo en el desarrollo de un software. Tiene el objetivo de que los mismos obtengan un estado óptimo antes de alcanzar la fase de diseño.

Existen varias técnicas para llevar a cabo el proceso de **captura de requisitos**. Es tarea del equipo de desarrollo determinar cuál o cuáles son las más factibles a utilizar para el producto a desarrollar. Entre algunas de estas técnicas se pueden mencionar:

- **Entrevistas:** es un medio tradicional de obtención de requisitos. La entrevista es un método muy efectivo que permite conocer los problemas de los clientes y encontrar requisitos generales. Para aplicar este método es necesario conocer la forma en que se debe de realizar una entrevista para lograr una buena comunicación entre el entrevistador y el entrevistado. (Ganesh, 2008)
- **Prototipos:** es la representación o visualización de parte del sistema. Es una herramienta valiosa para clarificar requisitos confusos. Proveen a los usuarios un contexto para entender mejor qué información necesitan proporcionar. (Ganesh, 2008)
- **Tormenta de ideas:** esta técnica es usada para generar nuevas ideas y encontrar la solución a cuestiones específicas. Es muy común en los comienzos del proceso de ingeniería de requisitos. (Ganesh, 2008)
- **Observación:** este método consiste en la identificación de requisitos cuando se observan a las personas haciendo su trabajo diario. Es muy usado para encontrar requisitos adicionales cuando el usuario es incapaz de explicar los requisitos que necesita para el nuevo sistema. (Ganesh, 2008)

Después de analizar las definiciones de cada una de las técnicas de captura de requisitos anteriormente citadas se puede pensar que son todas factibles si se emplean combinaciones de ellas, ya que por si solas no son suficientes para realizar un buen proceso de captura de requisitos. El uso de estas técnicas está estrechamente relacionado con las características del proyecto en el que vayan a ser aplicadas.

Existen varias **técnicas para validar los requisitos,** las cuales se aplican con el objetivo de examinar las especificaciones para asegurar que todos los requisitos han sido establecidos sin ambigüedad, sin inconsistencias, sin omisiones y que los errores detectados hayan sido corregidos. (Pressman, 2005) Algunas de esta técnicas son:

- **Revisiones:** los requisitos son analizados sistemáticamente por un equipo de revisores, en busca de anomalías y/u omisiones. (Sommerville, 2005)
- **Prototipos:** se muestra un modelo ejecutable del sistema a los usuarios finales y los clientes, para que puedan ver si dicho modelo cumple con sus necesidades reales. (Sommerville, 2005)
- **Generación de casos de prueba:** la elaboración de casos de prueba puede revelar los problemas con que puede contar un requisito y es parte fundamental de la programación externa. (Sommerville, 2005)

1.5 Arquitectura de software

La arquitectura de software es una disciplina importante puesto que los sistemas de software crecen tanto que se hace complicado que sean diseñados, especificados y entendidos por un solo individuo. (Kazman, 1996)

La mayoría de los autores coinciden en que una arquitectura de software define la estructura del sistema. Esta estructura se constituye de componentes, que nacen de la noción de abstracción, cumpliendo funciones específicas e interactuando entre sí. (Erica Camacho, 2004) En la disciplina arquitectura de software se determina el estilo arquitectónico y el patrón arquitectónico, donde:

Un **estilo arquitectónico** según Buschmann se define como una familia de sistemas de software en términos de su organización estructural. Expresa componentes y las relaciones entre estos, con las restricciones de su aplicación, así como también las reglas para su construcción. Así mismo, se considera como un tipo particular de estructura fundamental para un sistema de software, conjuntamente con un método asociado que especifica cómo construirlo. Éste incluye información acerca de cuándo usar la arquitectura que describe, sus invariantes y especializaciones, así como las consecuencias de su aplicación. (Erica Camacho, 2004) Los principales estilos arquitectónicos, así como una pequeña descripción de ellos son los que se muestran en la Tabla 2.

Tabla 2. Estilos arquitectónicos.
Fuente: (Erica Camacho, 2004)

Estilo	Descripción
Datos Centralizados	Sistemas en los cuales cierto número de clientes accede y actualiza datos compartidos de un repositorio de manera frecuente.
Flujo de Datos	El sistema es visto como una serie de transformaciones sobre piezas sucesivas de datos de entrada. El dato ingresa en el sistema, y fluye entre los componentes, de uno en uno, hasta que se le asigne un destino.
Máquinas Virtuales	Simulan alguna funcionalidad que no es nativa al hardware o software sobre el que está implementado.
Llamada y Retorno	El sistema se constituye de un programa principal que tiene el control del sistema y varios subprogramas que se comunican con éste mediante el uso de llamadas.
Componentes Independientes	Consiste en un número de procesos u objetos independientes que se comunican a través de mensajes.

Para poder entender que es un patrón arquitectónico se hace necesario primero conocer que se entiende por patrón.

Buschmann plantea que un **patrón** es una solución probada que se puede aplicar con éxito a un determinado tipo de problema que aparece con frecuencia. Consta de tres partes:

- **Contexto:** situación de diseño en la que aparece un problema de diseño.

- **Problema:** conjunto de fuerzas que aparecen repetidamente en el contexto.
- **Solución:** configuración que equilibra estas fuerzas.

Partiendo de esta definición se puede determinar que los **patrones arquitectónicos** expresan el esquema de organización estructural fundamental para sistemas de software. Proveen un conjunto de subsistemas predefinidos, especifica sus responsabilidades e incluye reglas y pautas para la organización de las relaciones entre ellos. Propone que son plantillas para arquitecturas de software concretas, que especifican las propiedades estructurales de una aplicación. (Buschmann, 1996) La Tabla 3 muestra algunos patrones arquitectónicos así como una breve descripción de acuerdo a Buschmann.

Tabla 3. Patrones arquitectónicos.
Fuente: (Erica Camacho, 2004)

Patrón	Descripción
Layers	Consiste en estructurar aplicaciones que pueden ser descompuestas en grupos de subtareas, las cuales se clasifican de acuerdo a un nivel particular de abstracción.
Pipes and Filters	Provee una estructura para los sistemas que procesan un flujo de datos. Cada paso de procesamiento está encapsulado en un componente filtro (filter). El dato pasa a través de conexiones (pipes), entre filtros adyacentes.
Blackboard	Aplica para problemas cuya solución utiliza estrategias no determinísticas. Varios subsistemas ensamblan su conocimiento para construir una posible solución parcial o aproximada.
Model-View-Controler	Divide una aplicación interactiva en tres componentes. El modelo (model) contiene la información central y los datos. Las vistas (view) despliegan información al usuario. Los controladores (controlers) capturan la entrada del usuario. Las vistas y los controladores constituyen la interfaz del usuario.
Reflection	Provee un mecanismo para sistemas cuya estructura y comportamiento cambia dinámicamente. Soporta la modificación de aspectos fundamentales como estructuras tipo y mecanismos de llamadas a funciones.

1.6 Patrones de diseño

Un patrón de diseño provee un esquema para refinar los subsistemas o componentes de un sistema de software, o las relaciones entre ellos. Describe la estructura recurrente de los componentes en comunicación, que resuelve un problema general de diseño en un contexto particular (Buschman, 1996).

Los patrones de diseño se dividen en dos vertientes, los patrones **GRASP**[9] y los **GoF**[10]

Patrones GRASP: describen los principios fundamentales de la asignación de responsabilidades a objetos, expresados en formas de patrones. (Tedeschi, 2013) Algunos de los patrones GRASP más utilizados son:

Tabla 4. Patrones de diseño GRASP.
Fuente: elaboración propia.

Patrón	Descripción
Experto	Asigna una responsabilidad al experto en información: la clase que posee la información necesaria para cumplir con la responsabilidad. (Larman, 1999)
Creador	Asigna a la clase B la responsabilidad de crear una instancia de clase A. (Larman, 1999)
Controlador	Asigna la responsabilidad de administrar un mensaje de eventos del sistema a una clase. (Larman, 1999)
Bajo acoplamiento	Asigna responsabilidades de modo que se mantenga bajo acoplamiento. (Larman, 1999)
Alta cohesión	Asigna responsabilidad de modo que se mantenga alta cohesión. (Larman, 1999)

Patrones GoF: se dieron a conocer a principios de los años 90 con el libro "Design Patterns. Elements of Reusable Object-Oriented Software"[11]. En dicho libro se hace una recopilación de 23 patrones de diseño comunes, clasificados en tres grupos de acuerdo a su naturaleza:

- **Patrones Creacionales:** inicialización y configuración de objetos. (Prieto, 2009)
- **Patrones Estructurales:** separan la interfaz de la implementación. Se ocupan de cómo las clases y objetos se agrupan, para formar estructuras más grandes. (Prieto, 2009)
- **Patrones de Comportamiento:** más que describir objetos o clases, describen la comunicación entre ellos. (Prieto, 2009)

Algunos de los patrones de diseño GoF son los que se muestran en la siguiente en la Tabla 5.

Tabla 5. Patrones de diseño GoF.
Fuente: elaboración propia.

Patrón	Descripción	Grupo
Fachada	Proporciona una interfaz unificada de alto nivel que, representando a todo un subsistema, facilite su uso. (Larman, 2003)	Estructural
Factoría	Proporciona una interfaz para la creación de familias de objetos interdependientes o interrelacionados, sin especificar sus clases concretas (Larman, 2003)	Creacional

[9] **GRASP**: por las siglas en inglés de **G**eneral **R**esponsibility **A**ssignment **S**oftware **P**atterns.
[10] **GoF:** por las siglas en inglés de **G**ang **o**f **F**our, en español Banda de los Cuatro.
[11] Traducido al español como: Patrones de Diseño. Elementos de software reusable orientado a objetos.

Singleton	Garantiza que una clase sólo tiene una única instancia, proporcionando un punto de acceso global a la misma. (Larman, 2003)	Creacional
Mediador	Encapsula la forma en que interactúan un grupo de objetos, promoviendo así un acoplamiento débil al evitar las referencias explícitas entre los objetos. (Buschmann, 2007)	Comportamiento

1.7 Pruebas

Pressman, declara que las pruebas de software son un elemento crucial para garantizar la calidad del producto y permiten validar las especificaciones, el diseño y la programación. Estas tienen como objetivo, además de descubrir errores, medir el grado en que el software cumple con los requerimientos definidos. (Pressman, 2005) Existen dos vertientes de pruebas, las pruebas de caja negra y las pruebas de caja blanca.

Las **pruebas de caja negra**, denominada también pruebas de comportamiento o pruebas funcionales, permiten obtener conjuntos de entradas que ejerciten los requisitos funcionales del software, complementándose con las pruebas de caja blanca, obteniendo errores en las siguientes categorías (Pressman, 2005):

- Funciones incorrectas o inexistentes.
- Errores en la interfaz.
- Errores en la estructura de datos.
- Rendimiento.
- Inicialización y terminación.

Para las pruebas de caja negra existen varias técnicas, algunas de ellas son:

- **Partición Equivalente:** permite examinar los valores válidos e inválidos de las entradas existentes en el software. Además descubre de forma inmediata una clase de errores que, de otro modo, requerirían la ejecución de muchos casos antes de detectar el error genérico. Esto permite reducir el número de casos de prueba a elaborar. (Pressman, 2005)
- **Análisis de valores límites:** los errores tienden a darse más en los límites del campo de entrada que en el centro. Esta es una técnica que complementa la partición equivalente. En lugar de seleccionar cualquier elemento de una clase de equivalencia, lleva a elección de casos de prueba en los extremos de la clase. (Pressman, 2005)
- **Prueba de comparación:** este tipo de pruebas se emplea cuando la fiabilidad del software es algo crítico, (por ejemplo cuando se desarrolla para aeronaves o plantas nucleares) varios

equipos de ingeniería del software desarrollan versiones independientes de la misma aplicación, usando los mismos requisitos. Todas las versiones son probadas con los mismos datos, para asegurar que todas proporcionan una salida idéntica. (Pressman, 2005)

- **Prueba de la tabla ortogonal:** esta prueba puede aplicarse a problemas en que el dominio de entrada es relativamente pequeño pero demasiado grande para solicitar pruebas exhaustivas. El método de la tabla ortogonal es útil al encontrar errores asociados con fallos localizados. (Pressman, 2005)

Analizadas todas esta técnicas se concluye que todas son útiles siempre que se escoja bien el contexto en el cual se van a aplicar. Cada proyecto de software tiene características muy particulares.
Otro tipo de **pruebas** que se aplica con frecuencia al software son las de **caja blanca**, este es un método de diseño de casos de prueba que usa la estructura de control del diseño procedimental para obtener los casos de prueba. (Pressman, 2005) Mediante los métodos de prueba de caja blanca:

- Se garantiza que se recorra por lo menos una vez los caminos independientes de cada módulo. (Pressman, 2005)
- Se ejecuten todas las decisiones lógicas en sus opciones verdadera y falsa. (Pressman, 2005)
- Se ejerciten todos los bucles en sus límites. (Pressman, 2005)
- Se usen las estructuras internas de datos para asegurar su validez. (Pressman, 2005)

Existen varias técnicas de pruebas de caja blanca, algunas de ellas son:

- **Camino básico:** permite obtener una medida de la complejidad lógica de un diseño procedimental y usar esa medida como guía para la definición de un conjunto básico de caminos de ejecución. Para obtener el conjunto de caminos independientes se construye el grafo de flujo asociado y se calcula su complejidad ciclomática. Los casos de pruebas obtenidos garantizan que se ejecute al menos una vez cada sentencia del programa. (Pressman, 2005)
- **Prueba de condición:** es un método de diseño de casos de prueba que ejercita las condiciones lógicas contenidas en el módulo de un programa. El propósito de esta técnica es detectar no solo errores en las condiciones, sino también otro tipo de errores. (Pressman, 2005)
- **Pruebas de flujos de datos:** selecciona caminos de prueba de un programa de acuerdo con la ubicación de las definiciones y los usos de las variantes del programa. (Pressman, 2005)
- **Prueba de bucles:** se centra exclusivamente en la validez de las construcciones de bucles. Se pueden definir cuatro clases diferentes de bucles: bucles simples, bucles concatenados, bucles anidados y bucles no estructurados. (Pressman, 2005)

Inicialmente puede parecer que una prueba de caja blanca profunda nos puede llevar a tener programas correctos, definiendo todos los caminos lógicos, generar casos de prueba que examinen exhaustivamente la lógica del programa. Desgraciadamente, estas pruebas incluso para programas pequeños el número de caminos lógicos que genera puede ser enorme. Las pruebas de caja blanca no se deben desechar como impracticables, se deben escoger una serie de caminos importantes a ejecutar. (Pressman, 2005)

Métricas de software

Luego de realizar el diseño es necesario mejorar la calidad del trabajo llevado a cabo a nivel del proyecto, para ello se emplean métricas. La IEEE[12] define como **métrica** *"una medida cuantitativa del grado en que un sistema, componente o proceso posee un atributo dado*". (IEEE, 2007)

El uso de métricas en un producto de software es esencial para entender la calidad del software. Estas brindan la posibilidad de evaluar la calidad del producto a partir de indicadores permitiendo ver la eficacia del proceso de desarrollo.

Según Pressman la calidad del diseño se puede evaluar aplicando métricas básicas para el diseño orientado a objetos. (Pressman, 2005) Los atributos de calidad involucrados son:

- **Responsabilidad:** responsabilidad asignada a una clase en un marco de modelado de un dominio o concepto, de la problemática propuesta. (Baryolo, 2010)
- **Complejidad de implementación:** grado de complejidad que posee la implementación de un diseño de clases específico. (Baryolo, 2010)
- **Reutilización:** grado de reutilización presente en una clase o estructura de clases, dentro de un diseño de software. (Baryolo, 2010)
- **Acoplamiento:** las conexiones físicas entre los elementos del diseño orientado a objeto, representan el acoplamiento dentro de un sistema orientado a objeto. (Pressman, 2005)
- **Complejidad del mantenimiento:** grado de esfuerzo necesario a realizar para desarrollar un arreglo, una mejora o una rectificación de algún error de un diseño de software. Puede influir indirecta, pero fuertemente en los costes y la planificación del proyecto. (Baryolo, 2010)
- **Cantidad de pruebas:** número o grado de esfuerzo para realizar las pruebas de calidad del producto diseñado. (Baryolo, 2010)

[12] **IEEE:** por las siglas en inglés de **I**nstitute of **E**lectrical and **E**lectronics **E**ngineers.

La métrica **Tamaño Operacional de Clases (TOC)** se encarga de contar el número de métodos u operaciones (de instancia privada y heredada) que están encapsulados dentro o por una clase. Evalúa los siguientes atributos de calidad:

Tabla 6. Atributos de calidad evaluados por la métrica TOC.
Fuente: elaboración propia.

Atributo de calidad	Modo en que lo afecta
Responsabilidad	Aumento del TOC provoca aumento de la responsabilidad asignada a la clase.
Complejidad de implementación	Aumento del TOC provoca aumento de la complejidad de implementación de la clase.
Reutilización	Aumento del TOC provoca disminución del grado de reutilización de la clase.

Para la evaluación de dichos atributos de calidad, se definen los siguientes criterios y categorías de evaluación:

Tabla 7. Criterios de evaluación de la métrica TOC.
Fuente: elaboración propia.

Atributo	Categoría	Criterio
Responsabilidad	Baja	<=promedio
	Media	promedio> y< 2*promedio
	Alta	>2*promedio
Complejidad de implementación	Baja	<=promedio
	Media	promedio> y< 2*promedio
	Alta	>2*promedio
Reutilización	Baja	>2*promedio
	Media	promedio> y< 2*promedio
	Alta	<=promedio

La métrica **Relaciones entre Clases (RC)** se encarga de contar el número de relaciones de uso de una clase con otra y evalúa los siguientes atributos de calidad:

Tabla 8. Atributos de calidad evaluados por la métrica RC.
Fuente: elaboración propia.

Atributo de Calidad	Modo en que lo afecta
Acoplamiento	Aumento del RC provoca aumento del acoplamiento de la clase.
Complejidad de mantenimiento	Aumento del RC provoca aumento de la complejidad de implementación de la clase.
Reutilización	Aumento del RC provoca disminución del grado de reutilización de la clase.
Cantidad de pruebas	Aumento del RC provoca aumento de la cantidad de pruebas de unidad necesarias para probar una clase.

Para la evaluación de dichos atributos de calidad, se definieron los criterios y categorías de evaluación:

Tabla 9. Criterios de evaluación para la métrica RC.
Fuente: elaboración propia.

Atributo	Categoría	Criterio
Acoplamiento	Ninguno	0
	Baja	1
	Media	2
	Alta	>2
Complejidad de mantenimiento	Baja	<=promedio
	Media	promedio> y< 2*promedio
	Alta	>2*promedio
Reutilización	Baja	>2*promedio
	Media	promedio> y< 2*promedio
	Alta	<=promedio
Cantidad de pruebas	Baja	<=promedio
	Media	promedio> y< 2*promedio
	Alta	>2*promedio

La métrica **Árbol de Profundidad de Herencia (APH)** se define como la longitud máxima desde el nodo hasta la raíz del árbol. A medida que crece el APH, es más probable que las clases de niveles inferiores hereden muchos métodos. Esto da lugar a posibles dificultades cuando se intenta predecir el comportamiento de una clase. Una jerarquía de clases profunda (con un valor grande de APH) lleva también a una mayor complejidad de diseño. Por el lado positivo, los valores grandes de APH implican que se pueden reutilizar muchos métodos. (Pressman, 2005)

1.8 Herramientas y tecnologías

Las herramientas y tecnologías estudiadas y definidas por el CEIGE para el desarrollo de software de gestión son:

- El **servidor web Apache 2.0** está desarrollado bajo el criterio Open Source. Funciona en una multitud de sistemas operativos, lo que lo hace prácticamente universal. Es un servidor altamente configurable de diseño modular. Implementa LDAP[13] para la autenticación. (Ciberaula, 2010)
- El gestor de base de datos **PostgreSQL 9.0** es un sistema de gestión de bases de datos relacional con su código fuente disponible libremente. Utiliza un modelo cliente/servidor. (PosgreSQL, 2010)

[13] **LDAP** por las siglas en inglés de **L**ightweight **D**irectory **A**ccess **P**rotocol.

- Las herramientas CASE[14] son herramientas individuales para ayudar al desarrollador de software o administrador de proyecto durante una o más fases del desarrollo de software. (B.Terry, 1990) **Visual Paradigm 8.0** es una herramienta CASE de diseño que soporta el estándar de Lenguaje de Modelado Unificado (UML[15]) diseñado para ayudar al desarrollo de software. Es compatible con los equipos de desarrollo de software en la captura de requisitos, software de planificación, el código de ingeniería, el modelado de clases, modelado de datos. (Paradigm, 2013)
- **RapidSVN 1.6.6** es un cliente gráfico para Subversion. Es fácil de usar, tanto para los que conocen Subversion como para los que empiezan, pudiendo acceder a direcciones SVN, subir y descargar contenido y sincronizarlo con el servidor original, comprobar su estado, crear y fusionar direcciones. (López, 2013)
- El IDE[16] **NetBeans 7.0** es un entorno de desarrollo, una herramienta para que los programadores puedan escribir, compilar, depurar y ejecutar programas. Está escrito en Java pero puede servir para cualquier otro lenguaje de programación. Es un producto libre y gratuito sin restricciones de uso. (NetBeans, 2012)

1.9 Frameworks de desarrollo

- **Zend Framework 1.11** está desarrollado por Zend bajo el criterio Open Source para PHP 5.0+. Implementa el patrón Modelo-Vista-Controlador y es completamente orientado a objetos. Esta arquitectura es de acoplamiento flexible lo cual permite a los desarrolladores utilizar cualquier componente. (Framework, 2013)
- **Doctrine 1.2.2** es un mapeador de objeto relacional (ORM[17]) para PHP 5.0+ que se encuentra en la parte superior de la capa de abstracción de base de datos. Permite escribir las consultas de base de datos en un dialecto orientado a objetos de propiedad SQL, que mantiene la flexibilidad sin necesidad de duplicar código innecesario. (Doctrine, 2013)
- **ExtJS 2.2** es una biblioteca de JavaScript para el desarrollo de aplicaciones web interactivas. Es capaz de flexibilizar el manejo de componentes de la página como el DOM[18], peticiones AJAX[19],

[14] **CASE** por las siglas en inglés de **C**omputer **A**ided **S**oftware **E**ngineering.
[15] **UML** por las siglas en inglés de **U**nified **M**odeling **L**anguage.
[16] **IDE:** por sus siglas en inglés de **I**ntegrated **D**evelopment **E**nvironment.
[17] **ORM:** por las siglas en inglés de **O**bject **R**elation **M**apper.
[18] **DOM**: por las siglas en inglés de **D**ocument **O**bject **M**odel.
[19] **AJAX:** por las siglas en inglés de **A**synchronous **J**avaScript **A**nd **X**ML.

DHTML[20], tiene la ventaja de crear interfaces de usuario bastante funcionales compatibles con la mayoría de los navegadores. (ExtJS, 2012)

- **Sauxe 1.5** contiene un conjunto de componentes reutilizables, logrando una mayor estandarización, flexibilidad, integración y agilidad en el proceso de desarrollo. Soluciona varios escenarios o aspectos arquitectónicos como gestión y configuración dinámica de caché, acceso a base de datos a través de una capa de abstracción, gestión dinámica de trazas entre otros. (Baryolo, 2010)

1.10 Lenguajes de modelado y desarrollo

Los lenguajes de modelado y desarrollo son notaciones en su mayoría visuales, que intentan representar un sistema de software a un nivel mucho más alto que los lenguajes de programación. (Definición, 2008) Los lenguajes de modelado y desarrollo a utilizar definidos por el CEIGE son:

- El **Lenguaje Unificado de Modelado 2.0** (UML[21]) es un lenguaje visual para especificar, construir y documentar los artefactos de sistemas intensivos en software. Este lenguaje se ha convertido en el lenguaje aceptado universalmente para los planos de diseño de software. (Larman, 1999)
- **PHP 5.2.6** es un lenguaje interpretado de alto nivel, especialmente pensado para desarrollos web y el cual puede ser incrustado en páginas HTML[22]. La mayoría de su sintaxis es similar a los lenguajes C, Java y Perl y es fácil de aprender. La meta de este lenguaje es permitir escribir a los programadores páginas web más dinámicas de una manera rápida y fácil. (PHP, 2013)

Conclusiones del capítulo.

En el capítulo se abordaron los conceptos fundamentales referentes a la Seguridad Informática. Se estudió un sistema de seguridad y algunos de los frameworks más utilizados, con el objetivo de estudiar y analizar la forma en que estos manejan las conexiones a la base de datos. Las conclusiones del estudio realizado muestran que estos sistemas no resuelven la problemática planteada en el marco de trabajo Sauxe. Al ser el sistema Acaxia el encargado de brindar seguridad a los sistemas que se le suscriban, se decide desarrollar sobre él un componente de gestión de políticas de control de acceso a nivel de base de datos. Se definió el flujo de trabajo, así como las herramientas y tecnologías para el desarrollo del componente.

[20] **DHTML:** por las siglas en inglés de **D**inamic **H**yper**T**ext **M**arkup **L**anguage.
[21] **UML:** por sus siglas en inglés de **U**nified **M**odeling **L**anguage.
[22] **HTML:** por sus siglas en inglés **H**yper**T**ext **M**arkup **L**anguage.

Capítulo 2: Propuesta de solución

Introducción

En el presente capítulo se realiza una propuesta de solución para el componente gestión de políticas de control de acceso a nivel de base de datos en el sistema Acaxia. Se describen los requisitos funcionales y no funcionales con que contará el componente. Se muestran los prototipos de interfaz de usuario asociados a cada uno de los requisitos funcionales. Se describe el estilo y patrón arquitectónico utilizado, así como los patrones de diseño. Se muestran los diferentes artefactos generados durante la fase de análisis y diseño.

2.1 Propuesta de solución

Con el componente se pretende aumentar la granularidad en el establecimiento de un conjunto de políticas para controlar el acceso a los recursos en el sistema Acaxia.

Con esta solución se establecen privilegios sobre los roles de base de datos, los que están determinados por el tipo de conexión que se esté utilizando.

En dicho componente se establecen cuatro niveles de conexión:

- **Conexión por defecto o a nivel de aplicación:** este modo de conexión es el que se utiliza en Acaxia por defecto. Consiste en la configuración de un fichero de texto con los parámetros necesarios para establecer la conexión a la base de datos. Al utilizar este tipo de configuración la aplicación realizará todas las transacciones hacia la base de datos empleado los parámetros existentes en el fichero relacionados con la conexión por defecto. Al establecer esta conexión, va a existir en la base de datos un rol de login el cual va a poseer privilegios sobre todos los recursos de la misma, de tal manera que cuando un usuario realiza una transacción desde cualquier lugar de la aplicación esta va a ser ejecutada por este rol de login.
- **Conexión a nivel de sistema:** este modo de conexión consiste en establecer en un fichero de texto los parámetros de conexión por cada SI que se encuentra suscrito a Acaxia. Para utilizar este tipo de conexión hay que crear en la base de datos por cada sistema suscrito un rol de login con los permisos necesarios. Siempre que se realiza una transacción a la base de datos; sin importar el usuario que la efectúa, se establece la conexión utilizando dichos parámetros; teniendo en cuenta el sistema desde el cual se hizo la solicitud de transacción.
- **Conexión a nivel de rol:** este modo de conexión consiste en crear en la base de datos un rol de login por cada rol existente en la aplicación con los permisos necesarios sobre los recursos para

ejecutar las acciones que tiene asignadas el rol de aplicación. Cada vez que un usuario realiza una transacción hacia la base de datos, se va a realizar a través del rol de base de datos correspondiente al rol que tiene asignado en la aplicación.

- **Conexión a nivel de usuario:** consiste en crear un rol de grupo y un rol de login en la base de datos por cada rol y usuario existentes en la aplicación. De tal manera que los roles de login pertenezcan a los roles de grupo creados, teniendo en cuenta la asignación de usuario a rol realizada en la aplicación. Los roles de grupo van a contener todos los permisos necesarios para que los usuarios que tenga asignados puedan realizar las operaciones en el sistema.

2.2 Modelo conceptual

A continuación se muestran los conceptos y el modelo conceptual pertenecientes al componente. Varios de estos conceptos están definidos en el modelo conceptual del proyecto Acaxia, el cual puede ser consultado en el Expediente de Proyecto Acaxia 2.0 en la siguiente dirección: http://10.58.19.250/svn/cigcentral/CSGTEC/Acaxia_v2.2.1.1_Beta/ExpedienteProyecto/1.%20ingenieria/1.1%20requisitos/Procesos/

Acciones: actividad o acción concreta que realiza el usuario en el sistema. (Acaxia, 2011)

Aplicación: tipo de programa informático diseñado como herramienta para permitir a un usuario realizar uno o diversos tipos de trabajos.

Conexión: establece los parámetros de conexión que deben ser configurados para acceder a la base de datos.

Funcionalidades: agrupa un conjunto de acciones del sistema. (Acaxia, 2011)

Privilegios: tipo de autorización que puede tener un rol o usuario sobre determinado recurso en la base de datos. (Acaxia, 2011)

Rol: agrupa una serie de permisos sobre sistemas, funcionalidades y acciones que se le asignarán a un conjunto de usuarios. (Acaxia, 2011)

Objetos de base de datos: componente de base de datos. (Acaxia, 2011)

Servicios: servicios web que brindan los sistemas que se encuentran suscritos en Acaxia y que se registran en el IoC[23].

Sistema: producto que se suscribe a Acaxia para que se le brinde seguridad. (Acaxia, 2011)

Usuario: persona que interactúa con el sistema y desempeña un rol en el mismo. (Acaxia, 2011)

[23] **IoC:** Inversión de control. Fichero XML donde se describen los servicios que brindan todos los sistemas.

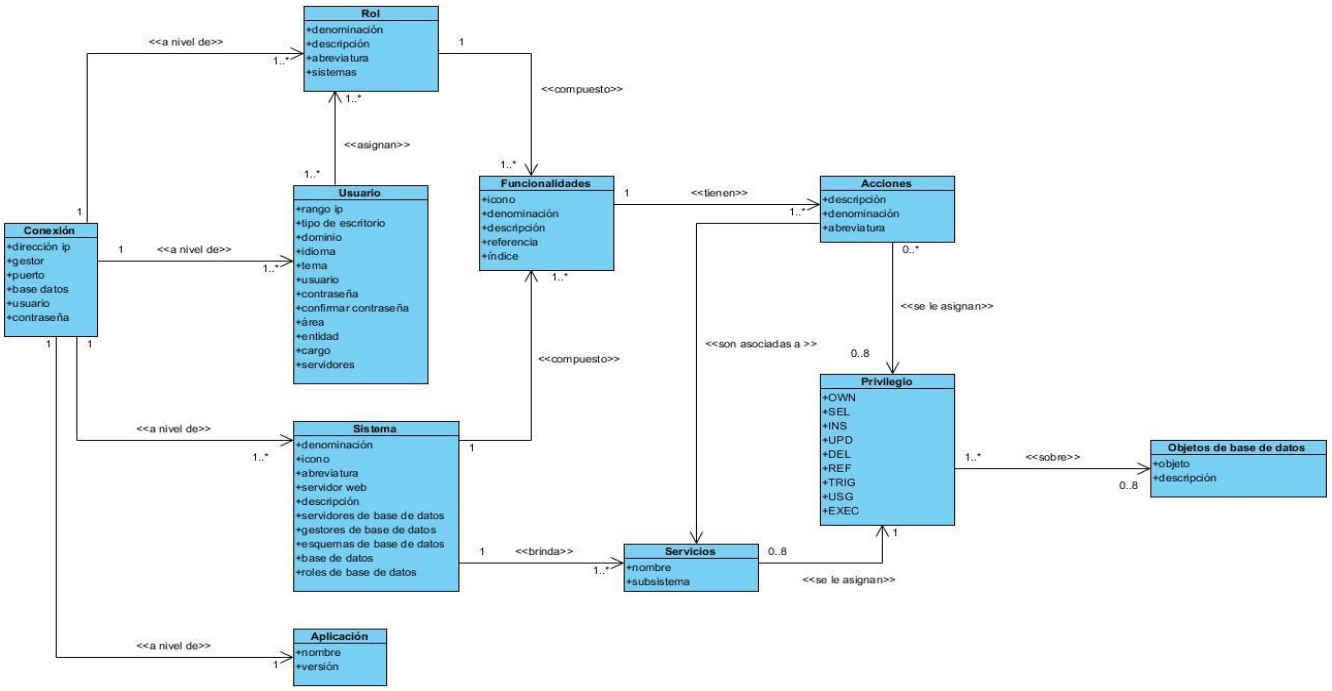


Figura 3. Modelo conceptual.
Fuente: elaboración propia.

2.3 Requisitos de software

Para el proceso de captura de los requisitos del sistema se utilizaron las técnicas:

1 **Tormenta de ideas**, para ello se estructuró un equipo de trabajo conformado por:

- Jefe del departamento de Tecnología.
- Arquitecto de software del departamento Tecnología.
- Arquitecto de datos del departamento Tecnología.
- Jefe del proyecto Acaxia.
- Analista principal del proyecto Acaxia.
- Los estudiantes analista y desarrollador involucrados en el desarrollo del componente.

Después de amplios debates donde todos los implicados mencionados anteriormente expusieron sus puntos de vistas, sin prejuicios y valoraciones que pudieran descartar las ideas del resto de los participantes, se logró una aproximación inicial a los requisitos que se implementarían en el componente.

2 Después de construidos los primeros **prototipos** del componente, fueron presentados al cliente, el cual refinó los requisitos que se reflejaban en el mismo y adicionó nuevos requisitos que no se habían logrado detectar durante las tormentas de ideas.

2.3.1 Requisitos funcionales

Durante la fase de captura de requisitos se identificaron 83 requisitos funcionales para el componente de gestión de políticas de control de acceso a base de datos en el sistema Acaxia (ver Anexo 1). Con el propósito de lograr un mayor entendimiento de los mismos, se realizó una descripción de cada uno de ellos. A continuación se muestran las descripciones de algunos requisitos determinados por el cliente quien desempeña al mismo tiempo el rol de arquitecto de Acaxia. Para consultar el resto de las descripciones dirigirse al Expediente de Proyecto Acaxia 2.0 en la siguiente dirección: http://10.58.19.250/svn/cigcentral/CSGTEC/Acaxia_v2.2.1.1_Beta/ExpedienteProyecto/1.%20ingenieria/1.1%20requisitos/Procesos/Descripci%C3%B3n%20de%20requisitos/

RF1 Gestionar tipo de conexión.

RF1.1 Establecer conexión.

Tabla 10. RF1.1 Establecer conexión.
Fuente: elaboración propia.

Precondiciones	- Se han registrado conexiones a base de datos en el sistema.	
Flujo de eventos		
Flujo básico Establecer conexión.		
1 Se selecciona la conexión a utilizar en el campo **Selección**.		
2 El sistema establece en la base de datos las modificaciones asociadas al tipo de conexión especificada y muestra un mensaje de información: **"La conexión fue modificada satisfactoriamente".**		
3 Concluye el requisito.		
Pos-condiciones		
1 Se establece una conexión a base de datos.		
Flujos alternativos		
1 N/A		
Pos-condiciones		
1 N/A		
Validaciones		
1 Solo puede establecerse una conexión.		
Conceceptos	**Conexión**	Visibles en la interfaz: Denominación Tipo de conexión Descripción Utilizados internamente: N/A
Requisitos especiales	N/A	
Asuntos pendientes	N/A	

Prototipo elemental de interfaz gráfica de usuario.

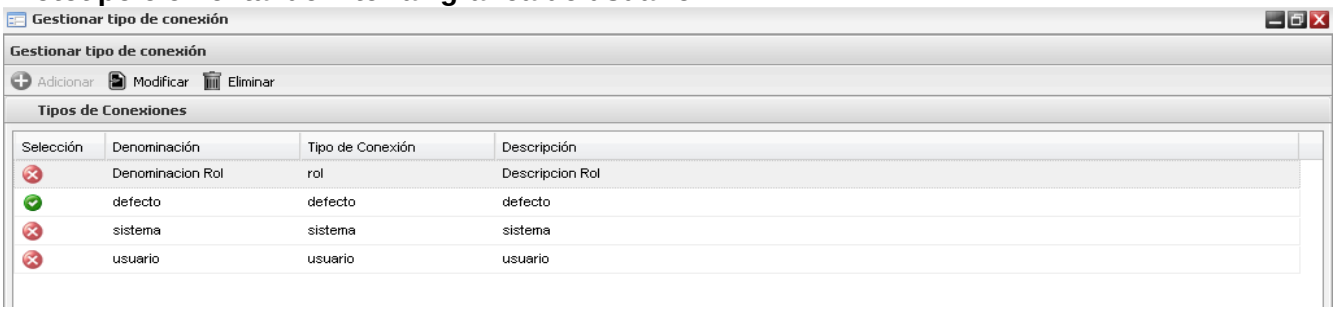


Figura 4. Interfaz del RF2.5 Establecer conexión.
Fuente: elaboración propia.

RF2 Relación entre servicios y objetos de base de datos.

RF2.1 Relacionar servicios con objeto tablas de base de datos.

Tabla 11. RF2.1 Relacionar servicios con objeto tablas de base de datos.
Fuente: elaboración propia.

Precondiciones	– Se deben haber registrado sistemas y establecido una conexión a base de datos para cada uno de ellos. – Se deben haber registrado en el IoC los servicios que brindan los sistemas registrados. – Debe haberse insertado en el nomenclador de objetos de base de datos el objeto tablas. – Deben existir tablas en la conexión a base de datos establecidas al subsistema seleccionado.
Flujo de eventos	
Flujo básico Relacionar servicios con objeto tablas de base de datos.	
1	Se despliega un sistema.
2	Se selecciona un servicio que brinda dicho sistema.
3	Se habilita el combobox Criterio de selección.
4	Se selecciona el objeto de base de datos **Tablas.**
5	Se habilita el botón **Relacionar.**
6	Se muestra un listado de tablas teniendo en cuenta la conexión de base de datos establecida en los subsistemas y los permisos que pueden ser asignados a un servicio sobre dicho objeto.
7	Se marcan o desmarcan los permisos que se deseen sobre una o varias tablas.
8	Se presiona el botón **Relacionar.**
9	El sistema registra los datos y muestra un mensaje de información: **"Permisos gestionados satisfactoriamente"**.
10	Concluye el requisito.
Pos-condiciones	
1	Se asignan un conjunto de privilegios a un servicio sobre objetos tablas de base de datos.
Flujos alternativos	
Flujo alternativo 7.a No se realiza ningún cambio.	
1	Se presiona el botón **Relacionar.**
2	El sistema muestra un mensaje de información: **"Por favor verifique, no ha**

	realizado ningún cambio".	
3	Volver al paso 7 del Flujo básico.	
Pos-condiciones		
1	N/A	
Validaciones		
1	N/A	
Conceptos	**Sistema**	Visibles en la interfaz: Denominación Utilizados internamente: N/A
	Servicio	Visibles en la interfaz: Denominación Utilizados internamente: N/A
	Criterio de selección	Visibles en la interfaz: Objetos de base de datos Utilizados internamente: N/A
	Privilegio	Visibles en la interfaz: OWN SEL INS UPD DEL Utilizados internamente: N/A
Requisitos especiales	N/A	
Asuntos pendientes	N/A	

Prototipo elemental de interfaz gráfica de usuario.

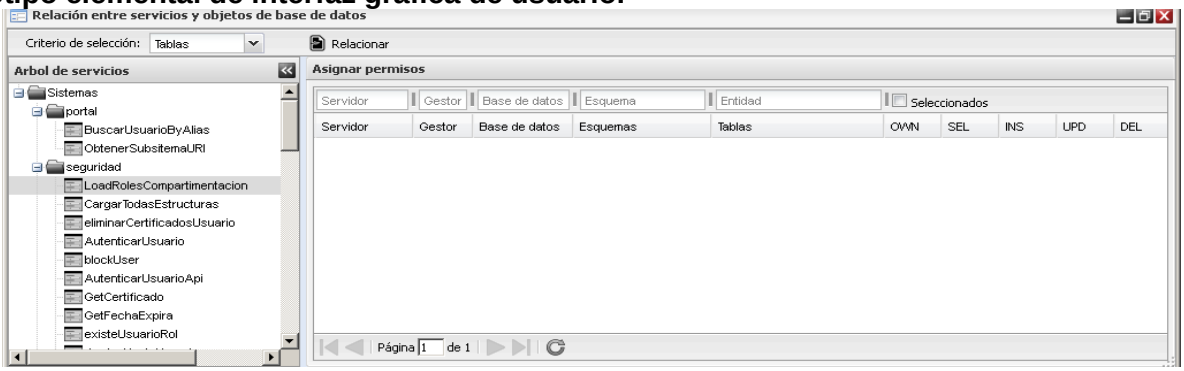


Figura 5. Interfaz del RF3.1 Relacionar servicios con objeto tablas de base de datos. Fuente: elaboración propia.

2.3.2 Requisitos no funcionales

Durante el proceso de captura de requisitos se identificaron que algunos de los requisitos no funcionales (RnF) del sistema Acaxia se ajustan al componente en desarrollo, los cuales se listan a continuación:

Eficiencia

RnF 1: los tiempos de respuesta y velocidad de procesamiento de la información serán rápidos, no mayores de cinco segundos para las actualizaciones y 20 para las recuperaciones.

Soporte.

RnF 2: la aplicación contará antes de su puesta en marcha con un período de pruebas, se le dará mantenimiento, configuración y se brindará el servicio de instalación.

RnF 3: utilizar el estándar de codificación elaborado en el departamento de Tecnología.

Restricciones de diseño

RnF 4: el lenguaje de programación a utilizar para desarrollar el sistema debe ser PHP para la lógica del negocio y ExtJS para la capa vista.

RnF 5: las herramientas a utilizar para desarrollar el sistema deben ser PostgreSQL 9.0 como gestor de base de datos y Pgadmin III como cliente, Doctrine para la capa de acceso a datos y Zend Framework para la lógica del negocio.

RnF 6: el sistema debe ser multiplataforma, haciendo énfasis en Linux y Windows.

Confiabilidad

RnF 7: el sistema debe garantizar protección contra acciones no autorizadas o que puedan afectar la integridad de los datos.

RnF 8: el sistema debe realizar verificaciones sobre las acciones irreversibles (eliminaciones).

2.4 Arquitectura de software

Estilo arquitectónico

El estilo arquitectónico definido para el componente de gestión de políticas de control de acceso a nivel de base de datos en el sistema Acaxia es el estilo en cuatro capas, las cuales se describen a continuación y se muestran en la Figura 6.

- **Capa Vista:** esta es la capa encargada de elaborar y mostrar las interfaces de los usuarios. Está compuesta principalmente por el marco de trabajo ExtJS y centra su desarrollo en dos componentes fundamentales: archivos con extensión JS y PHTML.

- **Capa Controladora:** en esta capa se encuentra ubicado el marco de trabajo Zend Framework y es en la que se implementan las clases controladoras.
- **Capa Modelo:** esta capa es en la que se implementan las clases del modelo, también es la encargada de comunicarse con el gestor de base de datos mediante el framework Doctrine. Está compuesta por el persistidor de configuración FastResponse encargado de comunicarse vía XML con los ficheros de configuración del sistema.
- **Capa de Datos:** en esta capa se encuentran los archivos de configuración XML y el gestor de base de datos.

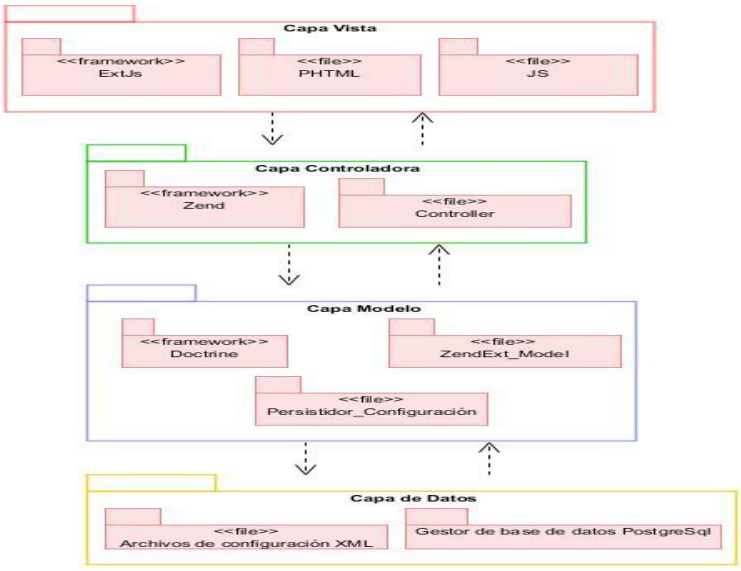


Figura 6. Arquitectura en cuatro capas.
Fuente: elaboración propia.

Patrón arquitectónico

El patrón arquitectónico Modelo-Vista-Controlador se emplea en las tres primeras capas de la arquitectura descrita anteriormente. Este patrón es el encargado de separar los datos de la aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos:

- **Modelo:** está compuesto por datos, reglas de negocio y las funcionalidades correspondientes para la comunicación con el persistidor de datos Doctrine.
- **Controlador:** gestiona las entradas y las respuestas del sistema al usuario.
- **Vista:** muestra la información del modelo al usuario.

2.5 Patrones de diseño

A continuación se enuncian los diferentes patrones de diseño utilizados durante el proceso de desarrollo de la solución.

Patrones GRASP:

- **Experto:** se evidencia el uso de este patrón en todas las clases a utilizar en el componente pues cada clase conoce su información y es la encargada de implementar las funcionalidades que les corresponde como por ejemplo la clase GestnomrelacionController.
- **Creador:** su uso se evidencia en las clases controladoras pues cada una de ellas se encarga de la creación de objetos de varias clases como son DatServicioObjetobdModel, DatObjetobd, entre otras.
- **Controlador:** las diferentes clases controladoras se encargan de llevar el control de todos los eventos relacionados con el negocio. Implementan las funcionalidades que dan respuesta a las peticiones del usuario como por ejemplo la clase controladora GestnomrelacionController.
- **Alta Cohesión:** el uso de éste patrón indica que la información almacenada en las clases debe ser coherente y relacionada a lo que se maneja en dicha clase, se evidencia su uso en todas las clases como por ejemplo en GestnomrelacionController.
- **Bajo Acoplamiento:** el uso de éste patrón se evidencia en la poca relación existente entre las clases que conforman el componente, por ejemplo en la clase Nomobjetospermisos.

Patrones GoF:

El patrón de comportamiento **Mediador** es el encargado de definir un objeto que encapsula cómo interactúan un conjunto de objetos. Estimula la pérdida de acoplamiento ocultando las referencias explícitas entre los objetos, permitiendo variar su interacción de forma independiente. Éste patrón se puede evidenciar en la clase DatSistemaDatServidores, la cual garantiza la comunicación entre varios objetos de distintas clases como por ejemplo las clases *DatSistema y DatServidores.*

El uso de estos patrones garantizó asignar a cada clase la responsabilidad que le corresponde, obtener el menor número de relaciones y dependencias entre clases y aumentar las posibilidades de reusabilidad de las mismas. Todos estos elementos ayudaron a la confección de un diseño de la solución de gran claridad y rendimiento.

2.6 Modelo de diseño

Diagramas de clases del diseño con estereotipos web

Los diagramas de clases especifican las diferentes clases que serán utilizadas en el sistema y las relaciones que existen entre ellas. Para el componente se elaboraron 11 diagramas de clases del diseño con estereotipos web. A continuación se muestra un diagrama de clases, el resto así como las descripciones de cada una de las clases pueden ser consultados en los modelos de diseño del Expediente de Proyecto Acaxia 2.0 en la siguiente dirección:

http://10.58.19.250/svn/cigcentral/CSGTEC/Acaxia_v2.2.1.1_Beta/ExpedienteProyecto/1.%20ingenieria/1.2%20arquitectura%20y%20dise%C3%B1o/Modelo%20de%20dise%C3%B1o/

RF2 Relacionar servicios con objetos de base de datos.

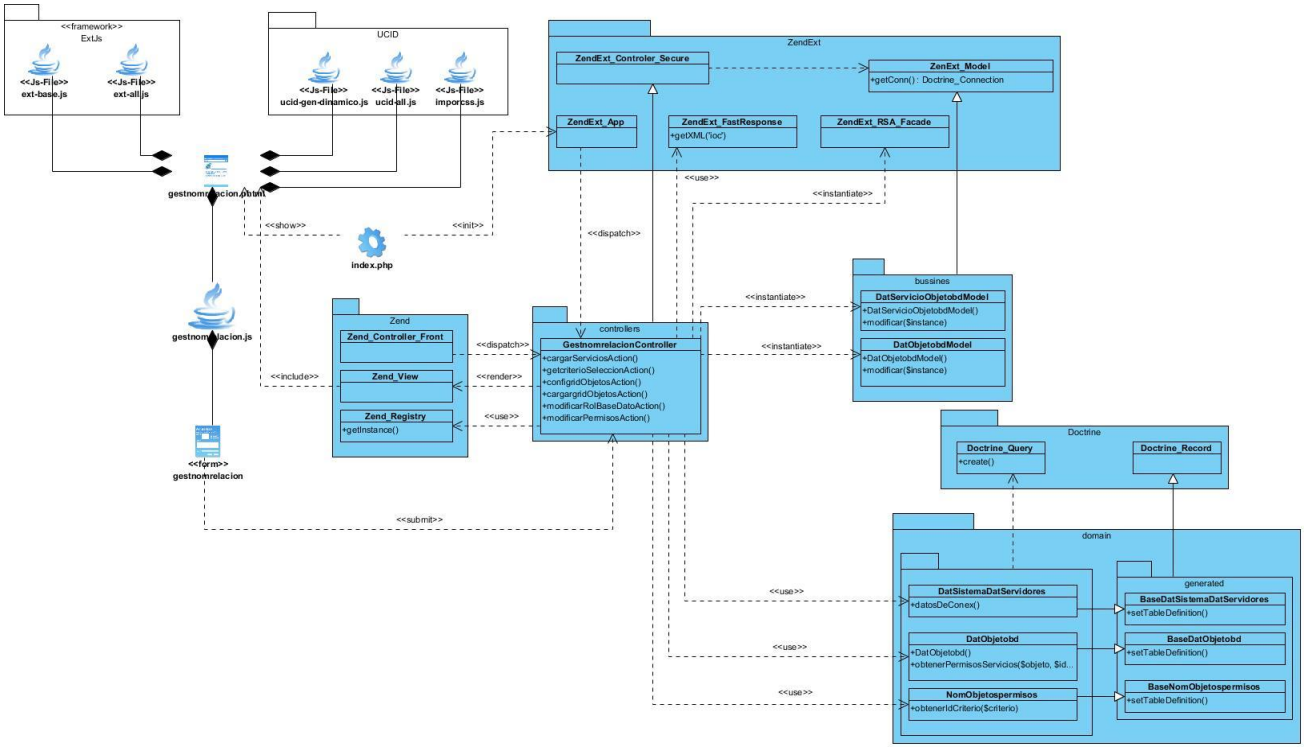


Figura 7. Relacionar servicios con objetos de base de datos.
Fuente: elaboración propia.

Diagrama de secuencia

Los diagramas de secuencia describen la interacción entre los objetos de una aplicación y los mensajes recibidos y enviados por los objetos. (ALTOVA, 2013)

En el presente epígrafe se muestra el diagrama de secuencia perteneciente al requisito Establecer conexión. El resto de los diagramas de secuencia pueden ser consultados en los modelos de diseño del Expediente de Proyecto Acaxia 2.0 en la siguiente dirección: http://10.58.19.250/svn/cigcentral/CSGTEC/Acaxia_v2.2.1.1_Beta/ExpedienteProyecto/1.%20ingenieria/1.2%20arquitectura%20y%20dise%C3%B1o/Modelo%20de%20dise%C3%B1o/

RF1.1 Establecer conexión.

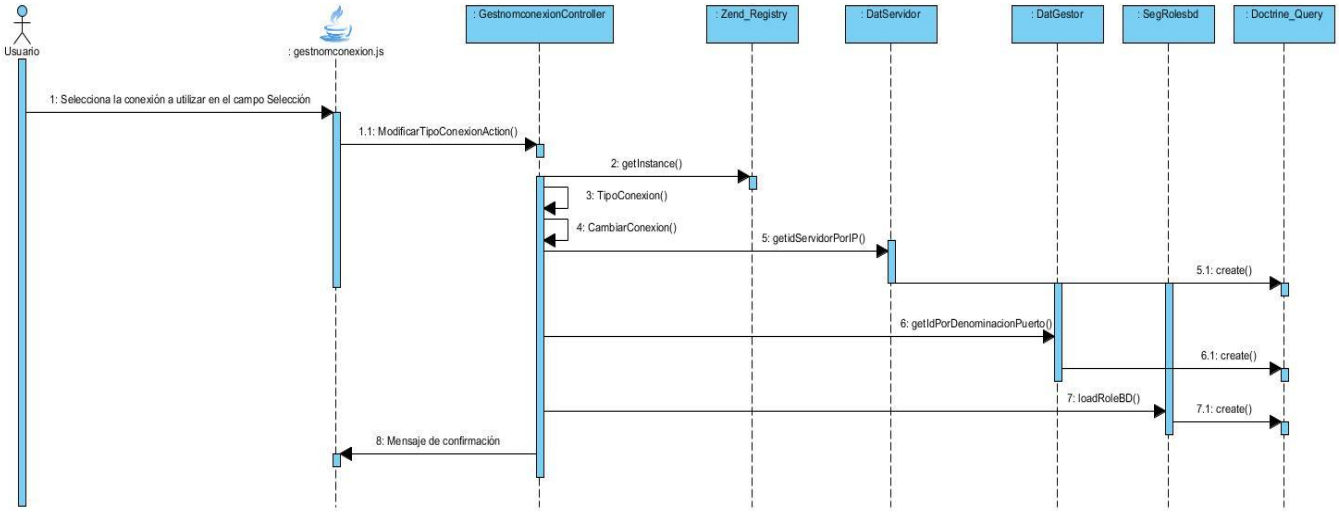


Figura 8.Diagrama de secuencia Establecer conexión.
Fuente: elaboración propia.

2.7 Modelo de datos

Otra de las etapas del diseño es la elaboración del **modelo de datos**, con el propósito de garantizar que los datos persistentes sean almacenados coherente y eficazmente y definir el comportamiento que debe ser implementado en la base de datos. (Ivar Jacobson, 1999) El modelo de datos es una definición lógica, independiente y abstracta de los objetos, operadores y demás que en conjunto constituyen la máquina abstracta con la que interactúan los usuarios. Los objetos nos permiten modelar la estructura de los datos. Los operadores nos permiten modelar su comportamiento. (Date, 2001)

El modelo de datos correspondiente al componente que se desarrolla está compuesto por 24 tablas, verificando que se encuentra en tercera forma normal. A continuación se enuncian algunas tablas correspondientes al componente. Para consultar el diagrama completo dirigirse al Anexo 2.

dat_accion contiene la información relacionada con las acciones del sistema, tales como identificador, denominación, descripción y abreviatura.

dat_servicioioc: contiene los servicios que son relacionados con algún objeto de base de datos.

seg_dat_acción_dat_servicioioc: se forma de la relación entre dat_acción y dat_servicioioc.

dat_objetobd: contiene los objetos de base de datos relacionados.

dat_esquema: contiene la descripción y denominación de los esquemas relacionados con sistemas.

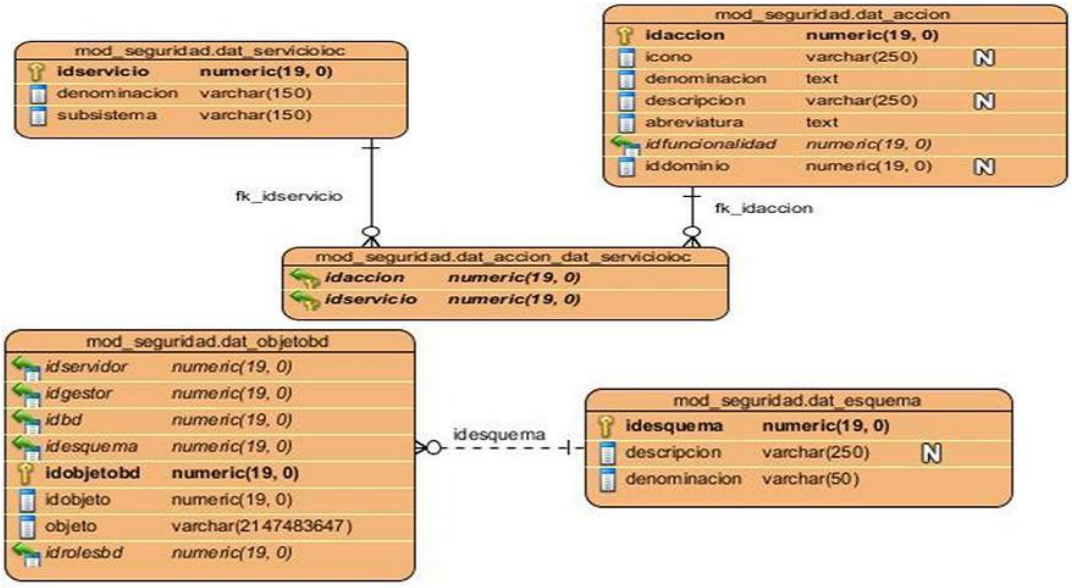


Figura 9. Modelo de datos.
Fuente: elaboración propia.

Conclusiones del capítulo

En el presente capítulo se enunciaron los aspectos fundamentales que se llevan a cabo durante el proceso de análisis y diseño del componente para la gestión de políticas de control de acceso a nivel de base de datos en el sistema Acaxia. Se expusieron los principales artefactos que propone el Modelo de desarrollo del CEIGE. Se describieron los requisitos funcionales y no funcionales con que contará el componente. Se elaboraron los diagramas de clases y de secuencia correspondientes al componente. Se construyó el modelo de datos con el propósito de hacer persistir los datos de manera coherente y eficaz y dar paso a la implementación de la base de datos. Una vez concluido el análisis y diseño de la solución propuesta puede darse paso al flujo de implementación y pruebas de la misma.

Capítulo 3: Implementación y pruebas

Introducción

En el presente capítulo se define el modelo de implementación para poner en práctica el análisis y diseño del componente para la gestión de políticas de control de acceso a nivel de base de datos en el sistema Acaxia realizado en el capítulo anterior. Se describen los estándares de codificación utilizados para garantizar un buen entendimiento del código. Se muestran los resultados de la aplicación de las métricas Tamaño Operacional de Clases y Relaciones entre Clases al diseño elaborado en el capítulo anterior. Se utiliza la técnica del camino básico para obtener la complejidad lógica de los procedimientos. Se muestran los resultados de las pruebas funcionales realizadas al componente para validar que los requisitos identificados fueron implementados correctamente y satisfacen las expectativas del cliente. Además se evidencia el cumplimiento de la idea a defender planteada; a través de un cuestionario respondido por los clientes, dando solución al problema a resolver.

3.1 Modelo de implementación

El modelo de implementación se inicia a partir de los resultados obtenidos en el diseño y describe cómo los elementos del modelo de diseño se implementan en términos de componentes. Describe también cómo se organizan los componentes de acuerdo con los mecanismos de estructuración y modularización disponibles en el entorno de implementación y en el lenguaje de programación utilizado. También describe cómo dependen los componentes unos de otros además de los recursos necesarios para poder ejecutar la herramienta desarrollada. (Acuña, 2013)

Diagrama de componentes

Uno de los artefactos generados en esta fase es el **diagrama de componentes**. El cual representa la estructura física del sistema, su agrupación por paquetes y las dependencias entre estos. A continuación se muestra el diagrama de componentes elaborado. **(Figura 10)**

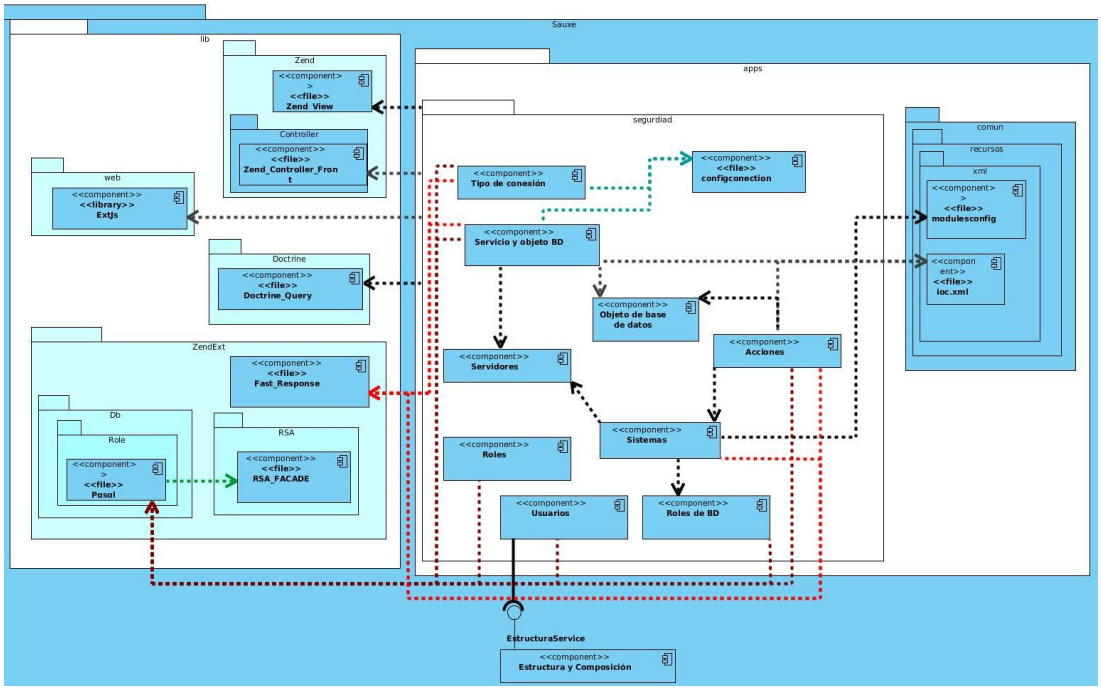


Figura 10. Diagrama de componentes.
Fuente: elaboración propia.

Diagrama de despliegue

Un diagrama de despliegue modela la arquitectura en tiempo de ejecución de un sistema. Muestra la configuración de los elementos de hardware (nodos) y cómo los elementos y artefactos del software se trazan entre estos. Permite modelar las relaciones físicas de los distintos elementos que componen un sistema y el reparto de los componentes sobre dichos nodos. (Visconti, 2012)

En el componente de gestión de políticas de control de acceso, el usuario accede, desde su puesto de trabajo al sistema que se encuentra instalado en el servidor de aplicaciones. Luego dicho servidor se conecta a los servidores de base de datos mediante el protocolo ODBC[24] para realizar las consultas y obtener los resultados deseados. (Figura 11)

Figura 11. Diagrama de despliegue.
Fuente: elaboración propia.

[24] **ODBC:** pos sus siglas en ingles **O**pen **D**ata **B**ase **C**onnectivity. Es un estándar de acceso a las bases de datos. Hace posible acceder a cualquier dato desde cualquier aplicación, sin importar qué sistema de gestión de bases de datos almacene los datos.

3.2 Estándares de codificación

Los estándares de codificación son pautas de programación que no están enfocadas a la lógica del programa, sino a su estructura y apariencia física para facilitar la lectura, comprensión y mantenimiento del código. El uso de estándares de codificación permite lograr un código más legible y reutilizable, de tal forma que se pueda aumentar su mantenibilidad a lo largo del tiempo. (Cuba, 2008)

Para el desarrollo del componente se utilizarán algunos de los estándares de codificación y normas propuestos como parte de la línea de arquitectura determinada para el desarrollo del ERP[25] Cuba. (Cuba, 2008)

Los estándares utilizados en la codificación fueron los siguientes:

- **Notación Húngara:** definir prefijos para cada tipo de datos y según el ámbito de las variables. La idea de esta notación es la de dar mayor información al nombre de la variable, método o función definiendo en ella un prefijo que indique su tipo de dato o ámbito. (Sperberg, 2013)
- **Notación PascalCasing:** es como la notación húngara pero sin prefijos. En este caso los identificadores y nombres de las variables, métodos y funciones están compuestos por múltiples palabras juntas, iniciando cada palabra con letra mayúscula. (Svensk, 2012)
- **Notación CamelCasing:** es parecido al PascalCasing con la excepción que la letra inicial del identificador no debe estar en mayúscula. (Svensk, 2012)

Nomenclatura de las clases

Los nombres de las clases comienzan con la primera letra en mayúscula y el resto en minúscula, en caso de que sea un nombre compuesto se empleará notación PascalCasing. Con sólo leerlo se reconoce el propósito de la misma.

Ejemplo: DatServidor.

Nomenclatura según el tipo de clases

- **Clase controladora:** después del nombre llevan la palabra “Controller”.

  Ejemplo: GestnomgestorController.

Clases del modelo:

- **business (Negocio):** las clases que se encuentran dentro de business después del nombre llevan la palabra “Model”.

Ejemplo: DatGestorModel

[25] **ERP:** por las siglas en inglés **E**nterprise **R**esource **P**lanning.

- **domain (Dominio):** el nombre que reciben las clases que se encuentran dentro de domain es el de la tabla en la base de datos.

Ejemplo: DatGestor

- **generated:** el nombre que reciben las clases que se encuentran dentro de generated comienza con el prefijo “Base” y después el nombre de la tabla en la base de datos.

Ejemplo: BaseDatGestor.

Nomenclatura de las funciones

El nombre a emplear para las funciones se escribe con la primera palabra en minúscula, en caso de que sea un nombre compuesto se empleará notación CamelCasing, y en caso de ser una acción de la clase controladora se debe especificar el nombre de dicha acción en minúscula y seguido el sufijo “Action”.

Ejemplo: modificargestorAction.

Nomenclatura de las variables

El nombre a emplear para las variables se escribe con la primera palabra en minúscula, en caso de que sea un nombre compuesto se empleará notación CamelCasing, y comenzando con un prefijo según el tipo de datos.

Ejemplo: $strgestor.

Prefijos para los tipos de datos:

Tabla 12. Prefijos a utilizar en la creación de variables.
Fuente: elaboración propia.

Tipos de datos	Prefijos
Arreglos	arr
Objetos	obj
Enteros	int
Cadena	str
Float	flt
Boolean	boo

Nomenclatura de los comentarios

Se debe comentar todo lo que se haga dentro del desarrollo, establecer las pautas que conlleven a lograr un código más legible y reutilizable y así se pueda aumentar su mantenimiento a lo largo del tiempo.Los comentarios deben ser lo bastante claros y precisos de forma tal que se entienda el propósito de lo que se está desarrollando.

1. En las clases.

Antes de la declaración de una clase se escribe una breve descripción donde se explique el propósito de la misma. Se escribe de la siguiente forma:

/**

* Nombre de la clase *

* Descripcion *

* @author *

* @package *(módulo)

* @subpackage *(sub módulo)

* @copyright *

* @version (versión - parche) */

2. En las funciones.

Antes de la declaración de la función se escribe una breve descripción donde se explica el propósito de la misma:

/**

* Nombre de la función *

* Descripcion *

* @author * (en caso de que no sea el autor de la clase)

* @param *(los parámetros que se le pasan a la función con su descripción)

* @throws *(en caso de que dispare una excepción)

* @return *(se pone lo que devuelve la función y un comentario)*/

Estilo del código

Cuando se escriba una sentencia en PHP la forma de utilizar los tab del mismo es la siguiente:

- **Sangría o indexado:** la política de sangría a utilizar en la implementación es por **tab**.

```
<?php
/**
 * Indentation
 */
class Example {
    var $theInt = 1;
    function foo($a, $b) {
    switch ( $a) {
    case 0 :
    $Other->doFoo ();
    break;
    default :
    $Other->doBaz ();
    }
    }
    function bar($v) {
    for($i = 0; $i < 10; $i ++) {
    $v->add ( $i );
    }
    }
}
?>
```

Figura 12.Estilo del código: sangría o indexado.
Fuente: Línea de Arquitectura del ERP - Cuba.

- **Brazas o llaves: e**n la declaración de clases o interfaces, métodos, bloques y switch, la apertura de llaves se hace en la misma línea.

```php
<?php
/**
 * Braces
 */
interface EmptyInterface {
}

class Example {
    function bar($p) {
        for($i = 0; $i < 10; $i ++) {
        }
        switch ( $p) {
            case 0 :
                $fField->set ( 0 );
            break;
            case 1 :
                {
                    break;
                }
            default :
                $fField->reset ();
        }
    }
}
?>
```

Figura 13. Estilo del código: brazas o llaves.
Fuente: Línea de Arquitectura del ERP - Cuba.

- **Espacios en blanco:** la declaración de los espacios en blanco en los arreglos es como se muestra en el ejemplo.

```php
<?php
class MyClass implements IO, I1, I2 {
}
class MyClass {
    public $a = 0, $b = 1, $c = 2, $d = 3;
    const MY_TRUE = 1, MY_FALSE = 2;
}
function foo() {
}
function bar(int $x, $y, $z = 1) {
}
?>
```

Figura 14. Estilo del código: Espacios en blanco.
Fuente: Línea de Arquitectura del ERP - Cuba.

3.3 Resultados obtenidos de la aplicación de las métricas para validar el diseño

Las métricas utilizadas para validar el diseño fueron Tamaño Operacional de Clases (TOC) y Relaciones entre Clases (RC).

Tamaño Operacional de Clases (TOC)

Ver los instrumentos y la tabla de resultados para la métrica TOC en el Anexo 3 de la versión digital del documento.

La Figura 15 muestra la representación de los resultados obtenidos agrupados en los intervalos definidos. El gráfico refleja que la mayoría de las clases tienen de 1 a 5 procedimientos. Esto demuestra que el funcionamiento general del componente está distribuido equitativamente entre las diferentes clases.

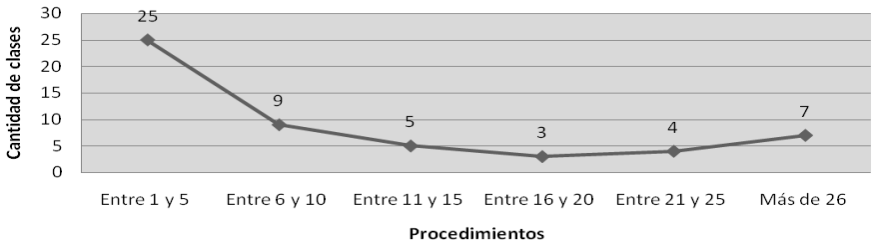


Figura 15. Representación de la evaluación de la métrica TOC.
Fuente: elaboración propia.

La Figura 16 muestra los resultados obtenidos en el instrumento en porciento agrupados en los intervalos definidos.

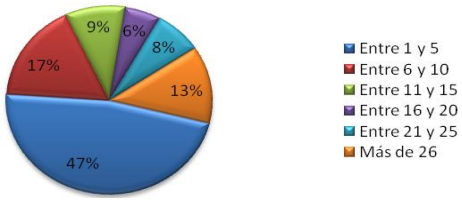


Figura 16. Representación en porciento de los resultados obtenidos en la evaluación de la métrica TOC.
Fuente: elaboración propia.

En la Figura 17 se observa la representación de la incidencia de los resultados de la evaluación de la métrica TOC en el atributo responsabilidad. Este gráfico muestra un resultado satisfactorio pues el 70% de las clases poseen una baja responsabilidad. Esta característica permite que en caso de fallos, como la responsabilidad está distribuida de forma equilibrada ninguna clase es demasiado crítica como para dejar al sistema fuera de servicio.

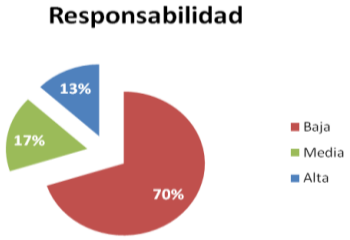


Figura 17. Resultados de la evaluación de la métrica TOC para el atributo responsabilidad.
Fuente: elaboración propia.

En la Figura 18 se observa la representación de la incidencia de los resultados de la evaluación de la métrica TOC en el atributo complejidad de implementación. Este gráfico muestra un resultado satisfactorio pues más del 50% de las clases poseen una baja complejidad de implementación. Esta característica permite mejorar el mantenimiento y soporte de estas clases.

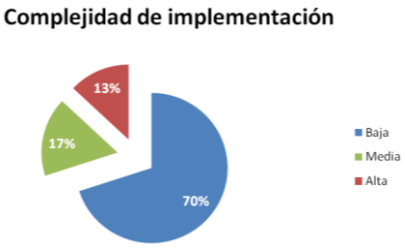


Figura 18. Resultados de la evaluación de la métrica TOC para el atributo complejidad de implementación.

Fuente: elaboración propia.

La Figura 19 muestra la representación de la incidencia de los resultados de la evaluación de la métrica TOC en el atributo reutilización. El diseño del componente tiene un grado de eficiencia aceptable pues solamente el 13% del total de las clases poseen una baja reutilización.

Figura 19. Resultados de la evaluación de la métrica TOC para el atributo reutilización.
Fuente: elaboración propia.

Analizando los resultados obtenidos de la métrica TOC, se puede concluir que el diseño del componente tiene una calidad aceptable teniendo en cuenta los resultados arrojados por los atributos analizados. Solo el 13% de las clases presentan una alta responsabilidad, una alta complejidad de implementación y una baja reutilización, lo cual demuestra que el resultado es satisfactorio.

Relaciones entre Clases (RC)

Ver los instrumentos y la tabla de resultados para la métrica RC en el Anexo 3.

El gráfico de la Figura 20 refleja que el 27% de las clases tienen cero dependencias, y el 45% una dependencia con otra clase. Este resultado es positivo pues demuestra que el 83% de las clases se encuentran dentro de los niveles aceptables de calidad.

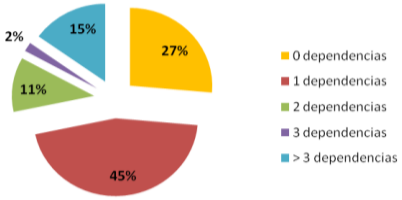


Figura 20. Representación en porciento de los resultados obtenidos en los intervalos definidos según la métrica RC.
Fuente: elaboración propia.

La Figura 21 muestra la representación de la incidencia de los resultados de la evaluación de la métrica RC en el atributo acoplamiento. Se evidencia un bajo acoplamiento entre las clases pues el 27% de las clases no presentan dependencias con otra y el 45% una dependencia con otra. Este resultado es muy favorable para el diseño del componente pues al existir poca dependencia entre las clases aumenta el grado de reutilización del componente.

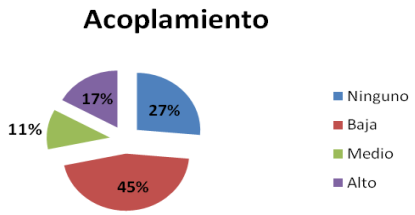


Figura 21. Resultados de la evaluación de la métrica RC para el atributo acoplamiento. Fuente: elaboración propia.

La Figura 22 muestra la representación de la incidencia de los resultados de la evaluación de la métrica RC en el atributo complejidad de mantenimiento. El gráfico refleja un resultado aceptable del atributo pues el 83% de las clases presentan una baja complejidad de mantenimiento.

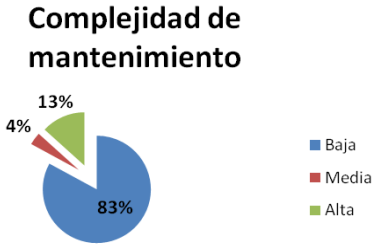


Figura 22. Resultados de la evaluación de la métrica RC para el atributo complejidad de mantenimiento. Fuente: elaboración propia.

La Figura 23 muestra la representación de la incidencia de los resultados de la evaluación del atributo reutilización. Esto evidencia que el 83% de las clases poseen una alta reutilización lo que es un factor fundamental que debe ser tenido en cuenta en el desarrollo de software.

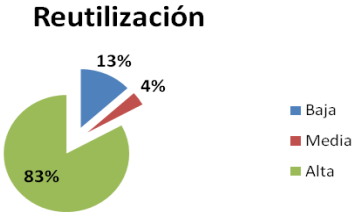


Figura 23. Resultados de la evaluación de la métrica RC para el atributo reutilización. Fuente: elaboración propia.

La Figura 24 muestra la representación de la incidencia de los resultados de la evaluación de la métrica RC en el atributo cantidad de pruebas.

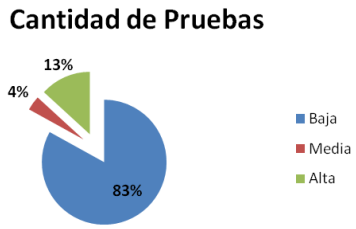


Figura 24. Resultados de la evaluación de la métrica RC para el atributo cantidad de pruebas. Fuente: elaboración propia.

Haciendo un análisis de los resultados obtenidos en la evaluación del instrumento de medición de la métrica RC, se puede concluir que el diseño del componente de gestión de políticas de control de acceso a nivel de base de datos en el sistema Acaxia tiene una calidad aceptable. El 83% de las clases que conforman el componente poseen menos de tres dependencias con otras clases. Los atributos de calidad se encuentran en un nivel satisfactorio; en el 72% de las clases el nivel de acoplamiento es mínimo. La complejidad de mantenimiento y la cantidad de pruebas son bajas a un 83%, lo que representa valores favorables para el diseño realizado. Así mismo, existe un alto grado de reutilización al 83%, comportamiento también favorable para este atributo de calidad.

3.4 Pruebas de software

Pruebas de caja blanca: la técnica utilizada para la aplicación de esta prueba fue la del camino básico. A continuación se enumeran las sentencias de código del método **cargarGridTipoConexionesAction()** a modo de ejemplo.

```
function cargarGridTipoConexionesAction(){
    $registry=Zend_Registry::getInstance();//1
    $dirconfigConection=$registry->config->xml->configConection;//1
    $configConection=new SimpleXMLElement($dirconfigConection,null,true);//1
    $conexiones=$configConection->children();//1
    $filasGrid=array();//1
    $ConexionesEstaticas=array("defecto","sistema","rol","usuario");
    $i=0;//1
    foreach ($conexiones as $conexion) {//2
        array_push($filasGrid, array(//3
            'idconexion'=>$i++,//3
            'seleccion'=>($conexion['seleccion']->__toString()=="false"?false:true),//3
            'tipoconexion'=>$conexion['denominacion']->__toString(),//3
            'descripcion'=>$conexion['descripcion']->__toString(),//3
            'tipo'=>$ConexionesEstaticas[$conexion['tipo']->__toString()]//3
            ));//3
    }//4
    $result['cantidad']=count($filasGrid);//5
    $result['datos']=$filasGrid;//5
    echo json_encode($result);//5
}
```

Figura 25. Código fuente de la funcionalidad cargarGridTipoConexionesAction(). Fuente: elaboración propia.

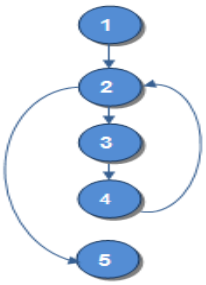


Figura 26. Grafo de flujo asociado a la funcionalidad cargarGridTipoConexionesAction(). Fuente: elaboración propia.

La complejidad ciclomática es la métrica de software que proporciona una medición cuantitativa de la complejidad lógica de un programa. Esta métrica calcula la cantidad de caminos independientes de cada una de las funcionalidades del programa. También provee el límite superior para el número de pruebas que se deben realizar para asegurar que se ejecute cada sentencia al menos una vez. (Pressman, 2005)

Luego de haber construido el grafo se realiza el cálculo de la complejidad ciclomática mediante las tres fórmulas descritas a continuación, las cuales deben arrojar el mismo resultado para asegurar que el cálculo de la complejidad sea el correcto.

1. **V(G) = R** donde R representa la cantidad total de regiones**.**
   V(G) = 2
2. **V(G) = A – N + 2** donde **A** es el número de aristas del grafo de flujo y **N** es el número de nodos del mismo.
   V(G) = 5 – 5 + 2
   V(G) = 2
3. **V (G) = P + 1** donde **P** es el número de nodos predicado contenidos en el grafo de flujo (se denomina nodo predicado a los nodos de los cuales parten dos o más aristas).
   V(G) = 1 + 1
   V(G) = 2

Dado que el cálculo de las tres fórmulas anteriormente mencionadas arrojó el mismo resultado se puede plantear que la complejidad ciclomática del método es dos. Esto significa que existen dos posibles caminos por donde el flujo puede circular. Este valor representa el número mínimo de casos de pruebas para el procedimiento tratado.

- **Camino básico #1:** 1 – 2 – 5.
- **Camino básico #2:** 1 – 2 – 3 – 4 – 2 – 5.

Para cada camino básico determinado se realiza un diseño de caso de prueba.

- **Caso de prueba para el camino básico #1:** Si count($conexiones)==1.
- **Caso de prueba para el camino básico #2:** Si count($conexiones)>1.

Pruebas de caja negra

Para evaluar los requisitos funcionales del componente de gestión de políticas de control de acceso a nivel de base de datos en el sistema Acaxia se realizaron pruebas de caja negra en la fase de pruebas internas. Para aplicarla se confeccionaron los diseños de caso de prueba para cada una de las funcionalidades del componente los cuales pueden ser consultados en el Expediente de Proyecto de Acaxia 2.0 en la siguiente dirección.

http://10.58.19.250/svn/cigcentral/CSGTEC/Acaxia_v2.2.1.1_Beta/ExpedienteProyecto/1.%20ingenieria/1.3%20implementaci%C3%B3n%20y%20prueba/Dise%C3%B1o%20de%20casos%20de%20prueba/

Para comprobar la calidad del componente se realizaron dos iteraciones para las revisiones por el grupo de calidad del centro CEIGE, así como pruebas exploratorias. Las no conformidades encontradas se clasificaron en:

- No conformidades detectadas en la documentación.
- No conformidades detectadas en la aplicación.

Resultados de las pruebas

En la Tabla 13 se recoge la cantidad de no conformidades detectadas en la documentación y la aplicación por cada iteración.

Tabla 13. Tabla de las no conformidades detectadas en la documentación y la aplicación por iteraciones. Fuente: elaboración propia.

Tipo de no conformidades	Documentación	Aplicación
Pruebas exploratorias		
Significativa	5	13
No significativa	3	2
Total	**8**	**15**
Primera iteración		
Significativa	0	7
No significativa	0	0
Total	**0**	**7**
Segunda iteración		
Significativa	0	0
No significativa	0	0
Total	**0**	**0**

A continuación se muestra un gráfico con el porciento que representan las no conformidades por cada una de las iteraciones.

Figura 27. Por ciento que representan las no conformidades de la documentación por cada una de las iteraciones.

Fuente: elaboración propia.

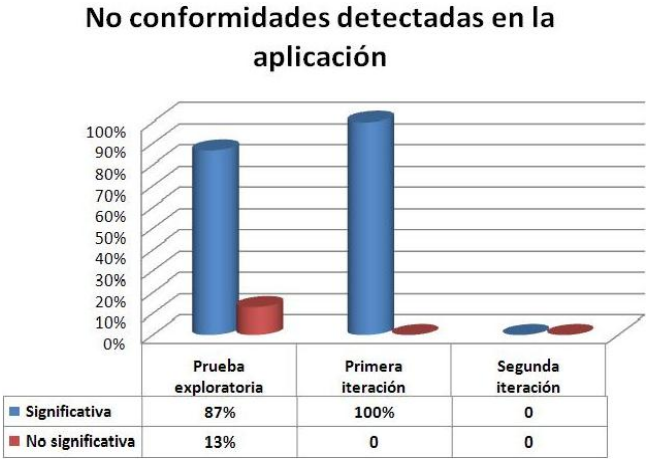


Figura 28. Por ciento que representan las no conformidades de la aplicación por cada una de las iteraciones.

Fuente: elaboración propia.

3.5 Validación del componente mediante cuestionario

En el Sistema de Gestión Integral de Seguridad Acaxia no existía un correcto manejo de las conexiones a base de datos debido a la poca gestión de políticas a este nivel. La falta de gestión de un control de acceso adecuado al gestor trae como consencuencia que se viole el principio de la Seguridad Informática referente al mínimo privilegio que deben poseer los roles en las aplicaciones. Con el objetivo de aportar una solución factible a este problema se decide que si se desarrolla un componente de gestión de políticas de control de acceso se logrará aumentar la granularidad a nivel de base de datos en el sistema Acaxia. Con el objetivo de validar el cumplimiento de la idea a defender para el componente, se confeccionó un cuestionario para que ser respondido por los clientes.

En el cuestionario se realizaron las preguntas de forma afirmativa, con el objetivo de evaluar cada una en un rango de valores de 1 a 5 en cuanto a la veracidad de la afirmación, teniendo para uno las menos ciertas y para cinco las más ciertas. El cuestionario puede ser consultado en el Anexo 4 en la presente versión digital. A continuación se muestran los resultados de la aplicación del mismo.

Figura 29. Resultados de la aplicación del cuestionario.
Fuente: elaboración propia.

En la gráfica de la Figura 29 se puede apreciar que las afirmaciones realizadas en el cuestionario fueron evaluadas de ciertas de un 4,25 a 5 puntos. Del análisis de las respuestas plasmadas por los clientes en el cuestionario, se infiere que el tipo de conexión de usuario es superior a la de rol y esta a su vez a la de sistema. Estas tres conexiones son superiores a la forma anterior de gestionar las conexiones, evidenciando un aumento de la granularidad en la gestión de políticas de control de acceso a nivel de base de datos en el sistema Acaxia.

Conclusiones del capítulo

En el desarrollo del presente capítulo se expusieron los artefactos generados durante la fase de implementación del componente. Se explicaron los estándares de codificación empleados para el desarrollo del mismo. Se validó el diseño propuesto en el capítulo anterior mediante métricas que permitieron demostrar que el componente cuenta con un diseño satisfactorio. Se concluye que las funcionalidades implementadas mostraron un correcto funcionamiento y dan respuesta a los requisitos funcionales. Esto se verificó a partir de las dos iteraciones de pruebas funcionales realizadas por el grupo de calidad del CEIGE. Además el departamento de Tecnología realizó una serie de pruebas a la solución emitiendo un aval que certifica la calidad del componente desarrollado (ver Anexo 5). Por último, se elaboró un cuestionario, el cual fue respondido por los clientes, donde se comprueba que se aumentó la granularidad en el sistema demostrando la idea a defender planteada en el marco conceptual de la investigación.

Conclusiones Generales

Luego del desarrollo del componente para la gestión de políticas de control de acceso a base de datos en el sistema Acaxia, se arriban a las siguientes conclusiones:

- Se realizó un estudio del estado del arte sobre el proceso de la gestión de conexiones y el control de acceso a base de datos en varios frameworks y un sistema de seguridad, donde se obtuvo como resultado que hasta el momento ninguno soluciona la situación problemática expuesta al inicio de la investigación.
- Se realizó el análisis y diseño del componente generando los artefactos establecidos por el modelo de desarrollo del CEIGE para dar paso al proceso de implementación.
- Se desarrolló el componente para la gestión de políticas de control de acceso a nivel de base de datos en el sistema Acaxia empleando herramientas y tecnologías libres obteniendo un producto funcional acorde a los requisitos identificados.
- Se realizaron pruebas para validar el diseño mediante las métricas Tamaño Operacional de Clase y Relaciones entre Clases arrojando resultados satisfactorios.
- Como constancia, la aplicación fue probada por el departamento de Calidad del CEIGE y por el grupo de trabajo del departamento. de Tecnología, los cuales constataron que el componente cumple los objetivos determinados al inicio de esta investigación.

Por lo anteriormente expuesto se concluye que el componente personaliza las transacciones hacia la base de datos aumentando la granularidad en dicho nivel lo cual constituye una valor agregado al Sistema de Gestión Integral de Seguridad Acaxia.

Recomendaciones

Con vista a mejorar la solución se recomienda:

- Establecer algún punto de restauración para salvar la base de datos antes de comenzar las configuraciones.

Referencias bibliográficas

Acaxia, Sistema de Gestión Integral de Seguridad. 2011. *Modelo Conceptual.* La Habana : Universidad de las Ciencias Informáticas, 2011. CIG-SEG-N-i1301.

Acuña, Karenny Brito. 2013. eumed.net. *eumed.net.* [En línea] 2013. [Citado el: 09 de Abril de 2013.] http://www.eumed.net/libros/2009c/584/RUP%20Diseno%20e%20implementacion%20del%20sistema.htm.

ALTOVA. 2013. ALTOVA. *iagramas de secuencia UML.* [En línea] 2013. [Citado el: 21 de Mayo de 2013.] http://www.altova.com/es/umodel/sequence-diagrams.html.

B.Terry, D.Logee. 1990. *Terminology for Software Engineering and Computer-aided Software Engineering. Software Engineering Notes.* Estados Unidos : s.n., 1990.

Baryolo, Ing. Oiner Gómez. 2010. *SOLUCIÓN INFORMÁTICA DE AUTORIZACIÓN EN ENTORNOS MULTIENTIDAD Y MULTISISTEMA. Tesis de Maestría.* La Habana : Universidad de las Ciencias Informáticas, 2010.

Buschmann, F, Kevlinn Henney, Douglas C. Schmidt. 2007. *Pettern-Oriented Software Architecture: A pattern language for distribuited computing.* Tennessee : J ohn Wiley & Sons, 2007. 978-0-470-05902-9.

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., & Stal, M. 1996. *Pattern – Oriented Software Architecture. A System of Patterns.* Inglaterra : John Wiley & Sons, 1996.

Casa, Ignacio. 2010. *Definición y Clasificación de los Sistemas de Información.* Chile : Escuela de Ingeniería Pontificia Universidad Católica, 2010.

CEIGE, Subdirección de Producción. 2013. *MODELO DE DESARROLLO DE SOFTWARE.* La Habana : Universidad de las Ciencias Informáticas, 2013. CEIGE-MD.

Ciberaula. 2010. Ciberaula. *Una Introducción a APACHE.* [En línea] Chile, 2010. [Citado el: 01 de Febrero de 2013.] http://linux.ciberaula.com/articulo/linux_apache_intro.

Cuba, Proyecto ERP. 2008. *Normas y estándares de Codificación del ERP. Línea de Arquitectura.* La Habana : Universidad de las Ciencias Informáticas, 2008.

Date, C. J. 2001. *Introducción a los sistemas de bases de datos.* Mexico : Pearson Educación de México, S.A, 2001. 968-444-419-2.

Definición. 2008. Definición. *Definición.* [En línea] Estados Unidos, 2008. [Citado el: 05 de Febrero de 2013.] http://definicion.de/lenguaje-de-programacion/.

Distancia, Educación a. 2006. *Modelos Conceptuales.* España : s.n., 2006.

Doctrine. 2013. Doctrine. *Introduction.* [En línea] 2013. [Citado el: 3 de Febrero de 2013.] http://docs.doctrine-project.org/projects/doctrine1/en/latest/en/manual/introduction.html#what-is-doctrine.

Erika Camacho, Fabio Cardeso, Gabriel Núñez. 2004. *Arquitecturas de software.* 2004.

ExtJS. 2012. ExtJS. Nuestra comunidad en español. *¿Qué es ExtJS?* [En línea] 2012. [Citado el: 04 de Febrero de 2013.]

Framework, Zend. 2013. Zend Framework. *About.* [En línea] 2013. [Citado el: 02 de Junio de 2013.] http://framework.zend.com/about/.

Ganesh, Gunda Sai. 2008. *Requirements Engineering: Elicitation Techniques.* Suecia : Universidad del Este, Departamento de Tecnología, Matemática y Ciencia de la Computación, 2008. PR003.

Hibernate. 2009. Tutoriales de programación Java. *Hibernate - Parte 1: Persistiendo Objetos Simples usando Mapeos en XML .* [En línea] 01 de Mayo de 2009. [Citado el: 04 de Febrero de 2013.] http://www.javatutoriales.com/2009/05/hibernate-parte-1-persistiendo-objetos.html.

IEEE. 2007. *Introducción a la Ingeniería de Requisitos. Ingeniería de software.* Estados Unidos : s.n., 2007.

Ivar Jacobson, Grady Booch, James Rumbaugh. 1999. *El Proceso Unificado de Desarrollo del Software.* s.l. : Addison-Wesley Professional, 1999. 0201571692.

Kazman, R. 1996. FTP Directory. *Tool Support for Architecture Analysis and Design.* [En línea] 1996. [Citado el: 12 de Mayo de 2013.] ftp://ftp.sei.cmu.edu/pub/sati/Papers_and_Abstracts/ISAW-2.ps.

Lafosse, Jérôme. 2010. *Expert IT Struts 2 - El framework de desarrollo de aplicaciones Java EE.* Barcelona : Ediciones ENI, 2010. ISBN: 978-2-7460-5542-1.

Larman, Craig. 1999. *UML y Patrones.* México : Prentice Hall, 1999. ISBN: 970-17-0261-1.

—. 2003. *UML y patrones: una introducción al análisis y diseño orientado a objetos y al proceso unificado.* s.l. : Alhambra S. A., 2003. 8420534382.

linux, Todo. 2009. Dialnet. *Frameworks PHP: desarrollo web rápido.* [En línea] 2009. [Citado el: 06 de Mayo de 2013.] http://dialnet.unirioja.es/servlet/revista?codigo=13693.

López, José María. 2013. softonic. *Cliente para Subversion para principiantes y expertos.* [En línea] 2013. [Citado el: 03 de Junio de 2013.] http://rapidsvn.softonic.com/.

Martín, Alfredo Payá. 2009. *ANÁLISIS Y USO DE FRAMEWORKS.* s.l. : UNIVERSIDAD PONTIFICIA COMILLAS, 2009.

Miguel Sánchez, Beatriz Jiménez, Francisco L. Gutiérrez. 2006. *Estudio del Control de Acceso en Sistemas Colaborativos.* España : Granada, 2006.

Mularien, Peter. 2010. *Spring Security 3.* Birmingham : Packt Publishing Ltd, 2010. SBN 978-1-847199-74-4.

NetBeans. 2013. NetBeans. *Bienvenido a NetBeans y www.netbeans.org.* [En línea] Estados Unidos, 2013. [Citado el: 04 de Febrero de 2013.] https://netbeans.org/index_es.html.

Paradigm, Visual. 2013. Visual Paradigm. *Visual Paradigm for UML.* [En línea] Kln, Hong Kong, 21 de Enero de 2013. [Citado el: 03 de Junio de 2013.] http://www.visual-paradigm.com/product/vpuml/.

Pfleeger, Charles P. 2006. *Security in computing.* s.l. : Prentice Hall, 2006. ISBN-10: 0-13-239077-9.

PHP. 2013. PHP. *PHP.* [En línea] 31 de Mayo de 2013. [Citado el: 02 de Junio de 2013.] http://www.php.net/manual/es/preface.php.

PosgreSQL-es. 2010. PosgreSQL-es. *Sobre PostgreSQL. Introducción.* [En línea] España, 02 de Octubre de 2010. [Citado el: 01 de Febrero de 2013.] http://www.postgresql.org.es/sobre_postgresql.

Pressman, Roger S. 2005. *Ingeniería de Software. Un enfoque práctico. Quinta edición.* 2005.

Prieto, Félix. 2009. *Patrones de diseño.* España : Departamento de Informática. Universidad de Valladolid, 2009.

Sommerville, Ian. 2005. *Ingeniería de Software. 7ma.* 2005.

Sperberg, Camilo. 2013. unreal4u´s Personal Network. *unreal4u´s Personal Network.* [En línea] 2 de 6 de 2013. [Citado el: 2 de 6 de 2013.] http://blog.unreal4u.com/2011/03/sobre-convenciones-y-notaciones-hungara-camelcase-etc/.

SpringSource. 2013. SpringSource. *Spring Framework.* [En línea] 2013. [Citado el: 19 de Mayo de 2013.] http://static.springsource.org/spring-security/site/index.html.

Svensk, Magnus. 2012. Högskolan I Gävle. *Högskolan I Gävle.* [En línea] 12 de 06 de 2012. [Citado el: 2 de 6 de 2013.] http://urn.kb.se/resolve?urn=urn:nbn:se:hig:diva-12010.

symfony.es. 2012. symfony.es. *Popularidad mundial de los frameworks PHP.* [En línea] 20 de Septiembre de 2012. [Citado el: 06 de Mayo de 2013.] http://www.symfony.es/noticias/2012/09/20/popularidad-mundial-de-los-frameworks-php/.

Tedeschi, Nicolás. 2013. MSDN. *¿Qué es un Patrón de Diseño?* [En línea] 2013. [Citado el: 25 de Febrero de 2013.] http://msdn.microsoft.com/es-es/library/bb972240.aspx.

Thais Figueras, Miguel La Llave. 2012. *Solución informática para la Compartimentación de la información en los sistemas que utilizan el Sistema de Gestión Integral de Seguridad ACAXIA.* La Habana : Universidad de las Ciencias Informáticas, 2012.

Visconti, Marcello. 2012. *Fundamentos de Ingeniería de de Software.* Chile : Universidad Técnica de Federico Santa María, 2012.

Web, Libros. 2013. Libros Web. *Symfony 1.2, la guía definitiva. Libros Web.* [En línea] España, 2013. [Citado el: 24 de Enero de 2013.] http://www.librosweb.es/symfony/capitulo_1/symfony_en_pocas_palabras.html.

ZendFramework. 2010. ZendFramework. *Parte I. Introducción a Zend Framework.* [En línea] 2010. [Citado el: 24 de Enero de 2013.] http://manual.zfdes.com/es/introduction.overview.html.

Anexos

Anexo 1. Listado de requisitos funcionales.

Tabla 14. Listado de requisitos funcionales para el componente.
Fuente: elaboración propia.

RF1	**Gestionar tipos de conexión.** **RF1.1** Adicionar conexión. **RF1.2** Modificar conexión. **RF1.3** Eliminar conexión. **RF1.4** Listar tipos de conexión. **RF1.5** Establecer conexión.
RF2	**Relacionar acciones con privilegios sobre objetos de base de datos.** **RF2.1** Relacionar acciones con privilegios sobre el objeto Tabla de base de datos. **RF2.2** Relacionar acciones con privilegios sobre el objeto Vistas de base de datos. **RF2.3** Relacionar acciones con privilegios sobre el objeto Secuencias de base de datos. **RF2.4** Relacionar acciones con privilegios sobre el objeto Funciones de base de datos. **RF2.5** Buscar objeto Tablas de base de datos. **RF2.6** Buscar objeto Vistas de base de datos. **RF2.7** Buscar objeto Secuencia de base de datos. **RF2.8** Buscar objeto funciones de base de datos. **RF2.9** Listar objeto Tablas de base de datos. **RF2.10** Listar objeto Vistas de base de datos. **RF2.11** Listar objeto Secuencias de base de datos. **RF2.12** Listar objeto Funciones de base de datos. **RF2.13** Relacionar acciones con privilegios sobre servicios. **RF2.14** Buscar servicios que brinda un sistema. **RF1.15** Listar servicios. **RF2.16** Comprobar conexión de servidores al asignar privilegios sobre el objeto funciones de base de datos. **RF2.17** Comprobar conexión de servidores al asignar privilegios sobre el objeto secuencias de base de datos. **RF2.18** Comprobar conexión de servidores al asignar privilegios sobre el objeto tablas de base de datos. **RF2.19** Comprobar conexión de servidores al asignar privilegios sobre el objeto vistas de base de datos.
RF3	**Relacionar servicios con objetos de base de datos.** **RF3.1** Relacionar servicios con objeto tablas de base de datos. **RF3.2** Relacionar servicios con objeto vistas de base de datos. **RF3.3** Relacionar servicios con objeto funciones de base de datos. **RF3.4** Relacionar servicios con objeto secuencias de base de datos. **RF3.5** Listar sistemas y servicios. **RF3.6** Listar objeto tablas de base de datos y permisos. **RF3.7** Listar objeto vistas de base de datos y permisos. **RF3.8** Listar objeto funciones de base de datos y permisos.

	RF3.9 Listar objeto secuencias de base de datos y permisos. **RF3.10** Buscar objeto tablas de base de datos. **RF3.11** Buscar objeto vistas de base de datos. **RF3.12** Buscar objeto funciones de base de datos. **RF3.13** Buscar objeto secuencias de base de datos.
RF4	**Gestionar sistemas.** **RF4.1** Asignar conexiones para bases de datos Postgres. **RF4.2** Listar servidores, gestores, base de datos, esquemas y rol de conexión.
RF5	**Gestionar Servidor.** **RF5.1** Adicionar Servidor de base de datos. **RF5.2** Modificar Servidor de base de datos. **RF5.3** Eliminar Servidor. **RF5.4** Listar servidores.
RF6	**Gestionar usuario.** **RF6.1** Asignar roles.
RF7	**Gestionar roles.** **RF7.1** Adicionar rol. **RF7.2** Modificar rol. **RF7.3** Eliminar rol. **RF7.4** Listar roles. **RF7.5** Buscar rol. **RF7.6** Regular acciones.
RF8	**Gestionar gestor de Base de datos.** **RF8.1** Adicionar gestor de bases de datos a un servidor. **RF8.2** Eliminar gestor de bases de datos asociado a un servidor. **RF8.3** Listar gestores de bases de datos asociados a un servidor. **RF8.4** Listar servidores de base de datos.
RF9	**Gestionar roles para bases de datos Postgres.** **RF9.1** Adicionar rol de base de datos Postgres. **RF9.2** Modificar rol de base de datos Postgres. **RF9.3** Eliminar rol de base de datos Postgres. **RF9.4** Asignar permisos sobre objetos de base de datos Postgres. **RF9.5** Listar roles de base de datos Postgres. **RF9.6** Listar Tablas y permisos. **RF9.7** Listar Vistas y permisos. **RF9.8** Listar Secuencias y permisos. **RF9.9** Listar Esquemas y permisos. **RF9.10** Listar Bases de Datos y permisos. **RF9.11** Listar Funciones y permisos. **RF9.12** Buscar Funciones. **RF9.13** Buscar Tablas. **RF9.14** Buscar Esquemas. **RF9.15** Buscar Secuencia. **RF9.16** Buscar Vista. **RF9.17** Buscar Bases de Datos. **RF9.18** Buscar roles de base de datos Postgres. **RF9.19** Listar servidores, gestores y bases de datos.

	RF9.20 Seleccionar rol de conexión a base de datos.
RF10	**Gestionar Nomenclador de Gestores de Bases de Datos.** **RF10.1** Adicionar nomenclador de gestor de bases de datos. **RF10.2** Modificar nomenclador de gestor de bases de datos **RF10.3** Eliminar nomenclador de gestor de bases de datos. **RF10.4** Listar nomenclador de gestores de bases de datos. **RF10.5** Buscar nomenclador de gestor de bases de datos.
RF11	**Gestionar nomenclador para objetos de base de datos Postgres.** **RF11.1** Adicionar objeto de base de datos. **RF11.2** Modificar objeto de base de datos. **RF11.3** Eliminar objeto de base de datos. **RF11.4** Listar objeto de base de datos.

Anexo 2. Modelo de datos.

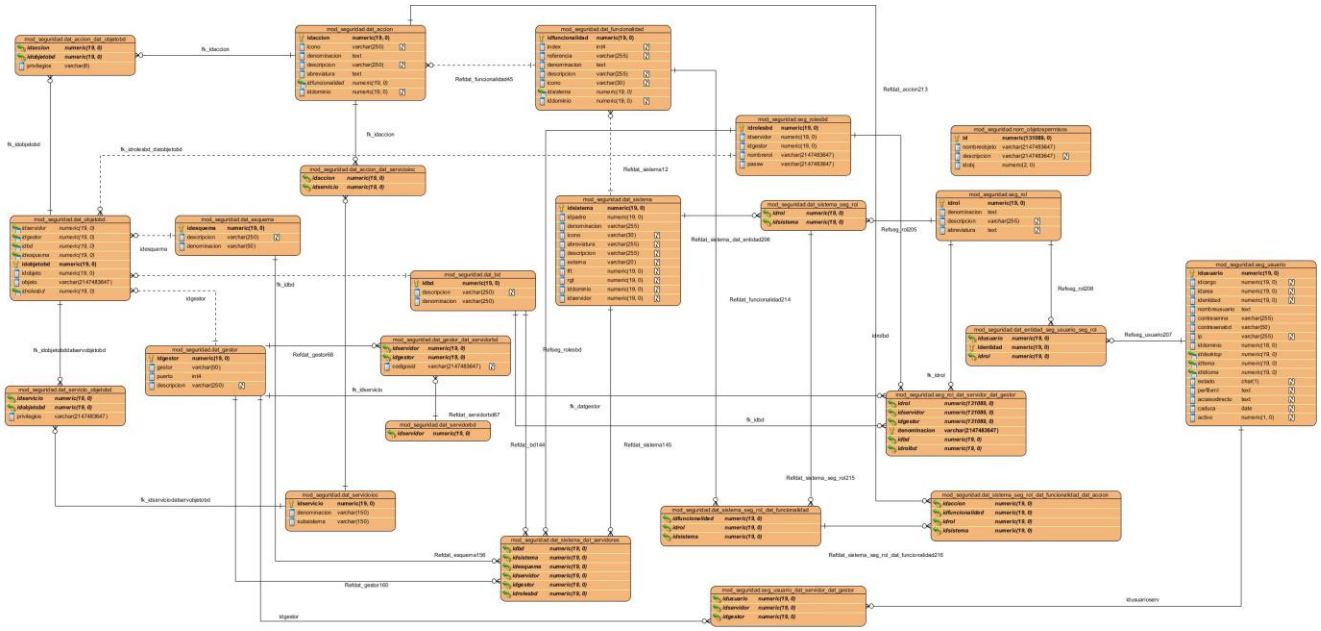


Figura 30. Modelo de datos general del componente desarrollado.
Fuente: elaboración propia.

Anexo 3. Instrumentos de evaluación de las métricas TOC y RC

Tabla 15. Instrumento de evaluación de la métrica TOC.
Fuente: elaboración propia.

Clase	Cantidad de procedimientos	Responsabilidad	Complejidad	Reutilización
GestaccionController	56	Alta	Alta	Baja
DatAccionModel	4	Baja	Baja	Alta
DatObjetobdModel	4	Baja	Baja	Alta
DatAccionDatServicioiocModel	3	Baja	Baja	Alta
DatAccionDatObjetobdModel	4	Baja	Baja	Alta
DatAccionDatObjetobd	1	Baja	Baja	Alta
DatSistemaDatServidores	29	Alta	Alta	Baja

DatAccionDatServicioioc	1	Baja	Baja	Alta
NomObjetospermisos	7	Baja	Baja	Alta
DatObjetobd	5	Baja	Baja	Alta
DatServicioioc	3	Baja	Baja	Alta
DatAccion	24	Media	Media	Media
GestsistemaController	49	Alta	Alta	Baja
DatSistemaModel	3	Baja	Baja	Alta
SegRol	20	Media	Media	Media
DatBd	13	Baja	Baja	Alta
DatEsquema	14	Media	Media	Media
DatServidor	14	Media	Media	Media
GestnomconexionController	17	Media	Media	Media
DatGestor	19	Media	Media	Media
SegRolesbd	7	Baja	Baja	Alta
SegRolDatServidorDatGestor	9	Baja	Baja	Alta
SegUsuarioDatServidorDatGestor	5	Baja	Baja	Alta
GestnomrelacionController	47	Alta	Alta	Baja
DatServicioObjetobdModel	4	Baja	Baja	Alta
GestservidorController	12	Baja	Baja	Alta
DatServidorModel	5	Baja	Baja	Alta
DatServidorbd	2	Baja	Baja	Alta
SegUsuarioDatSerautenticacion	3	Baja	Baja	Alta
GestusuarioController	64	Alta	Alta	Baja
SegUsuarioModel	7	Baja	Baja	Alta
DatFuncionalidad	23	Media	Media	Media
DatEntidadSegUsuarioSegRol	23	Media	Media	Media
DatReglas	2	Baja	Baja	Alta
GestrolController	29	Alta	Alta	Baja
SegRolModel	5	Baja	Baja	Alta
DatSistemaSegRolModel	3	Baja	Baja	Alta
DatSistemaSegRolDatFuncionalidadModel	3	Baja	Baja	Alta
DatSistemaSegRolDatFuncionalidadDatAccionModel	3	Baja	Baja	Alta
SegRolNomDominio	2	Baja	Baja	Alta
SegRolAsignacion	2	Baja	Baja	Alta
DatSistemaSegRol	6	Baja	Baja	Alta
DatSistemaSegRolDatFuncionalidad	8	Baja	Baja	Alta
DatSistemaSegRolDatFuncionalidadDatAccion	11	Baja	Baja	Alta
DatSistema	21	Media	Media	Media
GestgestorController	7	Baja	Baja	Alta
DatGestorModel	4	Baja	Baja	Alta
DatGestorDatServidorbdModel	2	Baja	Baja	Alta
DatGestorDatServidorbd	5	Baja	Baja	Alta

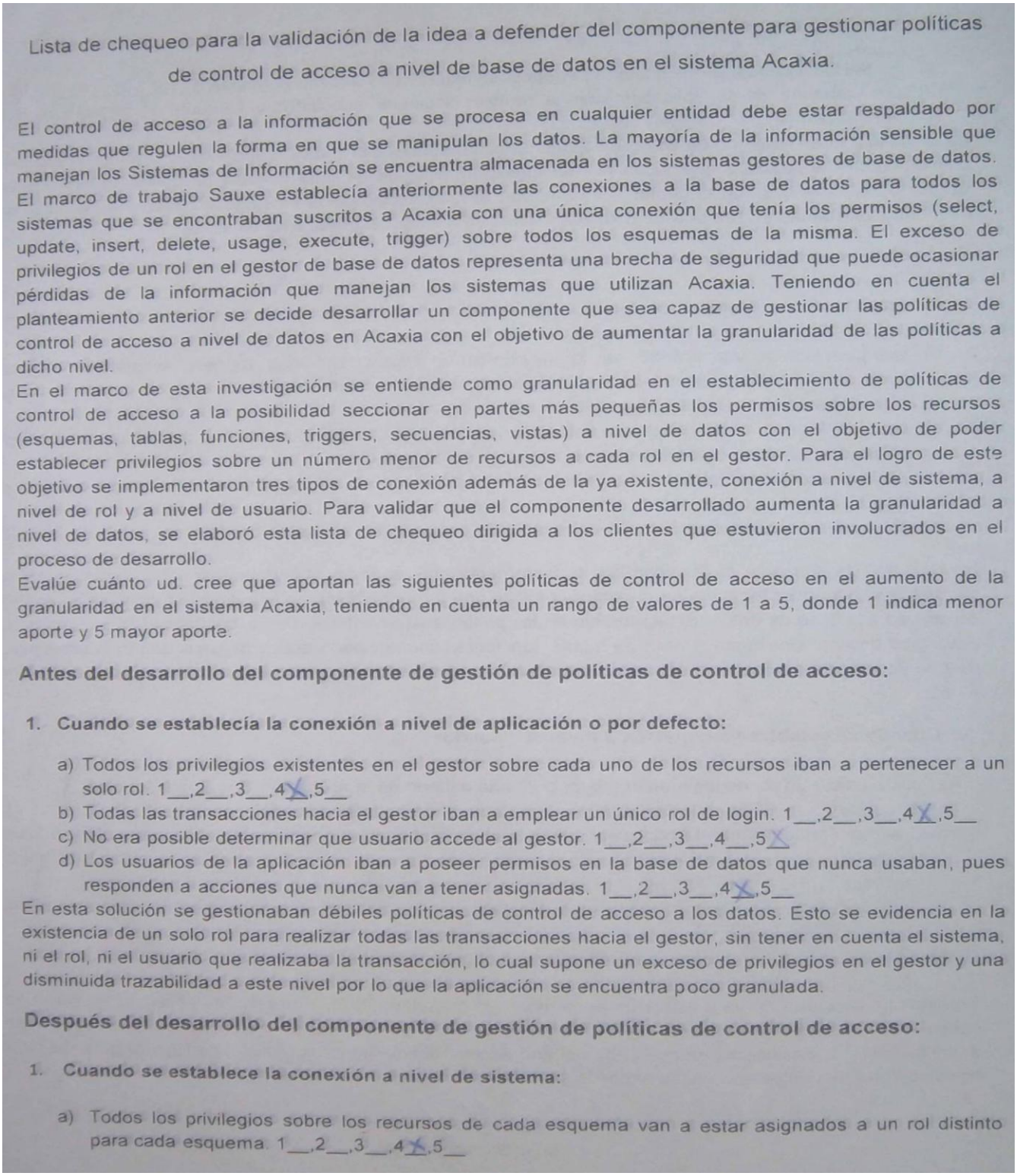
GestrolesbdController	56	Alta	Alta	Baja
GestnomgestorController	7	Baja	Baja	Alta
GestnomobjetoController	9	Baja	Baja	Alta
NomObjetospermisosModel	4	Baja	Baja	Alta

Tabla 16. Instrumento de evaluación de la métrica RC.
Fuente: elaboración propia.

Clase	Cantidad de relaciones	Acoplamiento	Complejidad	Reutilización	Cantidad de pruebas
GestaccionController	16	Alta	Alta	Baja	Alta
DatAccionModel	0	Ninguno	Baja	Alta	Baja
DatObjetobdModel	0	Ninguno	Baja	Alta	Baja
DatAccionDatServicioiocModel	0	Ninguno	Baja	Alta	Baja
DatAccionDatObjetobdModel	2	Media	Baja	Alta	Baja
DatAccionDatObjetobd	0	Ninguno	Baja	Alta	Baja
DatSistemaDatServidores	1	Bajo	Baja	Alta	Baja
DatAccionDatServicioioc	0	Ninguno	Baja	Alta	Baja
NomObjetospermisos	1	Bajo	Baja	Alta	Baja
DatObjetobd	1	Bajo	Baja	Alta	Baja
DatServicioioc	1	Bajo	Baja	Alta	Baja
DatAccion	1	Bajo	Baja	Alta	Baja
GestsistemaController	15	Alta	Alta	Baja	Alta
DatSistemaModel	0	Ninguno	Baja	Alta	Baja
SegRol	2	Media	Baja	Alta	Baja
DatBd	1	Bajo	Baja	Alta	Baja
DatEsquema	1	Bajo	Baja	Alta	Baja
DatServidor	1	Bajo	Baja	Alta	Baja
GestnomconexionController	6	Alta	Alta	Baja	Alta
DatGestor	1	Bajo	Baja	Alta	Baja
SegRolesbd	1	Bajo	Baja	Alta	Baja
SegRolDatServidorDatGestor	1	Bajo	Baja	Alta	Baja
SegUsuarioDatServidorDatGestor	1	Bajo	Baja	Alta	Baja
GestnomrelacionController	7	Alta	Alta	Baja	Alta
DatServicioObjetobdModel	0	Ninguno	Baja	Alta	Baja
GestservidorController	6	Alta	Alta	Baja	Alta
DatServidorModel	0	Ninguno	Baja	Alta	Baja
DatServidorbd	1	Bajo	Baja	Alta	Baja
SegUsuarioDatSerautenticacion	1	Bajo	Baja	Alta	Baja
GestusuarioController	4	Alta	Media	Media	Media
SegUsuarioModel	1	Bajo	Baja	Alta	Baja
DatFuncionalidad	1	Bajo	Baja	Alta	Baja

DatEntidadSegUsuarioSegRol	1	Bajo	Baja	Alta	Baja
DatReglas	1	Bajo	Baja	Alta	Baja
GestrolController	16	Alta	Alta	Baja	Alta
SegRolModel	2	Media	Baja	Alta	Baja
DatSistemaSegRolModel	0	Ninguno	Baja	Alta	Baja
DatSistemaSegRolDatFuncionalidadModel	0	Ninguno	Baja	Alta	Baja
DatSistemaSegRolDatFuncionalidadDatAccionModel	0	Ninguno	Baja	Alta	Baja
SegRolNomDominio	1	Bajo	Baja	Alta	Baja
SegRolAsignacion	1	Bajo	Baja	Alta	Baja
DatSistemaSegRol	1	Bajo	Baja	Alta	Baja
DatSistemaSegRolDatFuncionalidad	1	Bajo	Baja	Alta	Baja
DatSistemaSegRolDatFuncionalidadDatAccion	1	Bajo	Baja	Alta	Baja
DatSistema	1	Bajo	Baja	Alta	Baja
GestgestorController	2	Media	Baja	Alta	Baja
DatGestorModel	3	Alta	Media	Media	Media
DatGestorDatServidorbdModel	0	Ninguno	Baja	Alta	Baja
DatGestorDatServidorbd	0	Ninguno	Baja	Alta	Baja
GestrolesbdController	6	Alta	Alta	Baja	Alta
GestnomgestorController	2	Media	Baja	Alta	Baja
GestnomobjetoController	2	Media	Baja	Alta	Baja
NomObjetospermisosModel	0	Ninguno	Baja	Alta	Baja

Anexo 4. Cuestionario aplicado a los clientes.



Lista de chequeo para la validación de la idea a defender del componente para gestionar políticas de control de acceso a nivel de base de datos en el sistema Acaxia.

El control de acceso a la información que se procesa en cualquier entidad debe estar respaldado por medidas que regulen la forma en que se manipulan los datos. La mayoría de la información sensible que manejan los Sistemas de Información se encuentra almacenada en los sistemas gestores de base de datos. El marco de trabajo Sauxe establecía anteriormente las conexiones a la base de datos para todos los sistemas que se encontraban suscritos a Acaxia con una única conexión que tenía los permisos (select, update, insert, delete, usage, execute, trigger) sobre todos los esquemas de la misma. El exceso de privilegios de un rol en el gestor de base de datos representa una brecha de seguridad que puede ocasionar pérdidas de la información que manejan los sistemas que utilizan Acaxia. Teniendo en cuenta el planteamiento anterior se decide desarrollar un componente que sea capaz de gestionar las políticas de control de acceso a nivel de datos en Acaxia con el objetivo de aumentar la granularidad de las políticas a dicho nivel.

En el marco de esta investigación se entiende como granularidad en el establecimiento de políticas de control de acceso a la posibilidad seccionar en partes más pequeñas los permisos sobre los recursos (esquemas, tablas, funciones, triggers, secuencias, vistas) a nivel de datos con el objetivo de poder establecer privilegios sobre un número menor de recursos a cada rol en el gestor. Para el logro de este objetivo se implementaron tres tipos de conexión además de la ya existente, conexión a nivel de sistema, a nivel de rol y a nivel de usuario. Para validar que el componente desarrollado aumenta la granularidad a nivel de datos, se elaboró esta lista de chequeo dirigida a los clientes que estuvieron involucrados en el proceso de desarrollo.

Evalúe cuánto ud. cree que aportan las siguientes políticas de control de acceso en el aumento de la granularidad en el sistema Acaxia, teniendo en cuenta un rango de valores de 1 a 5, donde 1 indica menor aporte y 5 mayor aporte.

Antes del desarrollo del componente de gestión de políticas de control de acceso:

1. **Cuando se establecía la conexión a nivel de aplicación o por defecto:**

   a) Todos los privilegios existentes en el gestor sobre cada uno de los recursos iban a pertenecer a un solo rol. 1__,2__,3__,4X,5__
   b) Todas las transacciones hacia el gestor iban a emplear un único rol de login. 1__,2__,3__,4X,5__
   c) No era posible determinar que usuario accede al gestor. 1__,2__,3__,4__,5X
   d) Los usuarios de la aplicación iban a poseer permisos en la base de datos que nunca usaban, pues responden a acciones que nunca van a tener asignadas. 1__,2__,3__,4X,5__

En esta solución se gestionaban débiles políticas de control de acceso a los datos. Esto se evidencia en la existencia de un solo rol para realizar todas las transacciones hacia el gestor, sin tener en cuenta el sistema, ni el rol, ni el usuario que realizaba la transacción, lo cual supone un exceso de privilegios en el gestor y una disminuida trazabilidad a este nivel por lo que la aplicación se encuentra poco granulada.

Después del desarrollo del componente de gestión de políticas de control de acceso:

1. **Cuando se establece la conexión a nivel de sistema:**

   a) Todos los privilegios sobre los recursos de cada esquema van a estar asignados a un rol distinto para cada esquema. 1__,2__,3__,4X,5__

Figura 31. Cuestionario, primer cliente.
Fuente: elaboración propia.

b) Las transacciones hacia el gestor van a emplear un único rol de login por cada sistema. 1__,2__,3__,4__,5__
c) Es posible conocer a que sistema pertenece el usuario que accede al gestor. 1__,2__,3__,4__,5__
d) Los usuarios de la aplicación van a realizar acciones asociadas solamente al sistema al que pertenecen. 1__,2__,3__,4__,5__

La granularidad en este tipo de conexión aumenta con respecto a la solución que existía anteriormente. Este planteamiento se realiza teniendo en cuenta que los privilegios de toda la base de datos no van a estar asignados a un solo rol en el gestor, sino que van a estar seccionados teniendo en cuenta la cantidad de esquemas existentes y cada una de estas secciones va a estar asignada a rol diferente

2. Cuando se establece la conexión a nivel de rol:

a) Propicia la existencia de un rol de login en el gestor por cada rol de la aplicación. 1__,2__,3__,4__,5__
b) Los permisos se van a seccionar hasta el nivel de tablas, funciones, triggers, secuencias y vistas, pues para cada uno de estos recursos se van a definir exactamente los privilegios que cada rol de login necesita. 1__,2__,3__,4__,5__
c) Los permisos al rol de base de datos se le van a asignar teniendo en cuenta las acciones que tiene asignadas su homólogo en el sistema. 1__,2__,3__,4__,5__
d) Las transacciones hacia el gestor van a emplear un rol de login por cada rol de aplicación. 1__,2__,3__,4__,5__
e) Es posible conocer el rol perteneciente al usuario que accede al gestor. 1__,2__,3__,4__,5__

En este tipo de conexión la granularidad se incrementa con respecto al nivel anterior. Se puede arribar al planteamiento anterior teniendo en cuenta que existe una profundización en la gestión de políticas de control de acceso a la base de datos. El incremento en las políticas se ve reflejado en la posibilidad de seccionar los privilegios en el gestor hasta el nivel de tablas, funciones, secuencias, vistas y triggers, dando la posibilidad de que un rol pueda acceder solamente al recurso necesita, evitando que tenga permisos que nunca llegue a usar.

3. Cuando se establece a conexión a nivel de usuario:

a) Va a existir un rol de login en el gestor por cada usuario de la aplicación. 1__,2__,3__,4__,5__
b) Va a existir un rol de grupo en el gestor por cada rol de la aplicación. 1__,2__,3__,4__,5__
c) La granularidad aumenta hasta el nivel de tablas, funciones, triggers, secuencias y vistas, pues para cada uno de estos recursos se van a definir exactamente los privilegios que cada rol de grupo necesita. 1__,2__,3__,4__,5__
d) Las transacciones hacia el gestor van a emplear un rol de login por cada usuario de la aplicación. 1__,2__,3__,4__,5__
e) La granularidad aumenta, pues es posible conocer directamente el usuario que accede al gestor. 1__,2__,3__,4__,5__

En este tipo de conexión va a existir un aumento en la granularidad de la gestión de políticas de control de acceso a los datos con respecto al nivel anterior. A este planteamiento se arriba teniendo en cuenta esencialmente, la posibilidad de que cada usuario acceda directamente con un rol de login al gestor y no a través de otro que englobe a varios usuarios y la facilidad de conocer el usuario que accede al gestor.

Dr. Oiner Gómez Baryolo

Figura 32. Cuestionario, primer cliente.
Fuente: elaboración propia.

Lista de chequeo para la validación de la idea a defender del componente para gestionar políticas de control de acceso a nivel de base de datos en el sistema Acaxia.

El control de acceso a la información que se procesa en cualquier entidad debe estar respaldado por medidas que regulen la forma en que se manipulan los datos. La mayoría de la información sensible que manejan los Sistemas de Información se encuentra almacenada en los sistemas gestores de base de datos. El marco de trabajo Sauxe establecía anteriormente las conexiones a la base de datos para todos los sistemas que se encontraban suscritos a Acaxia con una única conexión que tenía los permisos (select, update, insert, delete, usage, execute, trigger) sobre todos los esquemas de la misma. El exceso de privilegios de un rol en el gestor de base de datos representa una brecha de seguridad que puede ocasionar pérdidas de la información que manejan los sistemas que utilizan Acaxia. Teniendo en cuenta el planteamiento anterior se decide desarrollar un componente que sea capaz de gestionar las políticas de control de acceso a nivel de datos en Acaxia con el objetivo de aumentar la granularidad de las políticas a dicho nivel.
En el marco de esta investigación se entiende como granularidad en el establecimiento de políticas de control de acceso a la posibilidad seccionar en partes más pequeñas los permisos sobre los recursos (esquemas, tablas, funciones, triggers, secuencias, vistas) a nivel de datos con el objetivo de poder establecer privilegios sobre un número menor de recursos a cada rol en el gestor. Para el logro de este objetivo se implementaron tres tipos de conexión además de la ya existente, conexión a nivel de sistema, a nivel de rol y a nivel de usuario. Para validar que el componente desarrollado aumenta la granularidad a nivel de datos, se elaboró esta lista de chequeo dirigida a los clientes que estuvieron involucrados en el proceso de desarrollo.
Evalúe cuánto ud. cree que aportan las siguientes políticas de control de acceso en el aumento de la granularidad en el sistema Acaxia, teniendo en cuenta un rango de valores de 1 a 5, donde 1 indica menor aporte y 5 mayor aporte.

Antes del desarrollo del componente de gestión de políticas de control de acceso:

1. **Cuando se establecía la conexión a nivel de aplicación o por defecto:**

   a) Todos los privilegios existentes en el gestor sobre cada uno de los recursos iban a pertenecer a un solo rol. 1__,2__,3__,4__,5X
   b) Todas las transacciones hacia el gestor iban a emplear un único rol de login. 1__,2__,3__,4__,5X
   c) No era posible determinar que usuario accede al gestor. 1__,2__,3__,4__,5X
   d) Los usuarios de la aplicación iban a poseer permisos en la base de datos que nunca usaban, pues responden a acciones que nunca van a tener asignadas. 1__,2__,3__,4__,5X

En esta solución se gestionaban débiles políticas de control de acceso a los datos. Esto se evidencia en la existencia de un solo rol para realizar todas las transacciones hacia el gestor, sin tener en cuenta el sistema, ni el rol, ni el usuario que realizaba la transacción, lo cual supone un exceso de privilegios en el gestor y una disminuida trazabilidad a este nivel por lo que la aplicación se encuentra poco granulada.

Después del desarrollo del componente de gestión de políticas de control de acceso:

1. **Cuando se establece la conexión a nivel de sistema:**

   a) Todos los privilegios sobre los recursos de cada esquema van a estar asignados a un rol distinto para cada esquema. 1__,2__,3__,4__,5X

Figura 33. Cuestionario, segundo cliente.
Fuente: elaboración propia.

b) Las transacciones hacia el gestor van a emplear un único rol de login por cada sistema. 1__,2__,3__,4__,5X
c) Es posible conocer a que sistema pertenece el usuario que accede al gestor. 1__,2__,3__,4__,5X
d) Los usuarios de la aplicación van a realizar acciones asociadas solamente al sistema al que pertenecen. 1__,2__,3__,4__,5X

La granularidad en este tipo de conexión aumenta con respecto a la solución que existía anteriormente. Este planteamiento se realiza teniendo en cuenta que los privilegios de toda la base de datos no van a estar asignados a un solo rol en el gestor, sino que van a estar seccionados teniendo en cuenta la cantidad de esquemas existentes y cada una de estas secciones va a estar asignada a rol diferente

2. Cuando se establece la conexión a nivel de rol:

a) Propicia la existencia de un rol de login en el gestor por cada rol de la aplicación. 1__,2__,3__,4__,5X
b) Los permisos se van a seccionar hasta el nivel de tablas, funciones, triggers, secuencias y vistas, pues para cada uno de estos recursos se van a definir exactamente los privilegios que cada rol de login necesita. 1__,2__,3__,4__,5X
c) Los permisos al rol de base de datos se le van a asignar teniendo en cuenta las acciones que tiene asignadas su homólogo en el sistema. 1__,2__,3__,4__,5X
d) Las transacciones hacia el gestor van a emplear un rol de login por cada rol de aplicación. 1__,2__,3__,4__,5X
e) Es posible conocer el rol perteneciente al usuario que accede al gestor. 1__,2__,3__,4__,5X

En este tipo de conexión la granularidad se incrementa con respecto al nivel anterior. Se puede arribar al planteamiento anterior teniendo en cuenta que existe una profundización en la gestión de políticas de control de acceso a la base de datos. El incremento en las políticas se ve reflejado en la posibilidad de seccionar los privilegios en el gestor hasta el nivel de tablas, funciones, secuencias, vistas y triggers, dando la posibilidad de que un rol pueda acceder solamente al recurso necesita, evitando que tenga permisos que nunca llegue a usar.

3. Cuando se establece a conexión a nivel de usuario:

a) Va a existir un rol de login en el gestor por cada usuario de la aplicación. 1__,2__,3__,4__,5X
b) Va a existir un rol de grupo en el gestor por cada rol de la aplicación. 1__,2__,3__,4__,5X
c) La granularidad aumenta hasta el nivel de tablas, funciones, triggers, secuencias y vistas, pues para cada uno de estos recursos se van a definir exactamente los privilegios que cada rol de grupo necesita. 1__,2__,3__,4__,5X
d) Las transacciones hacia el gestor van a emplear un rol de login por cada usuario de la aplicación. 1__,2__,3__,4__,5X
e) La granularidad aumenta, pues es posible conocer directamente el usuario que accede al gestor. 1__,2__,3__,4__,5X

En este tipo de conexión va a existir un aumento en la granularidad de la gestión de políticas de control de acceso a los datos con respecto al nivel anterior. A este planteamiento se arriba teniendo en cuenta esencialmente, la posibilidad de que cada usuario acceda directamente con un rol de login al gestor y no a través de otro que englobe a varios usuarios y la facilidad de conocer el usuario que accede al gestor.

Ing. René Rodrigo Bauta Camejo

Figura 34. Cuestionario, segundo cliente.
Fuente: elaboración propia.

Anexo 5. Acta de liberación del componente para la gestión de políticas de control de acceso a nivel de base de datos en el sistema Acaxia.

UCI Informáticas

CEIGE CENTRO DE INFORMATIZACIÓN DE LA GESTIÓN DE ENTIDADES

DEPARTAMENTO DE TECNOLOGÍA.
viernes, 24 de mayo de 2013

A quien pueda interesar:

Por este medio se hace constar que la solución Componente de gestión de políticas de control de acceso a nivel de base de datos para el sistema Acaxia de los autores José Eduardo Vázquez Hernández y Katia Saria Preval fue sometida a dos revisiones técnicas en las cuales se detectaron algunas no conformidades que fueron resueltas en su totalidad quedando esta solución estable y lista para su posterior uso.

Además los autores entregaron los artefactos que a continuación se describen:

1. Modelo conceptual
2. Documentos de descripción de requisitos.
3. Modelos de diseño
4. Diseños de casos de prueba
5. Manual de usuario

Para que conste firman a continuación los miembros del equipo que realizaron las revisiones, el autor y el tutor del trabajo.

Dado a los 27 días del mes de Mayo de 2013.

Nombre y apellidos	Firma
Revisores: René Rodigo Bauta Camejo. Andrés Reynaldo Hilord Hilenday H. Sardany Pooy	
Autor (es): Katia Soria Preval José Eduardo Vázquez Hernández	
Tutor (es): Hilenday H. Sardany Pooy Andrés Reynaldo Hilord	

Jefe de Departamento de Tecnología
René R. Bauta Camejo

CEIGE

Figura 35. Acta de liberación.
Fuente: elaboración propia.

Glosario de términos

A.

Artefactos: productos tangibles del proyecto que son creados, modificados y usados dentro de las actividades. Pueden ser modelos, elementos dentro del modelo, código fuente y ejecutables.

B.

Brecha de seguridad: vía por la que se puede atacar la seguridad de un sistema.

C.

Componente: unidad de composición de aplicaciones de software, que posee un conjunto de interfaces y un conjunto de requisitos, y que ha de poder ser desarrollado, adquirido, incorporado al sistema y compuesto con otros componentes de forma independiente, en tiempo y espacio.

E.

Entidad: forma parte de la estructura de un país y puede comportarse como un ministerio, empresa, asociación, área o cargo.

H.

Herramienta: es un objeto elaborado a fin de facilitar la realización de una tarea. Sirve como recurso.

M.

Métrica: medida cuantitativa del grado en que un sistema, componente o proceso posee un atributo dado.

R.

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

S.

Sistema: es un conjunto organizado de objetos o partes interactuantes e interdependientes, que se relacionan formando un todo unitario y complejo.

Software: conjunto de instrucciones que los ordenadores emplean para manipular datos.

V.

Validación: confirmación mediante el suministro de evidencia objetiva de que se han cumplido los requisitos para una utilización o aplicación específica prevista.