UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

FACULTAD 3

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

Título: ***"Desarrollo de una herramienta informática para la identificación y reparación de inconsistencias de estructura y datos entre bases de datos PostgreSQL".***

Autores: Odisleysi Martinez Furones

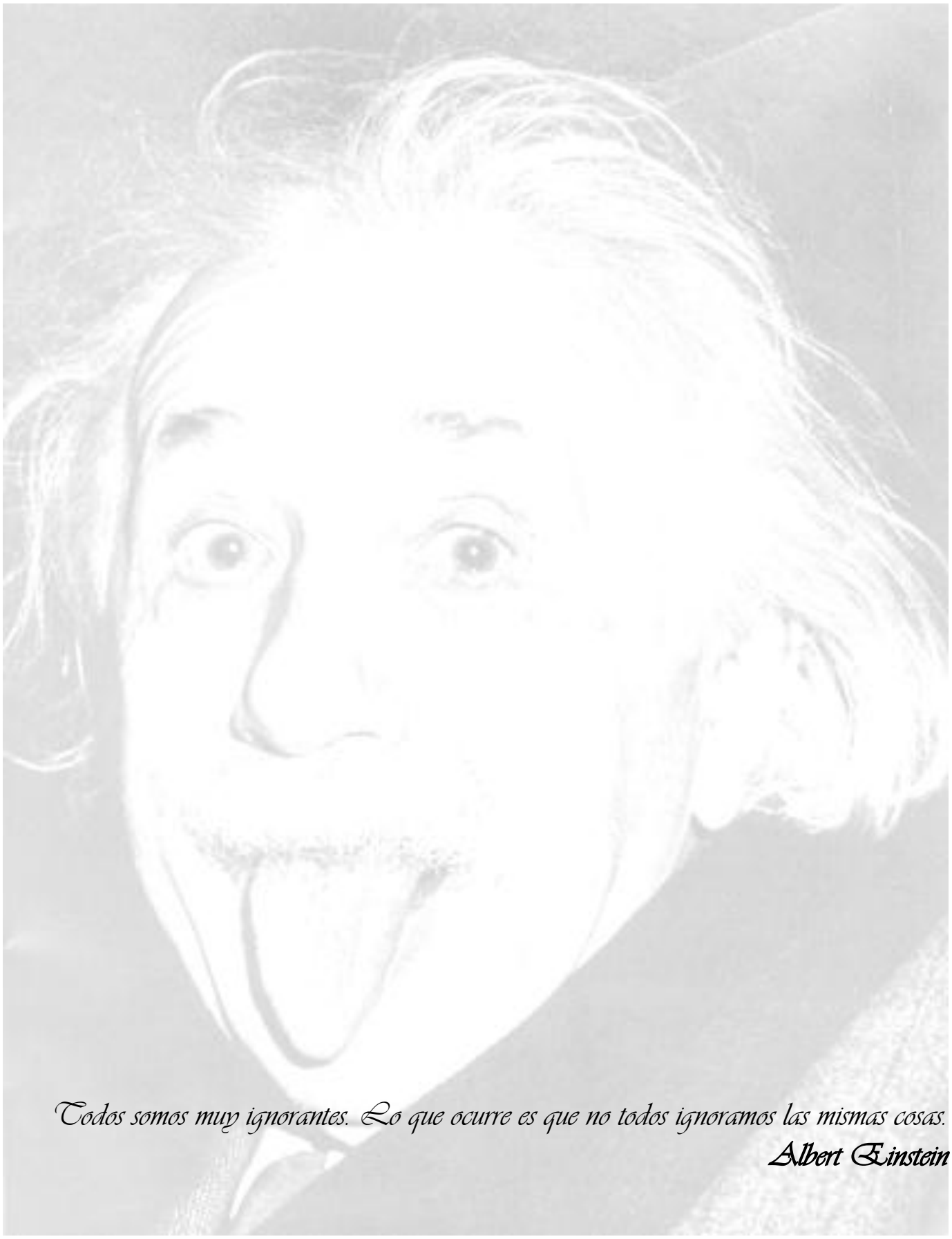
Eddy Figueredo Aguilar

Tutor: Ing. Juniel Tamayo Hernández

Co-tutor: Ing. Carlos Rafael Rodríguez Rodríguez

La Habana, Cuba

2012



Todos somos muy ignorantes. Lo que ocurre es que no todos ignoramos las mismas cosas.

Albert Einstein

Agradecimientos

De Odisleysi

A mi madre por ser un ejemplo de mujer, por ser tan sacrificada para con su familia y su trabajo, por ser una buena revolucionaria, por estar en los momentos buenos y malos, por no dejarme sola y confiar en mí en todo momento, incluso cuando yo misma no creía, por ser la principal causa de que todo esto fuera posible y sobre todo por ser madre y padre en todo momento. Mima te quiero con la vida y espero que estés orgullosa de mí como yo lo he estado toda la vida de ti.

A mis hermanos Osiel y Oniel, los quiero mucho. Espero Osi que pronto encuentres tu camino como yo encontré el mío. Gracias por estar ahí cuando te necesité, por ser un apoyo constante dándome fuerzas para seguir adelante y por confiar en que este día llegaría, que aunque no me lo dices mucho, con tus acciones me lo haces saber.

A mi padrastro Mario, por ser un ejemplo de sacrificio, por apoyarnos y alentarnos a seguir nuestras metas.

A mis abuelos por parte de madre, por la confianza que me han dado durante toda mi vida.

A mis tías por la educación que recibí de ellas, por ser todas en la misma medida mis segundas madres, por darme el amor que me han dado, por preocuparse y tratarme como una hija. Tías las quiero, ustedes han sido también mi ejemplo.

A mis primos que sin ellos no sería la persona que soy hoy.

A mi familia por parte de madre, por estar cuando hizo falta, por darme el amor y equilibrio de una familia inmensa y muy unida.

A mi novio Erick León Bolinaga, por ser mi apoyo, por brindarme su amor incondicional, por dedicarme sus horas ayudándome en la realización de la Tesis y sobre todo por guiarme y exhortarme a que continúe estudios.

A Carlos Enrique Rivero Medina por confiar en que llegaría hasta aquí, por darme fuerzas en los momentos difíciles, por ser mi apoyo espiritual durante mucho tiempo, por impulsarme a cada día a aprender más sobre nuestra carrera.

A mis amigos del Politécnico, Bárbara, Katia, Leonel y Erick por estar cuando los necesité y por seguir siendo mis amigos a pesar de la distancia.

A mis amigos de la Universidad, Carmen y Jorge, por la ayuda brindada en el desarrollo de la Tesis, por alentarme cada vez que hizo falta. Los adoro y espero que nuestra amistad perdure.
A mis compañeros de grupo, Yordenys y Darío, por ayudarme en algunas asignaturas, y sobre todo a Yordenys, por ser la ayuda primordial en el desarrollo de la herramienta. Muchas gracias, por los días brindados de tu tiempo, de todo corazón.
A todos los profesores que de una forma u otra tomaron parte activa en mi formación.

De Eddy

A mi madre y mi padre, a mis hermanas, mis abuelas y abuelos, mis tíos y tías, a mis primos, en fin a mi familia, a mi novia que ya es también mi familia, a mis amigos. A todos los que me ayudaron a terminar la carrera y a terminar esta Tesis, en fin aquí va mi lista: Yailin, Yordenys, Ale y René, la gente del apto, a los profes que me supieron responder, y que quisieron ayudarme, en fin agradecimientos a todos los que le interese saber que estoy agradecido y que en algún momento estuvieron cerca.

Dedicatoria

De Odisleysi

A mi madre, porque eres la luz de mis días, por tu ejemplo, tu amor inmenso, por no rendirte nunca, por ser luchadora y ser modelo a seguir en tu centro de trabajo y en tu casa. Te adoro mamita, espero poder darte todo lo que desees, y que la salud te acompañe, para que tengas la dicha de disfrutarla con tus hijos.
A mi hermano Osiel, ya eres un hombre, te he visto crecer y no me lo creo, has pasado de ser el niño más intranquilo a ser un hombre de bien, espero que la vida te depare cosas maravillosas como te las mereces.
A mi padrastro Mario, por ser parte de nuestra familia y ser lo más cercano que he tenido como padre.
A mi familia, por ser tan maravillosamente unida.
A mi novio Erick, por hacer mis días más felices.

De Eddy

A mi madre y mi padre, a mis hermanas, mis abuelas y abuelos, mis tíos y tías, a mis primos, en fin a mi familia, a mi novia que ya es también mi familia, a mis amigos, creo que en esta lista están todos los responsables de ser quien soy.

Resumen

El manejo de los datos mundialmente se ha hecho más eficiente con el surgimiento de las bases de datos (BD), dando paso a la necesidad de crear una forma para garantizar la seguridad de estos datos, de ahí surge el respaldo a la información almacenada. La redundancia de la información almacenada en las BD, es generalmente la principal causa de la ocurrencia de inconsistencias, incluyendo además los procesos de actualización o cualquier cambio que se haga en el estado de las mismas que no sea llevada a su respaldo. Mediante el presente trabajo se realiza un análisis del proceso de actualización en las BD y entre BD y su respaldo, con el objetivo de entender las causas de las apariciones de las inconsistencias y modelar una herramienta informática capaz de tratarlas, para luego implementarla. En la primera parte del mismo se realiza un estudio del estado del arte seleccionándose la estrategia para llevar a cabo la tarea. Se establece además una metodología para guiar el proceso de desarrollo del software, que incluye el desarrollo de los artefactos necesarios para implementar y probar la herramienta informática deseada.

Palabras claves

Herramienta informática, inconsistencias, proceso de actualización, redundancia, respaldo

Índice

Índice de figuras

Índices de tablas

Introducción

En la actualidad el mundo se maneja a través de la información, la cual está creciendo considerablemente en las instituciones y empresas. Un estudio de la Corporación Egan Marino C. (**EMC**^{2}) -corporación, fabricante mundial en soluciones de infraestructura de la información-, asegura que en el 2010 la cantidad de información generada mundialmente estuvo cerca de 1,2 zettabytes. Un zettabyte equivale a 1 trillón gigabytes. En 2009, y en medio de la gran recesión económica, la cantidad de información digital creció en un 62 %, en comparación con el 2008, donde se lograron 800 billones gigabytes o 0,8 zettabytes, con una tasa de crecimiento anual del 60 %, el universo digital está creciendo rápidamente y se estima que la información digital alcanzará los **1.8** zettabytes para el año 2011 (ANON. 2010 a)(EMC Corporation 2008).

Para contribuir a la organización y manipulación eficiente de la información se crearon los sistemas gestores de bases de datos (en lo adelante SGBD), los cuales influyen en el rendimiento, en el buen uso de las BD, en su correcto funcionamiento, y en su seguridad, ya que implican un mayor control sobre las mismas (Sumathi, Esakkirajan 2007). Sin embargo algunos de los inconvenientes o desventajas de los SGBD están dados por su tamaño, pues los SGBD son programas que requieren una gran cantidad de espacio en disco y de memoria para trabajar de forma eficiente. Otro de los inconvenientes es el costo económico que tienen, ya que varían en dependencia de la funcionalidad que ofrezca, del entorno y el mantenimiento que suele ser un porcentaje del precio del SGBD anualmente. Las prestaciones también constituyen otra de las desventajas debido a que los SGBD están desarrollados para ser útiles en muchas aplicaciones. Esto trae consigo que algunas de ellas no sean tan rápidas como en los sistemas de archivos. Los SGBD también son vulnerables a los fallos, como todos los datos están centralizados el sistema se hace más vulnerable ante los fallos que se produzcan. Por tanto, es importante tener en cuenta que, no obstante a la existencia de los SGBD, no se puede afirmar que las BD se vean inmunes a afectaciones de algún tipo (Cristina Manresa Yee 2005).

Otra de las afectaciones a las BD puede ocurrir debido a que los archivos que mantienen almacenada la información son creados por diferentes tipos de programas de aplicación y existe la posibilidad de que si no se controla detalladamente el almacenamiento, se pueda originar un duplicado de información.

Esto aumenta los costos de almacenamiento y acceso a los datos, además de que origina la inconsistencia de los datos, la cual ocurre cuando existe información contradictoria o incongruente en la BD, es decir diversas copias de un mismo dato no concuerdan entre sí, por ejemplo: se actualiza la dirección de un cliente en un archivo y que en otros archivos permanezca la anterior (Martínez 2002).

Se pueden identificar distintos tipos de inconsistencias, dentro de las que se encuentra: de datos, estructurales, permisos, funciones y disparadores. Encontrar estas diferencias es muy engorroso cuando se trata de una gran cantidad de información por lo que sería necesaria una forma de facilitar la búsqueda de las mismas.

Las BD también pueden presentar afectaciones referentes a la pérdida de la información. Según la Máquina internacional de negocio (IBM[1]) el 17,5 % de la pérdida de la información se debe a sabotajes o hechos terroristas, otro 17,5 a accidentes, un 14,0 a eventos atmosféricos, el 7,0 a inundaciones, los terremotos representan un 10,5 % mientras que se reporta un 8,8 por errores de software, 9,5 por caídas eléctricas o fallos eléctricos, otros factores definidos fueron por fallos de red y roturas de conducción con 3,5 % respectivamente, en tanto, por errores de hardware se identificó un 5,3 % y un 2,8 para otras eventualidades no definidas dentro de las mencionadas (Espinosa 2009).

Como se ha podido observar, las afectaciones que presenta una BD son bastante diversas, y en su conjunto hacen que aumente la probabilidad de que la misma presente fallos en algún momento. Debido a esto las empresas u organismos emplean generalmente las copias de respaldo para evitar la pérdida total o parcial de la información. Generalmente el contenido a salvaguardar se define de acuerdo con su importancia. Es importante tener en cuenta que estas copias de respaldo deben tener la garantía de poder recuperarlas una vez que ocurra un fenómeno determinado. Dentro de los mecanismos o tipos de copias en función de la cantidad de información con la que se trabaje se encuentran: la copia de seguridad total o íntegra, la copia de seguridad incremental, y la copia de seguridad diferencial (ANON. 2010 b).

Sin embargo, estos mecanismos, no garantizan la seguridad y disponibilidad total de la información pues puede darse el caso de existir diferencias en la copia de respaldo. Para identificar estas diferencias se han creado herramientas entre las que se destacan EMS DB Comparer for PostgreSQL (EMS Software Development 2011) y PostgreSQL Data Sync (SQL Maestro Group 2012 b). Estas herramientas están hechas para Windows, comparan y sincronizan el contenido de la BD y no resuelven todos los tipos de diferencias.

PostgreSQL es uno de los SGBD más conocidos y está basado en POSTGRES: sucesor del IGRES SGBD desarrollado durante el período comprendido entre los años 1975 a 1977 (Stonebraker, Rowe 1986), Versión 4.2, desarrollada en la Universidad de California Berkeley en el Departamento de Informática. POSTGRES abrió camino a muchos conceptos que solo se pusieron disponibles tiempo después en algunos sistemas de BD comerciales. PostgreSQL es un descendiente del código-abierto del código original de Berkeley. Apoya en gran parte al estándar SQL y ofrece muchos rasgos modernos. Además, el usuario puede crear nuevos tipos de datos, funciones, así como operadores, entre otros. Debido a la licencia libre, PostgreSQL puede usarse, puede modificarse, y puede distribuirse por todos de manera gratis para cualquier propósito, sea privado, comercial, o académico (The PostgreSQL Global Development Group 1996 a).

Debido a sus prestaciones, a sus características y a su licencia, hoy en día es uno de los SGBD más utilizados mundialmente.

[1] **IBM:** International business machines, empresa multinacional estadounidense de tecnología y consultoría, fabrica y comercializa hardware y software para computadoras, entre otros.

En la actualidad la versión más utilizada en la facultad 3 de la Universidad de las Ciencias Informáticas (UCI) es la 8.4 debido a que fue liberada en el 2009 con más de 250 mejoras respecto a sus anteriores versiones, dentro de las que se encuentra: la restauración de BD en procesos paralelos acelerando la recuperación de un respaldo hasta 8 veces, los privilegios por columna, la configuración de ordenamiento configurable por BD, las nuevas herramientas de monitoreo de consultas que le otorgan a los administradores mayor información sobre la actividad del sistema (León, Rodríguez, Veitia 2011).

Por lo antes expuesto se enfrenta la siguiente **situación problemática**: ha aumentado el volumen de la información almacenada, así como la complejidad de los mecanismos utilizados para su almacenamiento, los SGBD y las herramientas de administración existentes no garantizan la integridad de la información, todos los SGBD están propensos a la ocurrencia de fallas que provoquen la pérdida total o parcial de la información y no existe una herramienta informática libre que sea capaz de identificar y reparar las inconsistencias que se presentan entre BD PostgreSQL.

Teniendo en cuenta la situación problemática y por todo lo antes planteado surge el siguiente problema: ¿Cómo resolver las inconsistencias de estructura y de datos que se detectan entre BD PostgreSQL? Dado este **problema** se presenta como **objeto de estudio:** Las BD PostgreSQL y como **objetivo general**: Desarrollar una herramienta informática que permita detectar y eliminar las inconsistencias de estructura y de datos que ocurran en las BD PostgreSQL, contribuyendo a la integridad de la información almacenada en las BD de este gestor; estableciendo como **campo de acción**: Resolución de inconsistencias de estructura y datos en BD PostgreSQL.

Definiendo como **objetivos específicos:**

- ✓ Elaborar el marco teórico de la investigación.
- ✓ Desarrollar el diseño de la herramienta informática.
- ✓ Implementar y probar la herramienta informática desarrollada.

Teniendo como **idea a defender** que si se desarrolla una herramienta para el diagnóstico y reparación de inconsistencias de estructura y de datos en BD PostgreSQL se podrá contribuir a la integridad de la información almacenada en las BD de este gestor.

Para cumplir con el objetivo planteado se definen las siguientes **tareas de la investigación:**

- **Odisleysi Martinez Furones**
  - ✓ Analizar las principales herramientas que contribuyan a la solución del problema determinado.
  - ✓ Analizar los requerimientos funcionales y no funcionales de la herramienta informática a desarrollar.
  - ✓ Desarrollar el modelo de diseño de la herramienta informática.
  - ✓ Realizar pruebas a la herramienta informática desarrollada.
  - ✓ Redactar el trabajo de diploma donde se exponen los principales resultados del trabajo.
- **Eddy Figueredo Aguilar**
  - ✓ Desarrollar los artefactos definidos para la implementación del sistema: Diagrama de Clases y

Diagrama de Componentes.

- ✓ Analizar las principales tecnologías y herramientas a utilizar para el desarrollo de la solución propuesta.
- ✓ Implementar los casos de uso (en lo adelante CU) necesarios para el funcionamiento de la herramienta informática.
- ✓ Diseñar los casos de pruebas de integración, de caja negra y caja blanca para el sistema y aplicarlas para validar las funcionalidades implementadas.
- ✓ Redactar el trabajo de diploma donde se exponen los principales resultados del trabajo.

Diseño metodológico

Método teórico análisis histórico-lógico: Para realizar la recopilación de la información sobre las BD PostgreSQL, su trayectoria y comportamiento con respecto al fenómeno de la inconsistencia de datos, además de la investigación de las leyes generales del funcionamiento y desarrollo de este fenómeno.

Método teórico modelación: Para realizar el diseño de una propuesta de solución a la problemática planteada.

Método teórico Análisis-Sintético: Para analizar las teorías, documentos, estudiar la bibliografía referente al tema, etc.

Método empírico entrevista: Para identificar la Problemática y el Problema a resolver se hizo necesario utilizar este método haciéndoles encuestas a algunos arquitectos de BD de distintos proyectos de la UCI, dentro de los que se encuentra el Ing. Juniel Tamayo Hernández, arquitecto de BD del Proyecto División de Antecedentes Penales de la República Bolivariana de Venezuela (RAP) y el Ing. Vlamir Rodríguez Fernández administrador de BD del proyecto Fiscalía fase 1.

La investigación reflejada en el presente documento se encuentra estructurada en 3 capítulos y varios anexos donde se detalla todo lo referente al diseño y la implementación de la herramienta informática. Estos se exponen a continuación:

Capítulo 1: Fundamentación teórica.

Se analizan los distintos tipos de inconsistencias que se presentan entre BD PostgreSQL, y varias de las herramientas existentes para la solución de inconsistencias de BD. Además, se analizan las tecnologías y la arquitectura que se va a utilizar en el desarrollo de la herramienta.

Capítulo 2: Modelo de diseño.

Se analizan los requisitos del sistema y se desarrolla el modelo de diseño de la herramienta, basado en la arquitectura que mejor se adecue a los requerimientos planteados para la herramienta.

Capítulo 3: Implementación y pruebas.

Se exponen los estándares de programación utilizados en la implementación. Se desarrolla y prueba la herramienta.

Capítulo 1: Fundamentación teórica

En el presente capítulo se estudiarán los distintos tipos de inconsistencias que se presentan entre BD PostgreSQL, y algunas de las herramientas existentes para la solución de inconsistencias de BD. Además, se analizarán las tecnologías y la arquitectura que se van a utilizar en el desarrollo de la herramienta.

1.1. Bases conceptuales

Definir los conceptos principales ayuda al entendimiento y fundamenta las bases para la realización de la indagación, a continuación se desglosan los conceptos más importantes para la investigación:

1.1.1. SGBD

Los SGBD consisten en la recopilación de datos interrelacionados y un conjunto de programas para acceder a los datos. Son software que ayudan a mantener y utilizar una BD (Sumathi, Esakkirajan 2007).

1.1.2. PostgreSQL

PostgreSQL es un sistema de gestión de bases de datos objeto-relacional (SGBDOR), distribuido bajo licencia BSD[2] y con su código fuente disponible libremente. Es el SGBD de código abierto más potente del mercado y en sus últimas versiones no tiene nada que anhelar a otras BD comerciales. (The PostgreSQL Global Development Group 1996 b)

1.1.3. Inconsistencias de BD PostgreSQL

Las BD tienen como principal objetivo almacenar la información. Teniendo en cuenta el incremento diario de esta, es necesario lograr la creación de BD con un diseño óptimo para evitar los problemas que trae consigo la redundancia de la información, así es como se eliminan los datos repetidos o los problemas al realizar cualquier transformación en la misma. Generalmente cualquier anomalía con la BD surge a partir de una operación realizada.

Las causas de la ocurrencia de la inconsistencia de datos en un sistema pueden ser accidentales o intencionales (con fines indebidos). Dentro de las causas intencionales, se tiene el robo de la información o la lectura sin autorización y la modificación o la destrucción no autorizada. En las causas accidentales están: las fallas de conexión durante el procesamiento de transacciones, fallas del hardware debido al medioambiente o catástrofes, anomalías por acceso concurrente, anomalías que resultan de la distribución de los datos entre varios servidores y un error lógico que viole la suposición de que las transacciones respetan las restrictivas de consistencia de los datos o un error cometido por algún usuario al introducir datos (Rose 1993).

[2] **Licencias BSD** son una familia de licencias de software libre permisiva. La licencia original se utilizó para la Berkeley Software Distribution (BSD). La licencia BSD permite la redistribución, uso y modificación del software.

Las anomalías en BD son ciertos problemas que aparecen con frecuencia en el manejo de las mismas cuando el diseño no ha sido realizado de forma normalizada, o sea, no se han diseñado teniendo en cuenta un conjunto de reglas para evitar problemas de lógica, estas reglas son conocidas como formas normales. Las anomalías básicas que surgen a partir de transformaciones que se hagan en las BD se definen como:

- **Anomalía de inserción**: Imposibilidad de dar de alta una tupla por no disponer del valor de un atributo principal.
- **Anomalía de borrado**: Pérdida de información por dar de baja una tupla.
- **Anomalía de modificación**: Tiene que ver con la redundancia[3]. En general, la normalización reduce la redundancia, pero no la elimina por completo.

La inconsistencia ocurre cuando existe información contradictoria o incongruente en la BD, es decir diversas copias de un mismo dato no concuerdan entre sí. Esto ocurre debido a la existencia de redundancia en la BD, si por algún motivo la actualización de alguno de los datos redundantes fallara se estaría en presencia de inconsistencia. Los tipos de inconsistencias que se identificaron en las BD son los siguientes:

Inconsistencia de datos:

Los SGBD garantizan la integridad de la información, teniendo en cuenta la no existencia de redundancia, ya que esta significa en muchos casos la posibilidad de inconsistencia en la información, pues la **inconsistencia de datos** se identifica a partir de transformaciones que se hagan en las BD, surgiendo a partir de las anomalías de actualización, es decir, cuando un dato redundante es actualizado en una parte, y no coincide en otra.

Inconsistencia de estructura:

Las inconsistencias de estructura, como su nombre lo indica, tienen mucho que ver con la estructura de la BD y por lo tanto se definen en el diseño de la misma. Las **inconsistencias de estructura** son las diferencias estructurales que se dan en el diseño o las relaciones entre objetos de las BD, entre una BD origen y su respaldo, es decir, debido a cualquier problema o anomalía de actualización al cambiarse el nombre de una tabla o la relación entre ellas en una BD origen, no se hace en su respaldo quedando esta última inconsistente con respecto a su origen.

Inconsistencias de programación:

Se definen como las inconsistencias que surgen al realizar cambios en las consultas para el trabajo con las BD, al no realizarse los cambios en la copia de respaldo de la misma. Es decir, las diferencias estructurales surgidas en las consultas que afectan la respuesta o el funcionamiento de estas.

[3] **Redundancia:** repetición de la misma información en tuplas diferentes y consiguiente necesidad de propagar actualizaciones.

La herramienta que se pretende desarrollar resolverá las inconsistencias de datos, de estructura e inconsistencias específicas de programación, pues solo se tratan las relacionadas a la estructura de la programación de las vistas, además del nombre o definición de las funciones, disparadores, reglas, secuencias, restricciones e índices.

1.2. Herramientas para la detección y corrección de inconsistencias de BD

Para identificar y corregir las inconsistencias de bases datos mundialmente existen herramientas, algunas de las más relevantes se describen a continuación y se tomó como criterio de revisión sus tipos de licencias, los SGBD, sistema Operativo (SO), e inconsistencias que trabajan.

1.2.1. EMS DB Comparer for PostgreSQL 3.3

La herramienta EMS DB Comparer for PostgreSQL permite comparar y sincronizar BD o esquemas en diferentes servidores, así como en un único servidor, la comparación de todos los objetos de BD o solo los seleccionados, la comparación de la totalidad o de propiedades de los objetos seleccionados solamente. Muestra una representación visual de las diferencias entre BD con los detalles y secuencias de comandos de modificación para los diferentes objetos. Posee la capacidad para sincronizar BD de forma manual paso a paso o de forma automática, generar informes con diferencias de BD, añadir informes personalizados y automatizar la comparación y sincronización de BD utilizando la aplicación de consola. Permite trabajar con varios proyectos a la vez, comparar, guardar y cargar proyectos con todos sus parámetros, una amplia variedad de opciones para la comparación y sincronización. Es editor de secuencias de comandos de SQL integrado, con resaltado de sintaxis y es compatible con la versión más reciente de PostgreSQL (EMS Software Development 2010).

Inconvenientes:

Realiza comparación estructural solamente, es software propietario, solo su licencia shareware oscila desde los $1057.5 pesos cubanos hasta $2992.5 en dependencia del tipo de organización al que va dirigido, ya sea empresa o usuario independiente, trabajando sobre el SO Windows.

1.2.2. dbForge Schema Compare for SQL Server 1.10

El software dbForge Schema Compare for SQL Server es una herramienta confiable y fácil de usar; esta permite al usuario realizar la comparación y la sincronización de las estructuras de BD de Microsoft SQL Server. Permite que los usuarios fácilmente puedan realizar el análisis de las diferencias existentes en la estructura de las BD con solo echar un vistazo y con esto hacer extensivos los cambios al servidor de Microsoft SQL requerido dejando a un lado ese trabajo manual. Su plataforma es Windows (Devart 2004).

Inconvenientes:

Es software propietario, tiene un costo de $3748.75 pesos cubanos y es solo para Windows.

1.2.3. D-Softs Database Comparer 2.0

D-Softs Database Comparer (DBC) es una herramienta de comparación de BD SQL profesional y potente. Está diseñada para automatizar la migración de datos, analizar los datos corruptos y restaurar datos de copias de seguridad. No solo puede usarse para comparar los esquemas de dos BD, sino que los contenidos de ambas BD pueden compararse también. Soporta treinta tipos de objetos de BD MSSQL. Gracias a la fuerte compatibilidad, se puede comparar y sincronizar BD MSSQL 2000 o 2005 rápida y fácilmente. Habilitando la sintaxis resaltada, los resultados de comparación y sincronización son mostrados con claridad. DBC puede comparar el esquema de dos BD y localiza rápidamente las diferencias en las tablas, vistas, procedimientos almacenados o cualquier otro objeto en la BD. Es compatible con las especificaciones definidas por el usuario clave, así como la clave principal y clave única (D-Softs 2009).

Inconvenientes:

Es software propietario, tiene un costo de $4975 pesos cubanos, solo funciona para Microsoft SQL Server y es para el SO Windows.

1.2.4. PostgreSQL Data Sync

PostgreSQL Data Sync es una herramienta profesional diseñada para comparar y sincronizar el contenido de BD PostgreSQL. Permite la creación automática de secuencias de comandos de sincronización, además de un control total sobre la comparación y sincronización. Guarda todas las opciones en un archivo de proyecto para volver a ejecutar de forma instantánea (SQL Maestro Group 2012 a).

Ventaja:

Permite comparar dos esquemas de BD.

Inconveniente:

Es propietaria contando con un periodo de evaluación de 15 días. (SQL Maestro Group 2012 a)

1.2.5. DreamCoder for PostgreSQL

DreamCoder for PostgreSQL cuenta con características que le permiten realizar rápida y fácilmente todas las actividades de desarrollo de la BD, como crear y compilar procedimientos almacenados, exportar e importar datos, generar reportes, monitorear la actividad de la BD, sincronizar la BD, construir y ejecutar consultas y formatear el código SQL, entre muchas otras características. Soporta todas las versiones del servidor de BD PostgreSQL desde la versión 8.0 hasta la versión 8.3 (Mentat Technologies 2009).

Ventaja:

Utilidad para sincronizar dos esquemas de la BD.

Inconveniente:

Solo para el SO Windows. (Peter, Mentat Technologies Database Solutions 2012)

1.2.6. Conclusiones parciales

Luego de haberse descrito las herramientas se hace necesario mostrar una tabla donde se puedan observar las debilidades y fortalezas de cada una:

Herramientas	***Licencia***	***SGBD***	***SO***	***Inconsistencia que trabaja***
EMS DB Comparer for PostgreSQL 3.3	propietario	PostgreSQL	Windows	contenido, estructura
dbForge Schema Compare for SQL Server 1.10	propietario	Oracle, MySQL	Windows	contenido, estructura
D-Softs Database Comparer 2.0	propietario	MSSQL	Windows	esquemas, contenido
PostgreSQL Data Sync	propietario	PostgreSQL	Windows	contenido
DreamCoder for PostgreSQL	libre	PostgreSQL	Windows	contenido, estructura
Nueva herramienta	libre	PostgreSQL	Windows, Linux	contenido, estructura, programación

Tabla 1. Comparación entre las herramientas para la detección y corrección de inconsistencias de BD

El estudio realizado arrojó la existencia de varias herramientas que realizan la comparación y sincronización de BD, destacando solo las mencionadas, para el SGBD PostgreSQL. Mostró además que existen herramientas para trabajar con el gestor que incluyen entre sus funcionalidades la sincronización, tal es el caso de DreamCoder for PostgreSQL. Además, se estudiaron otras herramientas que no trabajan con PostgreSQL con el objetivo de analizar las funcionalidades necesarias y sacar definiciones sobre las inconsistencias a tener en cuenta. Se concluye entonces que la vía factible para resolver el problema es desarrollar una nueva herramienta, ya que ninguna de las herramientas expuestas resuelve todas las inconsistencias antes analizadas, además de que la mayoría son propietarias y corren sobre el SO Windows. La herramienta que se desarrollará es para la versión 8.4 de PostgreSQL aunque se pretende probar y extender a PostgreSQL 9.x.

1.3. Funciones de similitud sobre cadenas de texto

Como bien se ha tratado en epígrafes anteriores, las BD revolucionaron el mundo moderno, ya que almacenan grandes volúmenes de información, sin embargo, estas pueden presentar información duplicada que en ocasiones llevan a una decisión equivocada. El proceso que detecta este conflicto se conoce en el ámbito de las BD como: merge-purge, data deduplication o instance identification, así como detección de duplicados.

En 1946 el primero en plantear el proceso de detección de duplicados fue Dunn[4], al transcurrir de los años se fue perfeccionando, en el año 2000, Winkler[5] propuso mejoras al modelo. En su forma más simple, dicho proceso es como sigue:

Dado un conjunto R de registros.

[4] Doctor en Medicina **Halbert L. Dunn,** (1896-1975) fue la figura principal en el establecimiento del sistema nacional de estadísticas vitales en los Estados Unidos (Dunn [no date]).

[5] Creador de la función de distancia de cadena Jaro-Winkler (Winkler 1999).

1) Se define un umbral real ø ∈ [0,1].

2) Se compara cada registro de R con el resto.

3) Si la similitud entre una pareja de registros es mayor o igual que ø, se asumen duplicados; es decir, se consideran representaciones de una misma entidad real.

Para que la función tenga efectividad es necesaria alguna función de similitud que, dados dos registros con la misma estructura, devuelva un número real en el intervalo [0,1] igual a uno si ambos registros son idénticos y menor cuanto más diferentes sean. Tal valor depende de la similitud entre cada pareja de atributos respectivos. Luego, es necesaria otra función de similitud al nivel de atributo (no de registro), siendo frecuente en este contexto tratar los atributos, sin importar su tipo de datos, como cadenas de texto (strings). Estas funciones se clasifican en dos categorías: basadas en caracteres y basadas en palabras completas o tokens.

Se empleó una variante de las funciones de similitud basadas en palabras completas o tokens, en forma de algoritmo.

Las funciones de similitud basadas en tokens consideran cada cadena como un conjunto de subcadenas separadas por caracteres especiales, como por ejemplo espacios en blanco, puntos y comas, esto es, como un conjunto de tokens, y calculan la similitud entre cada pareja de tokens mediante alguna función de similitud basada en caracteres. Una de ellas es:

- Similitud de Monge-Elkan

Dadas dos cadenas A y B, sean $\alpha_1, \alpha_2 \ldots \alpha_k$ y $\beta_1, \beta_2 \ldots \beta_L$ sus tokens respectivamente. Para cada token α_i existe algún β_j de máxima similitud. Entonces la similitud de Monge-Elkan entre A y B es la similitud máxima promedio entre una pareja (α_i, β_j) (Monge, Elkan 1996).

Gelbukh[6] y otros presentan un modelo basado en la media aritmética generalizada, en lugar del promedio, el cual supera al modelo original sobre varios conjuntos de datos. El algoritmo para la similitud de Monge-Elkan tiene un orden computacional O(nm) (Amón, Jiménez 2009).

Esta similitud es la aplicada en el algoritmo utilizado para la implementación de la funcionalidad que permite comparar las BD, utilizando todos sus datos como conjuntos y para este caso particular, se define que la similitud debe ser máxima o la distancia entre cadenas del conjunto o tokens debe ser cero, en caso contrario nos encontramos en presencia de una inconsistencia. A continuación se reflejan funciones de distancia que fueron analizadas para obtener el resultado esperado.

1.3.1. Distancia de Levenshtein

Esta función de distancia compara dos cadenas o conjuntos devolviendo un número que sería la cantidad de pasos u operaciones de edición que son necesarias para convertir una cadena en otra, es considerada una operación de edición cualquier cambio al que sea objeto una cadena, ya sea adicionar un elemento, eliminarlo o cambiarlo. Desde el punto de vista informático tiene una complejidad O (n2)

[6] Prof. Dr. Alexander **Gelbukh** doctorado en Ciencias de la Computación. Académico de la Academia Mexicana de Ciencias (AMC).

siendo esto su principal debilidad pues n sería la cantidad de caracteres que se encuentran en la cadena más grande por lo que no se aconseja su uso para cadenas muy grandes (Levenshtein 1966).

1.3.2. Distancia de Hamming

En teoría de la información, la distancia de Hamming entre dos cadenas de igual longitud viene dada por el número de posiciones en las que los símbolos correspondientes son diferentes, en este caso tokens o datos. Mide el número mínimo de transformaciones que es necesario para convertir una cadena en otra (Hamming 1950).

1.3.3. Subsecuencia común más larga (SCML)

Es uno de los modelos computacionales más usados pues sirve como medida de similitud entre dos secuencias por medio de su longitud, consiste en obtener la secuencia común de caracteres más largos entre dos cadenas que se muestra con los símbolos que contienen en común (Soto, Pinzón 2010).
Existen múltiples algoritmos diseñados para dar solución a este problema junto a sus variantes en cuanto a complejidad. En el presente trabajo se concatenan las cadenas y se comparan, usando este tipo de algoritmo definiéndolo en el contexto de la investigación analizada se busca que la cadena común más larga tenga la misma longitud que las cadenas comparadas.

1.3.4. Conclusiones parciales

Para darle solución al problema tratado se emplea un híbrido de lo estudiado, teniendo en cuenta que se buscan inconsistencias, o sea, elementos que deberían ser los mismos y no lo son. Por tanto, se trabaja con elementos para la búsqueda de duplicados pero se analizan los resultados de forma inversa, para el caso de los algoritmos de distancia como los mencionados. Para encontrar una inconsistencia y mostrar la distancia entre cada caracter o token debe estar en su umbral mayor.

1.4. Ingeniería de requisitos

Los requerimientos para un sistema son la descripción de los servicios proporcionados por este y sus restricciones operativas. Estos requerimientos reflejan las necesidades de los clientes de un sistema que ayude a resolver algún problema como el control de un dispositivo, hacer un pedido o encontrar información. El proceso de descubrir, analizar, documentar y verificar estos servicios y restricciones se denomina **ingeniería de requisitos** (Sommerville 2005).

El término **requerimiento** en algunos casos, es simplemente una declaración abstracta de alto nivel de un servicio que debe proporcionar el sistema o una restricción de este. En el otro extremo, es una definición detallada y formal de una función del sistema.

Los requerimientos de sistemas de software se clasifican en funcionales y no funcionales. Los r**equerimientos funcionales** son declaraciones de los servicios que debe proporcionar el sistema, de la manera en que este debe reaccionar a entradas particulares y de cómo se debe comportar en situaciones particulares. En algunos casos, los requerimientos funcionales de los sistemas también

pueden declarar explícitamente lo que el sistema no debe hacer. Los **requerimientos no funcionales** son restricciones de los servicios o funciones ofrecidos por el sistema, incluyen restricciones de tiempo sobre el proceso de desarrollo y estándares. Los requerimientos no funcionales se aplican al sistema en su totalidad y apenas se aplican a características o servicios individuales del sistema (Sommerville 2005).

Los requerimientos del sistema para la realización de la herramienta informática que identifica y repara inconsistencias de datos de BD PostgreSQL fueron entregados por el cliente, el Ing. Juniel Tamayo Hernández. El mismo los obtuvo realizando la técnica de obtención de requisitos **minería de datos**.

1.4.1. Técnicas de minería de datos

La minería de datos ha dado lugar a una paulatina sustitución del análisis de datos dirigido a la verificación por un enfoque de análisis de datos dirigido al descubrimiento del conocimiento. La principal diferencia entre ambos se encuentra en que en el último se descubre información sin necesidad de formular previamente una hipótesis. La aplicación automatizada de algoritmos de minería de datos permite detectar fácilmente patrones en los datos, razón por la cual esta técnica es mucho más eficiente que el análisis dirigido a la verificación cuando se intenta explorar datos procedentes de repositorios de gran tamaño y complejidad elevada.

Los algoritmos de minería de datos se clasifican en dos grandes categorías: supervisados o predictivos y no supervisados o de descubrimiento del conocimiento. Los **algoritmos supervisados o predictivos** predicen el valor de un atributo (etiqueta) de un conjunto de datos, conocidos otros atributos (atributos descriptivos). A partir de datos cuya etiqueta se conoce se induce una relación entre dicha etiqueta y otra serie de atributos. Esas relaciones sirven para realizar la predicción en datos cuya etiqueta es desconocida. Esta forma de trabajar se conoce como aprendizaje supervisado, y se desarrolla en dos fases: entrenamiento[7] y prueba[8].

Cuando una aplicación no es lo suficientemente madura no tiene el potencial necesario para una solución predictiva, en ese caso hay que recurrir a los **métodos no supervisados o de descubrimiento** del conocimiento que descubren patrones y tendencias en los datos actuales (no utilizan datos históricos). El descubrimiento de esa información sirve para llevar a cabo acciones y obtener un beneficio (científico o de negocio) de ellas (García, Quintales, Peñalvo, Martín 2010).

1.5. Patrones de diseño orientados a objetos (PDOO)

Según Christopher Alexander, "cada patrón describe un problema que ocurre una y otra vez en nuestro medio, y luego describe el núcleo de la solución a ese problema, de tal manera que se puede utilizar esta solución un millón de veces, sin tener que hacer de la misma manera dos veces" (Alexander, Ishikawa, Silverstein 1977), es aplicable a los PDOO.

[7] **Entrenamiento:** primera fase del algoritmo predictivo donde se realiza la construcción de un modelo usando un subconjunto de datos con etiqueta conocida.
[8] **Prueba:** segunda fase del algoritmo predictivo donde se prueba del modelo sobre el resto de los datos.

En general, un patrón tiene cuatro elementos esenciales: el **nombre del patrón** que es un identificador utilizado para describir un problema de diseño, sus soluciones y sus consecuencias en una o dos palabras. Permite diseñar a un nivel más alto de abstracción. El **problema**, en él se explica el problema y su contexto, a veces se le incluirá una lista de condiciones que deben cumplirse antes de que tenga sentido para aplicar el patrón. La **solución** en la cual se describen los elementos que componen el diseño, sus relaciones, responsabilidades y colaboraciones. Es como una plantilla es aplicada en situaciones diferentes. Las **consecuencias** que son los resultados y las compensaciones de la aplicación del patrón, son fundamentales para evaluar alternativas de diseño y para la comprensión de los costos y beneficios de aplicar el patrón (Gamma, Helm, Johnson, Vlissides 1995).

1.6. Metodologías de desarrollo

Las metodologías son sistemas de principios y normas generales de organización y estructuración teórico-práctica de actividades (Carrillo 2006). Las metodologías de desarrollo de software surgen ante la necesidad de utilizar una serie de procedimientos, técnicas, herramientas y soporte documental a la hora de desarrollar un producto de software. Dichas metodologías pretenden guiar a los desarrolladores al crear un nuevo software, pero los requisitos de un software a otro son tan variados y cambiantes, que ha dado lugar a que exista una gran variedad de metodologías para la creación del software. Se podrían clasificar en dos grandes grupos: las metodologías tradicionales y las metodologías ágiles (Grupo de Ingeniería del Software y Sistemas de Información (ISSI), Torres, Emilio A. Sánchez López 2003).

La siguiente tabla establece una comparación entre los tipos de metodologías definidos:

Metodologías Ágiles	***Metodologías Tradicionales***
Basadas en heurísticas provenientes de prácticas de producción de código.	Basadas en normas provenientes de estándares seguidos por el entorno de desarrollo.
Especialmente preparados para cambios durante el proyecto.	Cierta resistencia a cambios.
Impuestas internamente (por el equipo).	Impuestas externamente.
Proceso menos controlado, con pocos principios.	Proceso mucho más controlado, con numerosas políticas/normas.
No existe contrato tradicional o al menos es bastante flexible.	Existe un contrato prefijado
El cliente es parte del equipo de desarrollo.	El cliente interactúa con el equipo de desarrollo mediante reuniones.
Grupos pequeños (<10 integrantes) y trabajando en el mismo sitio.	Grupos grandes y posiblemente distribuidos.
Pocos Artefactos.	Más artefactos.
Pocos Roles.	Más roles.
Menos énfasis en la arquitectura del software.	La arquitectura del software es esencial y se expresa mediante modelos.

Tabla 2. Diferencias entre metodologías ágiles y no ágiles

(Canós, Letelier, Penadés 2009)

1.6.1. Metodologías tradicionales

Las metodologías tradicionales imponen una disciplina de trabajo sobre el proceso de desarrollo del software, con el objetivo de conseguir un software más eficiente y predecible. Para ello, se hace un especial hincapié en la planificación total de todo el trabajo a realizar y una vez que está todo detallado, comienza el ciclo de desarrollo del producto software (Cáceres, Marcos, Kybele 2002).

A continuación se describe una de las metodologías tradicionales.

RUP:

La metodología RUP, llamada así por sus siglas en inglés Rational Unified Process, divide en 4 fases el desarrollo del software. La fase inicio donde se determina la visión del proyecto. La fase elaboración donde se determina la arquitectura óptima. La fase construcción donde se obtiene la capacidad operacional inicial. Y la fase transición donde se obtiene la versión estable (release) del proyecto. El ciclo de vida que se desarrolla por cada iteración, es llevada bajo dos disciplinas: la **disciplina de desarrollo** donde se entienden las necesidades del negocio, se trasladan las necesidades del negocio a un sistema automatizado, los requerimientos dentro de la arquitectura de software, se crea un software que se ajuste a la arquitectura y que tenga el comportamiento deseado y se asegura que el comportamiento requerido es el correcto y que todo lo solicitado está presente. La **disciplina de soporte** donde se guardan todas las versiones del proyecto, se administra horarios, recursos y el ambiente de desarrollo y se hace todo lo necesario para la salida del proyecto.

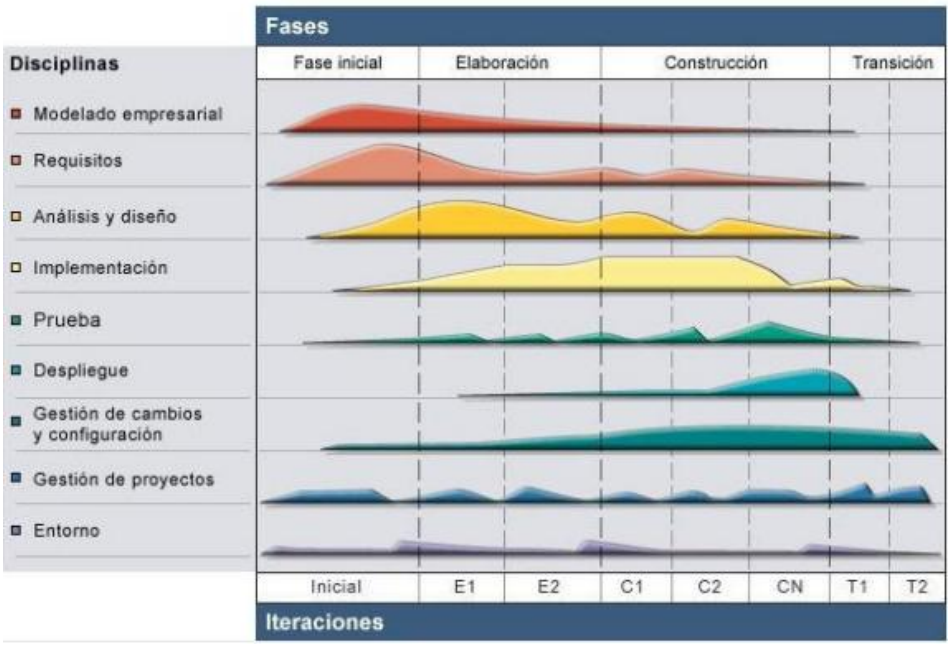


Fig. 1. Fases e iteraciones de la metodología RUP

Los elementos del RUP son las actividades, los trabajadores y los artefactos. Una particularidad de esta metodología es que en cada ciclo de iteración se hace exigente el uso de artefactos, siendo por este motivo una de las metodologías más importantes para alcanzar un grado de certificación en el desarrollo del software (Sánchez 2004).

Las metodologías tradicionales no se adaptan adecuadamente a los cambios, por lo que su uso no es conveniente cuando se trabaja en un entorno donde los requisitos no puedan predecirse. La poca flexibilidad a los cambios se evidencia debido a la existencia de un contrato prefijado.

1.6.2. Metodologías ágiles

Las metodologías ágiles son adaptativas, proponen procesos que se adaptan y progresan con el cambio, llegando incluso hasta el punto de cambiar ellos mismos. Las metodologías ágiles tienen principios o características que diferencian un proceso ágil de uno tradicional, algunos de los principios más importantes son:

- La prioridad es satisfacer al cliente mediante tempranas y continuas entregas de software que le aporte un valor.
- Dar la bienvenida a los cambios. Se capturan los cambios para que el cliente tenga una ventaja competitiva.
- Entregar frecuentemente software que funcione desde un par de semanas a un par de meses, con el menor intervalo de tiempo posible entre entregas (Roger S. Pressman 2005).

Existen diversas metodologías que se clasifican como metodologías ágiles. Y aunque entre ellas comparten muchas características tienen también diferencias significativas. A continuación se presentan algunas de las metodologías ágiles más representativas.

XP (eXtreme Programming):

Es una de las metodologías de desarrollo de software más exitosas en la actualidad utilizada para proyectos de corto plazo, pequeño equipo y cuyo plazo de entrega terminaba ayer. La metodología consiste en una programación rápida o extrema, cuya particularidad es tener como parte del equipo, al usuario final, pues es uno de los requisitos para llegar al éxito del proyecto (Sánchez 2004).

La programación extrema concede una gran importancia a las pruebas del software (testing). Aunque la mayoría de los procesos las tienen en cuenta, generalmente lo contemplan de una forma demasiado ligera y superficial. Sin embargo, XP lo toma como base para el desarrollo y cada programador que escribe código también escribe los casos de prueba. Estos forman parte del proceso continuo de generación de código y se integra el proceso continuamente con la generación del código, lo que garantiza una plataforma estable para el futuro desarrollo. Sobre dicha plataforma se genera un proceso de diseño evolutivo, que es la base del sistema y que se enriquece con cada iteración. Nunca se generan diseños futuros. El resultado es un proceso de diseño que combina adecuadamente la disciplina con la adaptabilidad (Cáceres, Marcos, Kybele 2002).

Fig. 2. Metodología XP (eXtreme Programming)

SCRUM

SCRUM es una Metodología Ágil para desarrollar el software en equipos. La filosofía en la que se basa es que el proceso de desarrollo software es muy complejo e imprevisible y que la única manera de manejarlo e intentar controlarlo es tratarlo como una caja negra y no como un proceso lineal definido en todas sus etapas. Las motivaciones más importantes en las que se enfoca esta idea son las que todos los ingenieros del software conocen: es muy difícil entender bien los requisitos desde las primeras fases del proyecto, los requisitos pueden cambiar durante el desarrollo y la evolución del sistema es imprevisible cuando se necesite cambiar o añadir tecnologías y herramientas en marcha. Su ciclo de vida no se trata como un flujo lineal en el que se puedan distinguir exactamente estas actividades; además no se precisa seguir un orden preciso en el proceso de desarrollo. El proyecto puede empezar con cualquier actividad y se puede pasar de una actividad a otra en cualquier momento maximizando la flexibilidad y la productividad del equipo. Hay que trabajar en equipos pequeños para reducir problemas de comunicación y coordinamiento y, a la vez, maximizar la cooperación; se precisa finalmente que todas las actividades y las entregas, tienen que ser flexibles (F. al proceso desarrollo SCRUM, Spada 2011).

AUP (Agile Unified Process)

Agile Unified Process es una versión simplificada de RUP, desarrollada por Scott Amber y consta de cuatro fases las cuales se expondrán seguidamente:

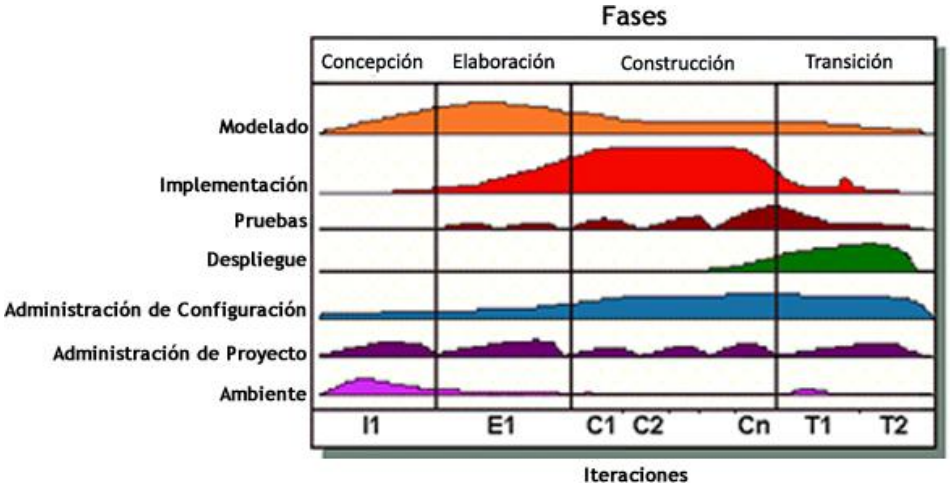


Fig. 3. Fases y flujos de AUP.

Inception (Concepción):

Suele ser la fase más corta del desarrollo. Los clientes deben participar activamente del modelado en alto nivel de los requerimientos ya que este determina el alcance inicial del proyecto. El objetivo es identificar una arquitectura viable, explorar la utilización del producto escribiendo CU, así como identificar los procesos de negocio mediante la realización de diagramas de flujo de datos. También identificar las principales entidades del negocio y sus relaciones, identificar las principales reglas de negocio y los principales requerimientos técnicos, comenzar la realización de un glosario que contenga los principales términos técnicos y del negocio.

Elaboración:

Durante esta fase deberían capturarse la mayoría de requisitos del sistema, establecer y validar la base de la arquitectura del sistema. Esta base se llevará a cabo a través de varias iteraciones, y servirá de punto de partida para la fase de construcción. La fase de elaboración termina, por tanto, al alcanzar el hito de la arquitectura del sistema.

Construcción:

Es la fase más larga del proyecto, y completa la implementación del sistema tomando como base la arquitectura obtenida durante la fase de elaboración. A partir de ella, las distintas funcionalidades son incluidas en distintas iteraciones, al final de cada una de las cuales se obtendrá una nueva versión ejecutable del producto.

Por tanto, esta fase concluye con el hito de obtención de una funcionalidad completa, que capacite al producto para funcionar en un entorno de producción. También en esta fase es necesario trabajar al lado del cliente para identificar sus necesidades paso a paso.

Transición:

En la fase final del proyecto se lleva a cabo el despliegue del producto en el entorno de los usuarios, lo que incluye la formación de estos. En esta fase se debe:

- Aplicar lluvia de ideas para intentar comprender la causa de defectos detectados.
- Finalizar la documentación general del sistema. El mejor momento para terminar la documentación general del sistema es en esta fase donde el alcance del sistema está verdaderamente estabilizado.
- Comprende pruebas del sistema, de los usuarios e instalación del sistema.
- Entre sus objetivos busca la aceptación de los stakeholders o (participantes) del negocio, de operaciones, soporte y de costos estimados.

Entregables:

Entre los diagramas de Lenguaje Unificado de Modelado (LUM o UML, por sus siglas en inglés, Unified Modeling Language) recomendados por el sitio web **ambysoft**[9] aplicables para AUP se tienen (Ambler 2011):

[9] **Ambysoft Inc.:** es un sitio web que comparte una gran cantidad de material gratuito sobre las prácticas de desarrollo de software y las mejores técnicas del proceso de software en general (Ambler 2011).

- Diagramas de CU
- Especificación de CU
- Diagrama de clases
- Diagramas de secuencia
- Interfaces graficas de usuario
- Diagrama de componentes
- Diagrama de despliegue

1.6.3. Conclusiones parciales

Las metodologías tradicionales son usadas por grupos grandes de trabajo, su uso generalmente implica un plan de trabajo más amplio. Teniendo en cuenta que se trata de un equipo de trabajo pequeño y que además se cuenta con poco tiempo para la implementación, luego del análisis realizado, se decidió utilizar una **metodología ágil**. A partir del estudio realizado sobre las metodologías ágiles se decidió utilizar **AUP** pues esta cuenta con las características necesarias para guiar el desarrollo de la herramienta propiciando el cumplimiento de su objetivo.

1.7. Lenguajes de modelado de software

Los lenguajes de modelado son un conjunto estandarizado de símbolos y sus relaciones que permiten expresar un Sistema de Información mediante un esquema teórico, el cual representará su diseño.

Como ejemplos destacados se puede hablar de la técnica de modelado de objetos (OMT[10]), de Rumbaugh, del lenguaje de modelado Booch, de Grady Booch y de UML, que influido inicialmente por OMT y Booch y tomando además conceptos de otros lenguajes/métodos de la orientación a objetos, unificaba bajo un mismo estándar un lenguaje válido, en principio, para expresar los diferentes aspectos de cualquier desarrollo orientado a objetos (OO) (Raúl Hernández 2012).

1.7.1. Lenguaje unificado de modelado (UML)

El UML se define como un lenguaje que permite especificar, visualizar y construir los artefactos de los sistemas de software. Es un sistema notacional (que, entre otras cosas, incluye el significado de sus notaciones) destinado a los sistemas de modelado que utilizan conceptos OO.

El UML es un estándar establecido de la industria para construir modelos OO. Nació en 1994 por iniciativa de Grady Booch y Jim Rumbaugh para combinar sus dos famosos métodos: el de Booch y la OMT. Más tarde se les unió Ivar Jacobson, creador del método ingeniería de software orientada a objetos (OOSE[11]). En respuesta a una petición de Asociación para fijar los estándares de la industria (OMG[12]) para definir un lenguaje y una notación estándar del lenguaje de construcción de modelos, en 1997 propusieron el UML como candidato (Larman 2004).

[10] **OMT:** Object Modeling Technique o la Técnica de Modelado de Objetos.
[11] **OOSE:** Object-Oriented Software Engineering, Ingeniería de Software Orientada a Objetos.
[12] **OMG:** Object Management Group, asociación para fijar los estándares de la industria.

El lenguaje unificado de modelado preescribe un conjunto de notaciones y diagramas estándar para modelar sistemas OO, y describe la semántica esencial de lo que estos diagramas y símbolos significan. UML se puede usar para modelar distintos tipos de sistemas: sistemas de software, hardware, y organizaciones del mundo real (Kobluk, Mariño, Valesani 2003).

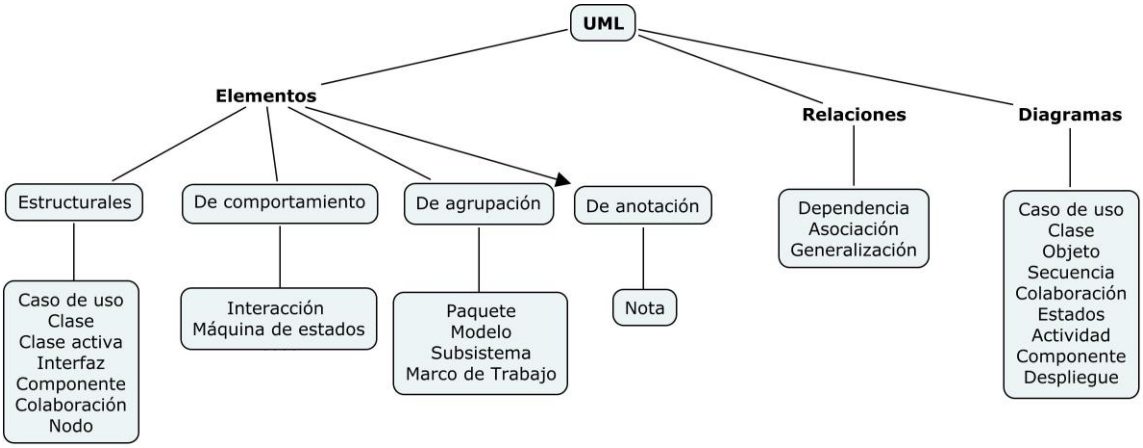


Fig. 4. Composición de UML

1.8. Estilos arquitectónicos

"La arquitectura es la organización fundamental de un sistema integrado en sus componentes, la relación entre ellos y el medio, y los principios que establecen su desarrollo y evolución."(Committee 2000)

Los estilos arquitectónicos son un conjunto de reglas de diseño que identifica las clases de componentes y conectores que se pueden utilizar para componer en sistema o subsistema, junto con las restricciones locales o globales de la forma en que la composición se lleva a cabo.

1.8.1. Estilo arquitectónico de llamada y retorno

El estilo arquitectónico que se utilizará es el de llamada y retorno debido a que esta familia de estilos enfatiza la capacidad de modificación y la escalabilidad. Son los estilos más generalizados en sistemas en gran escala. Los miembros de la familia son las arquitecturas de programa principal y subrutina, los sistemas basados en llamadas a procedimientos remotos, los sistemas OO y los sistemas jerárquicos en capas.

Modelo-Vista-Controlador (MVC):

El patrón MVC separa el modelado del dominio, la presentación y las acciones basadas en datos ingresados por el usuario en tres clases diferentes:

- **Modelo:** El modelo administra el comportamiento y los datos del dominio de aplicación, responde a requerimientos de información sobre su estado (usualmente formulados desde la vista) y responde a instrucciones de cambiar el estado (habitualmente desde el controlador).
- **Vista:** Maneja la visualización de la información.

- **Controlador:** Interpreta las acciones del ratón y el teclado, informando al modelo y/o a la vista para que cambien según resulte apropiado.

Tanto la vista como el controlador dependen del modelo, el cual no depende de las otras clases. Esta separación permite construir y probar el modelo independientemente de la representación visual.

Ventajas:

Soporte de vistas múltiples: Dado que la vista se halla separada del modelo y no hay dependencia directa del modelo con respecto a la vista, la interfaz de usuario puede mostrar múltiples vistas de los mismos datos simultáneamente. Por ejemplo, múltiples páginas de una aplicación Web pueden utilizar el mismo modelo de objetos, mostrado de maneras diferentes.

Adaptación al cambio: Los requerimientos de interfaz de usuario tienden a cambiar con mayor rapidez que las reglas de negocios. Los usuarios pueden preferir distintas opciones de representación, o requerir soporte para nuevos dispositivos como teléfonos celulares. Dado que el modelo no depende de las vistas, agregar nuevas opciones de presentación generalmente no afecta al modelo.

Desventajas:

Complejidad: Se profundiza la orientación a eventos del código de la interfaz de usuario, que puede llegar a ser difícil de depurar.

Costo de actualizaciones frecuentes: Desacoplar el modelo de la vista no significa que los desarrolladores del modelo puedan ignorar la naturaleza de las vistas. Si el modelo experimenta cambios frecuentes, por ejemplo, podrían desbordar las vistas con una lluvia de requerimientos de actualización (Reynoso 2004).

Arquitecturas en Capas:

Es una organización jerárquica tal que cada capa proporciona servicios a la capa inmediatamente superior y se sirve de las prestaciones que le brinda la inmediatamente inferior. Las capas suelen ser entidades complejas, compuestas de varios paquetes o subsistemas. En un estilo en capas, los conectores se definen mediante los protocolos que determinan las formas de la interacción.

Ventajas:

El estilo soporta un diseño basado en niveles de abstracción crecientes, lo cual a su vez permite a los implementadores la partición de un problema complejo en una secuencia de pasos incrementales. Admite optimizaciones y refinamientos. Proporciona amplia reutilización. Al igual que los tipos de datos abstractos, se pueden utilizar diferentes implementaciones o versiones de una misma capa en la medida que soporten las mismas interfaces de cara a las capas adyacentes. Esto conduce a la posibilidad de definir interfaces de capa estándar, a partir de las cuales se pueden construir extensiones o prestaciones específicas.

Desventajas:

Muchos problemas no admiten un buen mapeo en una estructura jerárquica. Incluso cuando un sistema se puede establecer lógicamente en capas, consideraciones de presentación pueden requerir acoplamientos específicos entre capas de alto y bajo nivel. A veces es también extremadamente difícil encontrar el nivel de abstracción correcto. Además, los cambios en las capas de bajo nivel tienden a filtrarse hacia las de alto nivel, en especial si se utiliza una modalidad relajada; también se admite que la arquitectura en capas ayuda a controlar y encapsular aplicaciones complejas, pero complica no siempre razonablemente las aplicaciones simples (Reynoso 2004).

Arquitecturas OO:

Los componentes del estilo se basan en principios OO: encapsulamiento, herencia y polimorfismo. Son asimismo las unidades de modelado, diseño e implementación, y los objetos y sus interacciones son el centro de las incumbencias en el diseño de la arquitectura y en la estructura de la aplicación.

Las interfaces están separadas de las implementaciones. En general la distribución de objetos es transparente, y en el estado de arte de la tecnología apenas importa si los objetos son locales o remotos. En cuanto a las restricciones, puede admitirse o no que una interfaz pueda ser implementada por múltiples clases. En cuanto a los componentes, los objetos interactúan a través de invocaciones de funciones y procedimientos.

Ventajas:

Entre las cualidades, la más básica concierne a que se puede modificar la implementación de un objeto sin afectar a sus clientes. Asimismo es posible descomponer problemas en colecciones de agentes en interacción. Además, un objeto es ante todo una entidad reutilizable en el entorno de desarrollo.

Desventajas:

Entre las limitaciones, el principal problema del estilo se manifiesta en el hecho de que para poder interactuar con otro objeto a través de una invocación de procedimiento, se debe conocer su identidad. Esta situación contrasta con el caso de estilos tubería-filtros, donde los filtros no necesitan poseer información sobre los otros filtros que constituyen el sistema. La consecuencia inmediata de esta característica es que cuando se modifica un objeto (por ejemplo, se cambia el nombre de un método, o el tipo de dato de algún argumento de invocación) se deben modificar también todos los objetos que lo invocan. También se presentan problemas de efectos colaterales en cascada: si A usa B, y C también lo usa, el efecto de C sobre B puede afectar a A (Reynoso 2004).

Arquitecturas basadas en componentes:

Se trata de una forma de desarrollo de software, o rama de la Ingeniería de software que basa su funcionamiento en la descomposición del diseño en componentes funcionales. Es así que uno de estos componentes tratados como subsistemas puede convertirse en un sistema mayor por sí solo. Este estilo de arquitectura contempla una definición de componentes lógicos que expongan interfaces de comunicación bien definidas. En esencia un componente es una pieza de código preelaborado que

encapsula alguna funcionalidad expuesta a través de interfaces estándar (WebCab Components Company [no date]). En resumen “Un componente es una unidad de composición de aplicaciones 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” (Szyperski 1998).

Ventajas:

Permite un desarrollo de software rápido debido a la reutilización de sus componentes, simplifica el soporte del sistema de acuerdo con el grado de acoplamiento entre los componentes pues se pueden hacer cambios, adicionar o eliminar, sin afectar el funcionamiento final del sistema.

Desventajas:

No se puede pensar en combinar un número muy grande de componentes, al no poder garantizar el correcto funcionamiento de cada componente, siendo necesario probarlos.

1.8.2. Conclusiones parciales

La arquitectura seleccionada fue la arquitectura en capas pues dentro de su funcionamiento contempla la reutilización de componentes, siendo este concepto utilizado en la implementación. También permite el mantenimiento y las mejoras a la solución, debido al bajo acoplamiento entre las capas, la alta cohesión de las capas y la habilidad de cambiar su implementación sin cambiar las interfaces. Tiene en cuenta otras soluciones, ya que se reusan las funcionalidades expuestas por las diferentes capas, especialmente si las capas de las interfaces son diseñadas con la reutilización en mente. Además, la capacidad de realizar pruebas se beneficia de tener interfaces bien definidas para cada capa, así como de la habilidad para cambiar a diferentes implementaciones de las interfaces (Trowbridge, Mancini, Quick, Hohpe, Newkirk, Lavigne 2003).

1.9. Lenguajes de programación y herramientas de desarrollo

Con el objetivo de lograr la portabilidad de la herramienta que se quiere desarrollar y la independencia de plataforma, se pretende desarrollar una herramienta de escritorio. Teniendo en cuenta además la experiencia del grupo de trabajo para el desarrollo de la aplicación. Es importante destacar que el uso de una herramienta de escritorio permite un amplio control sobre el contenido, compartir datos de negocio y proporcionar seguridad a la misma debido a que la máquina donde esté ejecutándose la aplicación puede tener mecanismos para su seguridad (Varela 2011).

1.9.1. Selección del lenguaje de programación

Se hace necesario utilizar un lenguaje que permita desarrollar la herramienta informática, de manera que sea multiplataforma, que permita satisfacer todas las funcionalidades requeridas, además de ser libre debido a las aristas que va tomando eventualmente el país y por consiguiente la UCI. Dada las

características necesarias para la selección del lenguaje de programación se analizan los más potenciales:

Python:

Python es un lenguaje de programación creado por Guido Van Rossum a principios de los años 90 cuyo nombre está inspirado en el grupo de cómicos ingleses “Monty Python”. Posee una sintaxis muy limpia y que favorece un código legible. Se trata de un lenguaje interpretado o de script, con tipado dinámico, fuertemente tipado, multiplataforma y OO.

- **Lenguaje interpretado o de script**

Un lenguaje interpretado o de script es aquel que se ejecuta utilizando un programa intermedio llamado intérprete, en lugar de compilar el código a lenguaje máquina que comprenda y ejecute directamente una computadora (lenguajes compilados). La ventaja de los lenguajes compilados es que su ejecución es más rápida. Sin embargo, los lenguajes interpretados son más flexibles y más portables.

Python tiene, no obstante, muchas de las características de los lenguajes compilados, por lo que se podría decir que es semiinterpretado. (Duque 2012)

Inconveniente:

Su principal inconveniente es la lentitud por tratarse de un lenguaje interpretado, lo que implica un bajo rendimiento.

Pre-procesador de hipertexto (Php):

PHP es un lenguaje interpretado de alto nivel empapado en páginas HTML y ejecutado en el servidor. Al nivel más básico, PHP puede procesar la información de formularios, generar páginas con contenidos dinámicos, o mandar y recibir cookies.

PHP también soporta otros servicios que usen protocolos como IMAP, SNMP, NNTP, POP3, HTTP y derivados. También se pueden abrir sockets de red directos e interactuar con otros protocolos.

PHP fue concebido en otoño de 1994 por Rasmus Lerdorf. Las primeras versiones no distribuidas al público fueron usadas en sus páginas web para mantener un control sobre quien consultaba su currículum. (Bakken, Aulbach, Schmid, Winstead, Lars Torben Wilson, Lerdorf, Suraski, Zmievski, Ahto 2001)

Inconveniente:

Teniendo en cuenta que la herramienta informática que se desea implementar es factible que sea una herramienta de escritorio, el principal inconveniente que presenta el lenguaje PHP es que es un lenguaje interpretado de alto nivel inmerso en páginas HTML y ejecutado en el servidor. Es necesario utilizar una biblioteca gráfica, a la que se conecta, para poder manejar componentes de una interfaz gráfica de

usuarios con el objetivo de realizar aplicaciones de escritorio, algo que no favorece el desarrollo en corto tiempo de la herramienta deseada.

Java:

El lenguaje Java fue diseñado e implementado por un grupo de trabajadores de la empresa Sun Microsystems[13]. El mismo se encontraba encabezado por James Gosling, considerado en todo el mundo como el padre de Java. Gosling ya era conocido por haber sido autor de utilitarios para ambientes Unix y Solaris, pero saltó a la fama con la creación del lenguaje Java (Ángel López 1997).

Java es un lenguaje de programación OO, independiente de arquitecturas hardware, SO y sistemas de ventanas. Utiliza el concepto de máquina virtual (MV), la MV se encarga de traducir este código para que cualquier ordenador pueda ejecutarlo. De esta manera, un código generado en Java puede correr en cualquier plataforma, en donde se haya portado la misma. Siendo la máquina virtual de Java (MVJ) un programa nativo, es decir, ejecutable en una plataforma específica, capaz de interpretar y ejecutar instrucciones expresadas en un código binario especial, el cual es generado por el compilador del lenguaje Java (León, Rodríguez, Veitia 2011).

Como tal, es de uso general, y su sintaxis y semántica es lo bastante completa para poder abarcar programas de todo tipo. Otro aspecto a destacar, es que se trata de un lenguaje OO. Java es un lenguaje que es ejecutado en múltiples plataformas, es uno de los escasos lenguajes cuyos programas pueden ser transportados de SO, computadora o entorno, sin necesidad de cambiar el código. Ha sido concebido, desde sus orígenes, como un lenguaje capaz de producir código totalmente transportable. Dentro de sus principales características se encuentran su simplicidad, ya que es un código similar a C/C++ pero eliminando algunos elementos conflictivos (punteros, herencia múltiple, etc.), su portabilidad, ya que posee una representación y comportamiento único, es un sistema abstracto de ventanas que presenta el mismo comportamiento en distintos entornos. Es multiplataforma, un lenguaje robusto, pues posee una fuerte comprobación de tipos y de límites de los arreglos, tiene ausencia de punteros y manejo de errores (excepciones). Es seguro, debido a que no se puede acceder a memoria directamente mediante punteros y tiene un gestor de seguridad (Administrador de Seguridad) para el código binario. Es OO puros ya que obliga a trabajar en términos que facilitan la reutilización. Es multihilo, da soporte a la programación de procesos concurrentes. Es dinámico ya que permite la carga dinámica de clases y la búsqueda de nuevos objetos o clases en entornos distribuidos. Es un lenguaje interpretado. Y presenta un ambiente de ejecución de Java compuesto por la MVJ y la Java API: lenguaje básico + biblioteca estándar de clases (Javier Parapar López 2008).

[13] **Sun Microsystems** fue una empresa informática que se dedicaba a vender computadoras, componentes informáticos, software y servicios informáticos. Fue adquirida en el año 2009 por Oracle Corporation (Vázquez 2009).

Ventajas:

De los lenguajes analizados el equipo de desarrollo está más familiarizado con Java, cuenta con una amplia documentación y ayuda, además de un potente IDE de desarrollo. El uso del concepto de MV posibilita la implementación de herramientas multiplataforma, cuenta además con bibliotecas gráficas que facilitan la implementación de interfaces gráficas de usuario.

1.9.2. MVJ

La MVJ es una de las piezas fundamentales de la plataforma Java. Básicamente se sitúa en un nivel superior al Hardware del sistema sobre el que se pretende ejecutar la aplicación, y este actúa como un puente que entiende tanto el código binario como el sistema sobre el que se pretende ejecutar. Así, cuando se escribe una aplicación Java, se hace pensando que será ejecutada en una MVJ en concreto, siendo esta la que en última instancia convierte de código binario a código nativo del dispositivo final. La gran ventaja de la MVJ es aportar portabilidad al lenguaje, de manera que desde Sun Microsystems se han creado para diferentes arquitecturas, así un programa .class escrito en Windows puede ser interpretado en un entorno Linux. Tan solo es necesario disponer de dicha MVJ para dichos entornos. De ahí el famoso axioma que sigue a Java, "escríbelo una vez, ejecútalo en cualquier parte" (Domínguez, J. G.R Hernández, Sotolongo, Acosta, Díaz 2010).

Conclusiones parciales:

Para el desarrollo de la aplicación se va a utilizar JDK 1.7, teniendo en cuenta que la herramienta a desarrollar será libre, para su posterior uso se utilizara en las máquinas cliente OpenJDK 7 la versión libre del kit de desarrollo Java.

1.9.3. Lenguaje a nivel de BD a tener en cuenta

Como la herramienta va a interactuar directamente con la estructura y los datos de las BD, es importante analizar el lenguaje “Lenguaje Procedural/Lenguaje de consulta estructurado PostgreSQL” (PL/PgSQL), el mismo se describe a continuación:

PL/PgSQL:

PL/pgSQL es un lenguaje procedural cargable para el sistema de BD PostgreSQL. Los objetivos del diseño de PL/pgSQL fueron crear un lenguaje procedural cargable que: se pudiera usar para crear funciones y procedimientos disparados por eventos, añadiera estructuras de control al lenguaje SQL, realizara cálculos complejos, heredara todos los tipos definidos por el usuario, las funciones y los operadores, pudiera ser definido para ser fiable para el servidor y fuera fácil de usar.

El gestor de llamadas PL/pgSQL analiza el texto de las funciones y produce un árbol de instrucciones binarias interno la primera vez que la función es invocada por una aplicación. El código binario producido es identificado por el manejador de llamadas mediante el identificador de la función. Esto asegura que el cambio de una función por parte de una secuencia DROP/CREATE tendrá efecto sin tener que establecer una nueva conexión con la BD (Wieck 2002).

Se decide utilizar el lenguaje PL/PgSQL para el trabajo con los datos en las BD PostgreSQL debido a las características que presenta. A continuación se muestran algunas de ellas:

Para todas las expresiones y sentencias SQL usadas en la función, el intérprete de código binario de PL/pgSQL crea un plan de ejecución preparado usando los gestores de SPI, funciones SPI_prepare() y SPI_saveplan(). Esto se hace la primera vez que las sentencias individuales se procesan en la función PL/pgSQL. Así, una función con código condicional que contenga varias sentencias que puedan ser ejecutadas, solo preparará y almacenará las opciones que realmente se usarán durante el ámbito de la conexión con la BD.

El lenguaje PL/pgSQL no es sensible a las mayúsculas. Todas las palabras clave e identificadores pueden usarse en cualquier mezcla de mayúsculas y minúsculas.PL/pgSQL es un lenguaje orientado a bloques. Puede haber cualquier número de sub-bloques en la sección de sentencia de un bloque. Los sub-bloques pueden usarse para ocultar variables a otros bloques de sentencias. Las variables declaradas en la sección de declaraciones se inicializan a su valor por defecto cada vez que se inicia el bloque, no cada vez que se realiza la llamada a la función (Wieck 2002).

1.9.4. Selección de las herramientas a utilizar

Basándose en la metodología seleccionada, en los lenguajes, y en las tareas y artefactos se seleccionan las siguientes herramientas a utilizar:

Visual Paradigm para UML (VP):

VP para UML es una herramienta profesional que soporta el ciclo de vida completo del desarrollo de software. El software de modelado UML ayuda a una rápida construcción de aplicaciones de calidad, mejores y a un menor coste. Permite modelar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación. Se integra con varios ambientes de desarrollo integrados (IDE[14]) lo cual posibilita pasar del código al modelado y viceversa. Disponible en múltiples lenguajes y plataformas: Microsoft Windows (98, 2000, XP, o Vista) Linux o Java. Esta herramienta posee unas características gráficas muy cómodas que facilitan la realización de los diagramas de modelado que sigue el estándar de UML(Estrada 2011) y los utilizados para el modelado de la aplicación: los diagramas de clases del diseño, el diagrama de casos de uso del sistema, los diagramas de secuencia, los diagramas de componentes, el diagrama de despliegue y el modelo de dominio.

NetBeans 7.1:

NetBeans nació como un proyecto estudiantil en la República Checa en 1996. Su nombre original era Xelfi y fue el primer IDE para Java, escrito en Java. Es un IDE desarrollado por Sun Microsystems, de código abierto y multiplataforma. Permite diseñar aplicaciones de forma fácil con solo arrastrar objetos a la interfaz de un formulario. Es una plataforma pensada para escribir, compilar, depurar y ejecutar

[14] **IDE:** integrated development environment o entornos de desarrollo integrados.

programas. No solo permite el desarrollo de aplicaciones de escritorio, también permite el desarrollo de aplicaciones para la web y para dispositivos portátiles.

La programación en este IDE se realiza a través de módulos. Las aplicaciones construidas a partir de módulos pueden ser extendidas ya que permiten ser desarrolladas independientemente por otros desarrolladores de software, de ahí que sea una aplicación flexible/extensible.

NetBeans permite desarrollar aplicaciones en lenguajes como Java, C/C++, PHP, Python, entre otros. Es un IDE multilenguaje completo y modular que le brinda facilidades al programador para el desarrollo intuitivo, a la vez que posee una gran cantidad de módulos o plugins que sirven de ayuda al implementador para hacer su trabajo más sencillo y eficiente (Estrada 2011).

NetBeans IDE 7.1 introduce soporte para JavaFX 2.0, permitiendo la completa compilación y depuración. También proporciona importantes mejoras de Swing GUI Builder, soporte de CSS3, y herramientas para la depuración visual de Swing y las interfaces de usuario JavaFX. Otros puntos destacados incluyen las nuevas características de depuración de PHP, además está disponible en inglés, portugués brasileño, japonés, ruso y chino simplificado (Oracle Corporation and/or its affiliates 2012).

1.10. Métricas de calidad del software

Una métrica es una medida cuantitativa del grado en que un sistema, componente o proceso posee un atributo dado (IEEE 1990).

Según Pressman, las métricas del software proporcionan una manera cuantitativa de valorar la calidad de los atributos internos del producto, permitiendo por tanto al ingeniero valorar la calidad antes de construir el producto. Las métricas proporcionan la visión interna necesaria para crear modelos efectivos de análisis y diseño, un código sólido y pruebas minuciosas (R. S Pressman 2002).

Para medir el diseño existen numerosas métricas pero entre las más referenciadas se encuentran la familia de métricas de diseño OO propuesta por Chidamber y Kemerer.

Chidamber y Kemerer establecen 6 métricas basadas en clases para medir cinco atributos básicos en el diseño OO: acoplamiento, complejidad de una clase, reutilización, cohesión y herencia.

- ✓ Métodos ponderados por clase (Weighted Methods per Class –WMC)
- ✓ Profundidad del árbol de herencia (Depth of Inheritance Tree -DIT)
- ✓ Número de hijos (Number of children-NOC)
- ✓ Acoplamiento entre objetos (Coupling Between Object classes-CBO)
- ✓ Respuesta de una clase (Response for a class-RFC)
- ✓ Falta de cohesión en los métodos (Lack of cohesion in methods-LCOM)(Chidamber, Kemerer 1994)

A continuación se describirán las métricas utilizadas

1.10.1. Profundidad del árbol de herencia (Depth of Inheritancetree -DIT)

Es la distancia desde una clase, a la clase raíz del árbol de herencia. Si la clase se encuentra en situación de herencia múltiple, el DIT será la longitud máxima hasta la raíz.

Chidamber y Kemerer (Chidamber, Kemerer 1994) proponen esta métrica como medida de la complejidad de una clase, complejidad del diseño y el potencial de rehúso. Esto se debe a que las clases que estén más profundas en el árbol de herencia heredarán más métodos. Por lo tanto mientras mayor sea el valor de DIT las clases serán más complejas de desarrollar y de mantener, la jerarquía será más profunda haciendo el diseño más trabajoso, la probabilidad de detección de fallos será mayor. Por otra parte indica que se reutilizan muchos métodos. Es deseado obtener un número bajo de DIT.

1.10.2. Acoplamiento entre objetos (Coupling between object classes-CBO)

El acoplamiento es la dependencia de una clase con varias clases del sistema. Existe dependencia cuando la clase utiliza métodos o atributos de las otras clases (Chidamber, Kemerer 1994).

El CBO de una clase es el número de clases a las que ella está relacionada, sin tener en cuenta las relaciones por herencia, y mide la complejidad de la misma. El CBO alto no es deseado porque puede ser perjudicial para el diseño y mide la posible reutilización (mientras más independiente es una clase más fácil es de reutilizar). También un alto valor del mismo reduce el encapsulamiento y da medida de la complejidad de pruebas.

1.11. Conclusiones del capítulo

Luego del estudio realizado de las características del proceso de resolución de inconsistencias de estructura y de datos, del objeto de investigación y de las tendencias actuales de desarrollo de software, díganse metodologías de desarrollo, lenguajes de modelado, lenguajes de programación, etc. Se puede concluir lo siguiente:

- ✓ Existen múltiples herramientas para la detección y corrección de las inconsistencias entre BD. Estas herramientas son en su mayoría propietarias y además no solucionan todas las inconsistencias surgidas, al no contemplar algunas funcionalidades, como la comparación de datos. Teniendo en cuenta estos aspectos, y la variedad de inconsistencias que ocurren entre BD PostgreSQL, se decide implementar una herramienta que agrupe las funcionalidades necesarias para resolver las inconsistencias analizadas. La herramienta a desarrollar “Herramienta Informática para la Solución de Inconsistencias” (HIPSI), tiene como principal objetivo solucionar las inconsistencias analizadas (de datos, de estructura y de programación) que surgen a partir de las anomalías y su desarrollo será bajo licencia libre.
- ✓ Dentro de las metodologías se decide utilizar la metodología ágil AUP, debido a que posee características que tributan a las necesidades del desarrollo del software, es decir entregables suficientes para hacer entendible el sistema para posteriores implementaciones sobre el mismo. Se analizaron varios lenguajes de programación teniendo en cuenta la necesidad existente de

desarrollar una herramienta libre. Analizando la necesidad de lograr la portabilidad de la misma y la independencia de la plataforma, además de optimizar su rendimiento y que sea una herramienta informática de escritorio, se tomó como lenguaje para el desarrollo a Java. Este lenguaje de programación es portable, multiplataforma, robusto, seguro, orientado a objetos puros, multihilo y dinámico. Como IDE para la programación se seleccionó NetBeans 7.1 pues cuenta con bibliotecas gráficas muy potentes para el trabajo con interfaces gráficas de usuario, posee además muy buenos mecanismos de auto completamiento y corrección que facilita la implementación permitiendo que la misma se haga de forma rápida y eficiente. Es además una herramienta libre y sin restricciones de uso. Teniendo en cuenta la metodología seleccionada, las tareas y artefactos se trabajarán con la herramienta Visual Paradigm 8.0 para el modelado de la herramienta informática, ya que VP tiene una versión libre cuando su uso es no comercial. El lenguaje a nivel de BD que se utilizará es el PL/PgSQL para realizar el trabajo con las BD PostgreSQL.

Capítulo 2: Propuesta de solución

En el actual capítulo se formaliza la propuesta de solución del problema trazado, guiado por la metodología ágil de desarrollo de software AUP. Se realiza una descripción del negocio para el entendimiento del mismo, se presentan los requisitos funcionales y no funcionales del sistema que ayudan al modelado del sistema y que permite definir la arquitectura y comprender el negocio.

2.1. Modelo de dominio

Un modelo de dominio es una representación visual de clases conceptuales o de objetos reales en un dominio de interés (Calvopiña Morillo, Velasco Pacha 2012). Es un modelo conceptual, una herramienta para la comunicación que no necesariamente está bien o mal, sino que va a ser de mayor o menor utilidad (Larman 2003).

A continuación se muestra el modelo de dominio definido.

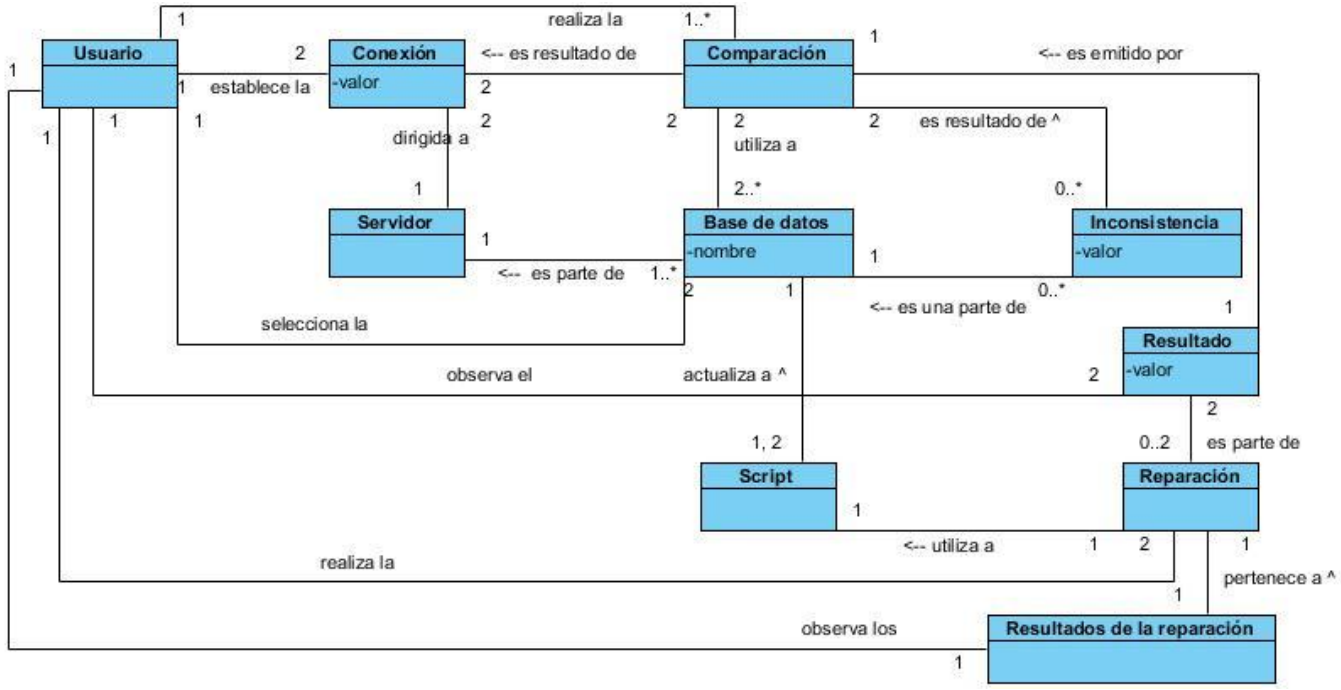


Fig. 5. Modelo de dominio

2.2. Requisitos del sistema

La captura de requisitos es un paso fundamental para saber exactamente lo que debe hacer y las cualidades o propiedades que debe tener el sistema. Los requisitos funcionales y no funcionales identificados para la realización de la herramienta informática son los siguientes:

2.2.1. Requisitos funcionales

RF_MC.001. Configurar Conexiones.

RF_MC.002. Validar Conexiones.

RF_MR.001. Configurar parámetros de comparación de datos.

RF_MR.002. Ejecutar comparación de datos.

RF_MR.003. Seleccionar datos correctos.

RF_MR.004. Generar scripts de reparación de datos.

RF_MR.005. Ejecutar scripts de reparación de datos.

RF_MR.006. Configurar parámetros de comparación de estructura.

RF_MR.007. Ejecutar comparación de estructura.

RF_MR.008. Seleccionar los elementos estructurales correctos.

RF_MR.009. Generar scripts de reparación de elementos estructurales.

RF_MR.0010. Ejecutar scripts de reparación elementos estructurales.

2.2.2. Requisitos no funcionales

- **Usabilidad:**

RNF_1. Cumplir con las pautas de diseño de las interfaces.

RNF_2. Agrupar vínculos y botones por grupos funcionales.

RNF_3. Mostrar los mensajes, títulos y demás textos que aparezcan en la interfaz del sistema en idioma español.

RNF_4. Utilizar campos de selección en la interfaz en los casos que sea posible.

RNF_5. Usar combinaciones de teclas para agilizar la interacción del usuario con el sistema.

- **Fiabilidad:**

RNF_6. Asegurar la disponibilidad del sistema.

- **Eficiencia:**

RNF_7. Responder en tiempos de hasta 5 segundos las peticiones que se realicen en el sistema.

- **Restricciones de diseño:**

RNF_8. Desarrollar el sistema para un entorno de escritorio.

RNF_9. Desarrollar el flujo del sistema en forma de asistente (Wizzard).

- **Requisitos para la documentación de usuarios y ayuda del sistema:**

RNF_10. Confeccionar manual de usuario.

- **Requisitos de software:**

RNF_11. Instalar en las estaciones de trabajo el software necesario para el correcto funcionamiento del sistema.

Las configuraciones de software de las máquinas clientes deben contar al menos con los siguientes elementos:

- MVJ.

- **Requisitos de hardware:**

RNF_12. Proporcionar características mínimas de hardware a las estaciones de trabajo.

Las características técnicas mínimas de hardware deben ser las siguientes:

Procesador de doble núcleo a 2.0 GHz.

2 GB de RAM, DDR2 667 MHz.

160 GB de disco duro, SATA 5400 RPM.

Adaptador de red Ethernet 1 Gbps.

Los requisitos fueron conciliados y aprobados por los autores del trabajo de diploma y el cliente, principal interesado en el desarrollo de la herramienta que identifique y repare las inconsistencias de datos entre BD PostreSQL.

2.3. Arquitectura del sistema

La arquitectura que se seleccionó fue la arquitectura en capas, véase el epígrafe 1.8.2.

2.3.1. Arquitectura en 2 capas

La arquitectura está definida en 2 capas, estas son:

Presentación:

La capa de **presentación**, es la encargada de interactuar con el usuario de la aplicación mediante una interfaz de usuario. La interfaz de usuario estará contenida por varias vistas, por las cuales el usuario va a ir realizando las operaciones mediante pasos ordenados para arribar a la reparación de la base de datos que contenga inconsistencias.

Lógica de negocio:

La capa **lógica de negocio** es la responsable de realizar las tareas para las cuales se diseña el sistema. También es la encargada de gestionar el almacenamiento de los datos, mediante el SGBD PostgreSQL.

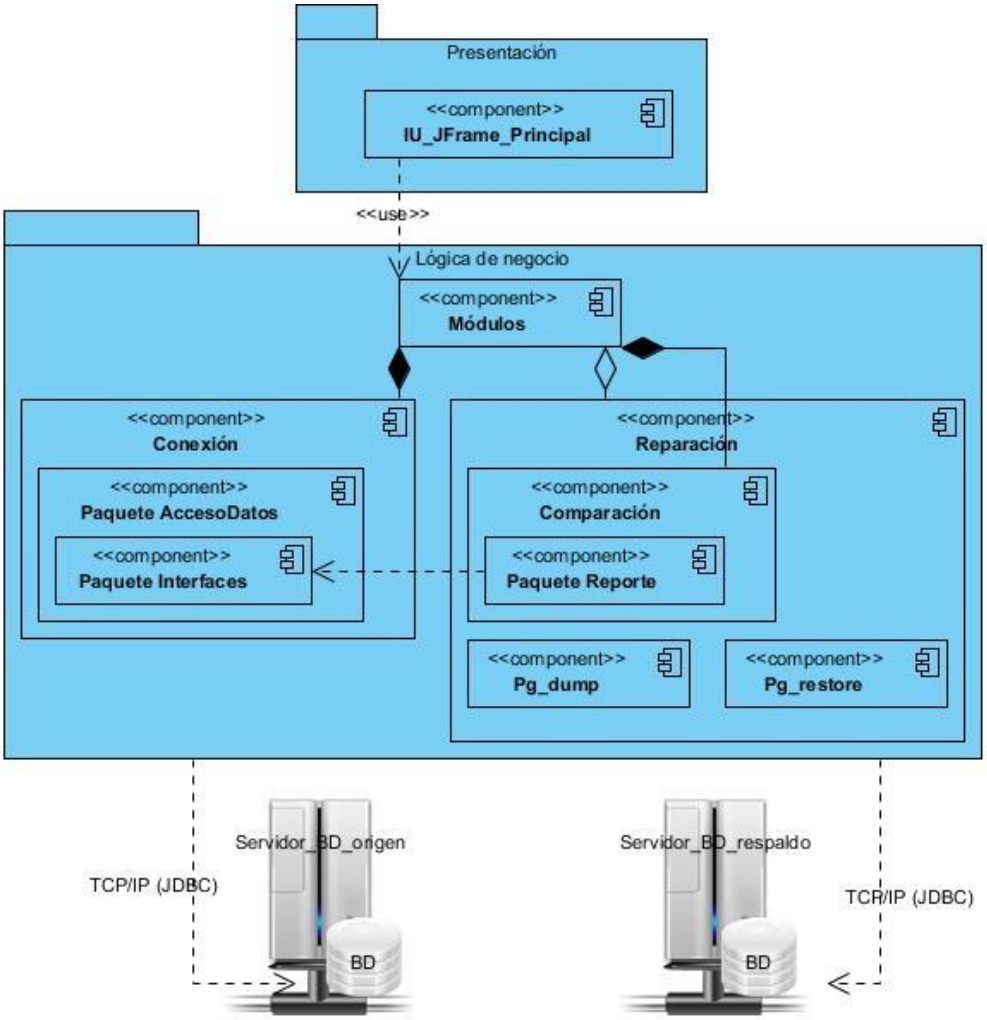


Fig. 6. Arquitectura en 2 capas

2.4. Definición del modelo de CU del sistema

2.4.1. Definición de los Actores del Sistema y Descripción

"Los actores son las diferentes personas (o dispositivos) que utilizan el sistema o producto dentro del contexto de la función y el comportamiento que se describirá. Los actores representan los papeles que juegan las personas (o dispositivos) conforme el sistema opera. Un actor es algún elemento que se comunica con el sistema o producto y que es externo al sistema en sí mismo. Cada actor tiene una o más metas cuando utiliza el sistema en sí mismo."(Roger S. Pressman 2005)

En este epígrafe se definirán el (los) Actores del Sistema involucrados en el Proceso de Identificación y Reparación de datos entre BD PostgreSQL.

Actor del Sistema	***Descripción***
Usuario	Usuario que accede al sistema para realizar las funcionalidades de este.

Tabla 3. Actores del sistema

2.4.2. Diagrama de casos de uso del sistema

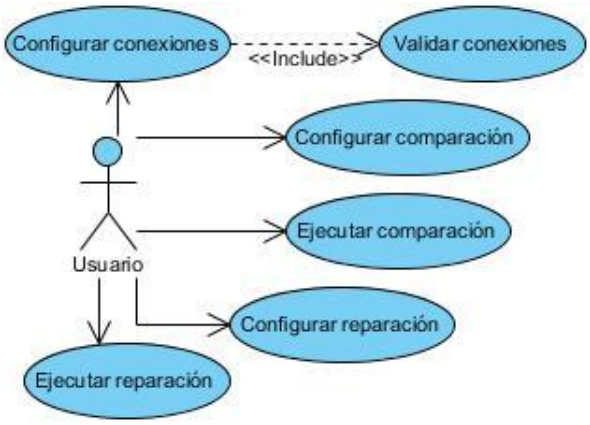


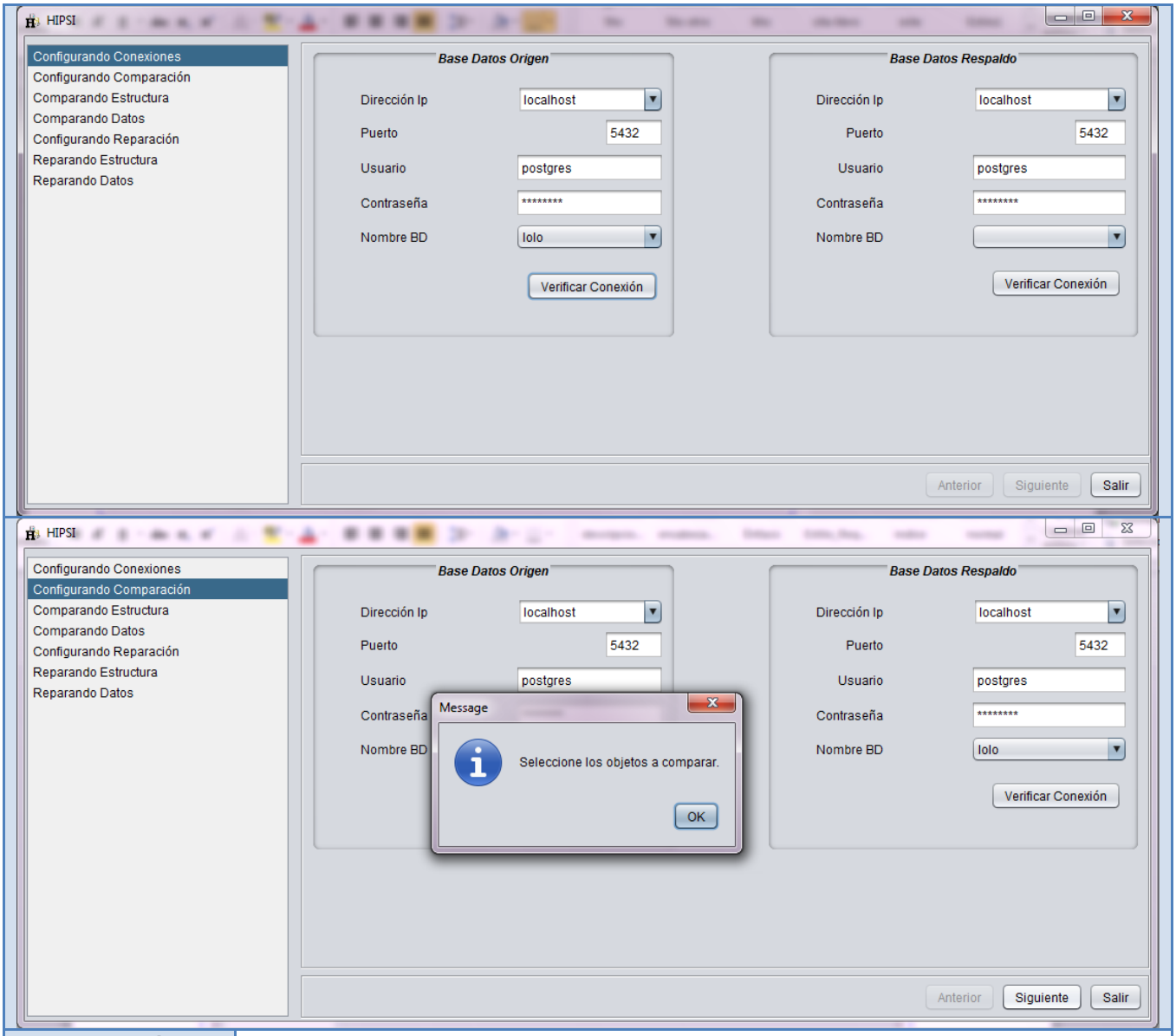
Fig. 7. Diagrama de casos de uso del sistema

2.4.3. Especificación de los CU del sistema del módulo Configuración

CU <Configurar conexiones>

Caso de Uso:	Configurar conexiones
Actores:	Usuario
Resumen:	El CU se inicia cuando se ejecuta la aplicación y se muestra la primera interfaz de usuario.
Precondiciones:	• El sistema debe estar ejecutándose correctamente. • En caso de ejecutar la aplicación fuera de los servidores de BD, debe existir conexión de red con estos.
Referencias	RF_MC.001, RF_MC.002, RF_MR.003, RF_MR.008, CU Validar conexiones, CU Configurar comparación.
CU asociados	CU incluido <Validar conexiones>
Prioridad	Crítico

Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
	1.1 El sistema muestra una interfaz para que el usuario introduzca los datos de las conexiones. • En caso de que el usuario desee salir de la herramienta (Ver sección “Salir”).
1.2 El usuario introduce los datos de la BD origen y presiona el botón “Verificar conexión” de la BD origen (Véase Punto de Inclusión Validar conexiones).	1.3 El sistema invoca el CU **Validar Conexiones**.
1.4 El usuario introduce los datos de la BD respaldo y presiona el botón “Verificar conexión” de la BD respaldo (Véase Punto de Inclusión Validar conexiones).	1.5 El sistema invoca el CU **Validar Conexiones**.
	1.6 El sistema habilita el botón Siguiente.
1.7 El usuario presiona el botón “Siguiente” de la interfaz.	1.8 El sistema muestra un mensaje informativo: “Seleccione los objetos a comparar.”.
1.9 El usuario presiona el botón “Aceptar” del mensaje.	1.10 El sistema invoca el CU **Configurar comparación**.
	1.11 El CU termina.
Sección “Salir”	
Flujo Normal de Eventos	**Flujo Normal de Eventos**
2.1 El usuario presiona el botón Salir.	2.2 La herramienta se cierra.
Prototipo de Interfaz	



Puntos de Inclusión:	• **Validar conexiones** El Sistema inicia el CU Validar conexiones.
Poscondiciones	• El sistema muestra la interfaz inicial del CU **Configurar comparación**.

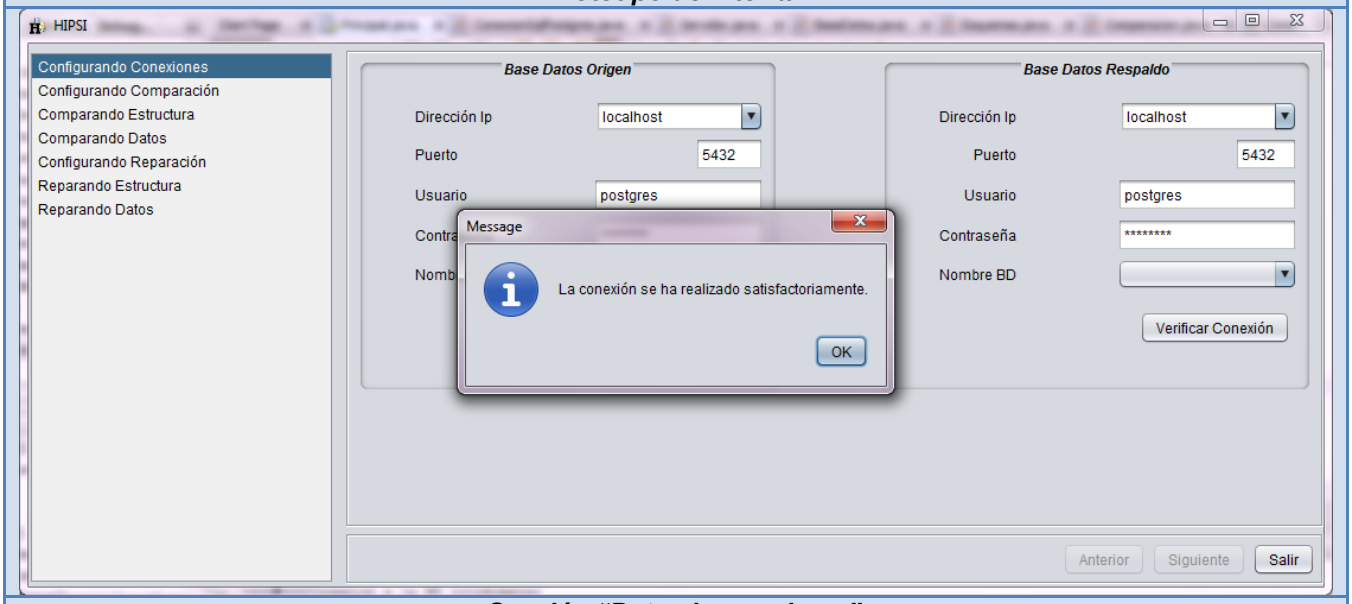
Tabla 4. Especificación del CU Configurar conexiones

CU <Validar conexiones>

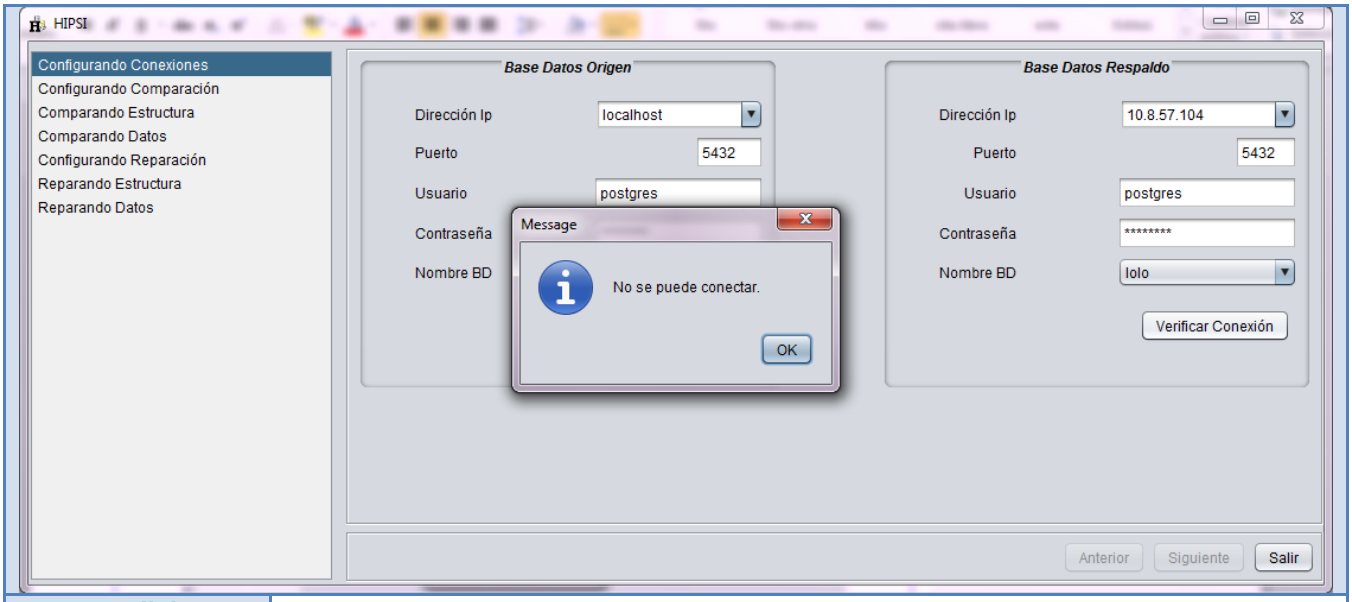
Caso de Uso:	Validar conexiones
Actores:	Usuario
Resumen:	El CU se inicia cuando el usuario presiona el botón “Verificar Conexión”
Precondiciones:	• El sistema debe estar ejecutándose correctamente. • En caso de ejecutar la aplicación fuera de los servidores de BD, debe existir conexión de red con estos.
Referencias	RF_MC.001, RF_MC.002, CU Configurar comparación.
Prioridad	Crítico

Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1.1 El usuario presiona el botón “Verificar conexión”.	1.2 El sistema chequea que todos los datos hayan sido introducidos. • Si se detecta algún dato obligatorio faltante (Ver sección Datos incompletos).

	1.3 El sistema intenta establecer conexión con las BD con los datos introducidos por el usuario. • En caso de que falle la conexión o se introduzcan mal los datos (ver sección “Conexión fallida”).
	1.4 El sistema muestra un mensaje indicando que se estableció la conexión a las BD de manera satisfactoria.
1.5 El usuario presiona el botón “Aceptar” del mensaje.	
	1.6 El CU termina.
Prototipo de Interfaz	



Sección “Datos incompletos”	
Acción del Actor	**Respuesta del Sistema**
	2.1 El sistema muestra un mensaje al usuario indicando que dejó en blanco o contiene datos incorrectos.
2.2 El actor selecciona la opción “Aceptar” del mensaje.	2.3 El sistema regresa al CU **Configurar conexiones** en el paso 3.1 de la sección conexión fallida.
Sección “Conexión fallida”	
Acción del Actor	**Respuesta del Sistema**
	3.1. El sistema muestra un mensaje al usuario indicando los errores detectados.
3.2. El actor selecciona la opción “Aceptar” del mensaje.	3.3. El sistema regresa al CU **Configurar conexiones** en el paso 1.1 del flujo normal de eventos.
Prototipo de Interfaz	



Poscondiciones	• El sistema muestra la interfaz inicial del CU **Configurar conexiones**.

Tabla 5. Especificación del CU incluido Validar conexiones

2.5. Diseño del sistema

El diseño del sistema es crucial para la resolución de un problema y la construcción de la solución del mismo, este debe tener en cuenta los requerimientos del sistema, no dejando pasar por alto ninguno de ellos, permitiendo resolver la organización del sistema.

2.5.1. Modelo de diseño

En el modelo de diseño se identifican clases, interfaces, las relaciones claves entre las mismas y subsistemas. Se definen las estructuras de datos necesarias para implementar el software, la relación entre los principales elementos estructurales del programa, la comunicación del software consigo mismo, con los sistemas que operan con él y con los operadores que lo emplean, así como la descripción procedimental de los componentes del software.

2.5.2. Diagrama de clases del diseño

El diagrama de clases es el diagrama principal para el análisis y diseño. Un diagrama de clases presenta las clases del sistema con sus relaciones estructurales y de herencia. La definición de clase incluye definiciones para atributos y operaciones (Olivares 2009).

En la siguiente imagen se muestra el diagrama de clases del CU Configurar conexiones con el CU incluido Validar conexiones.

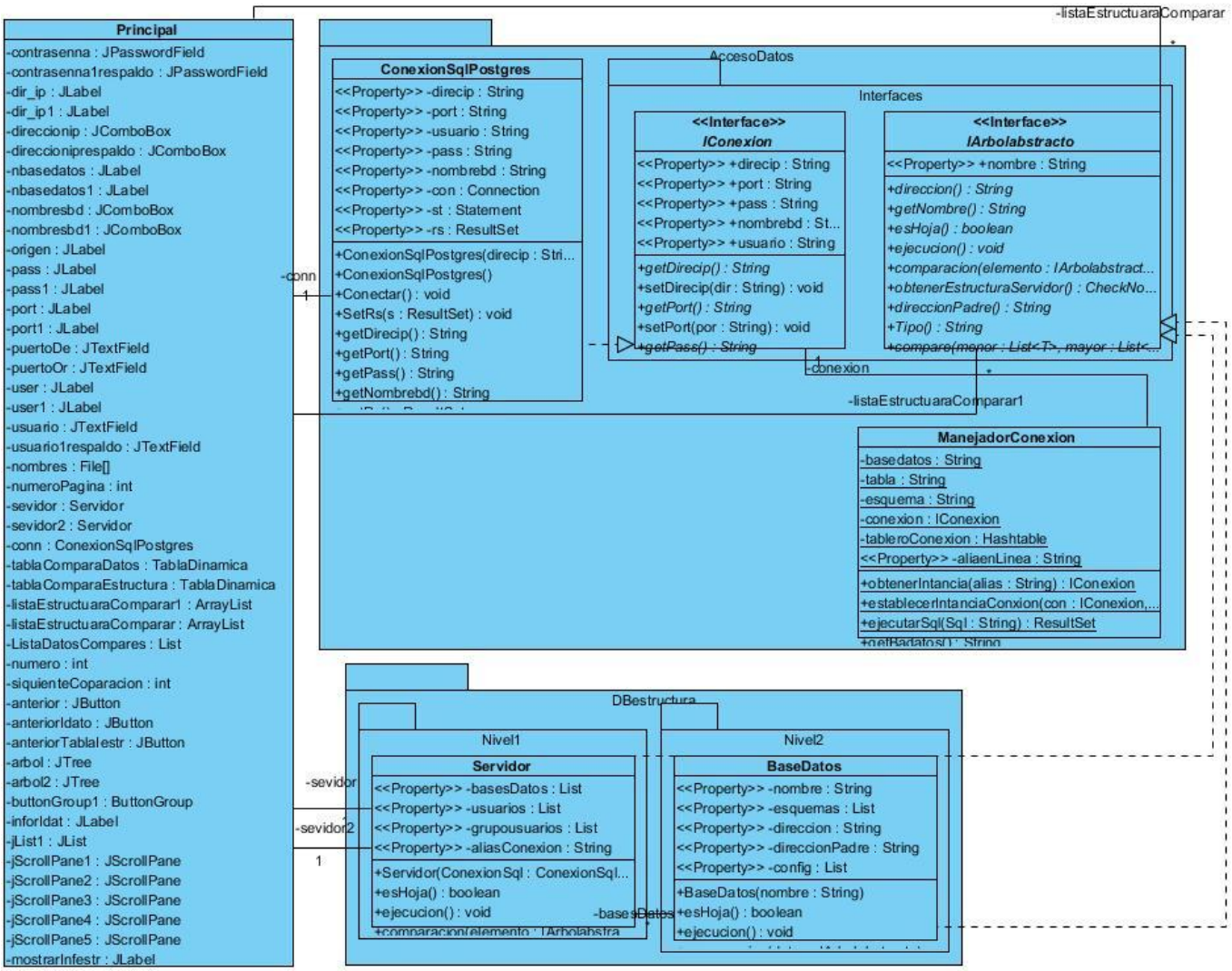


Fig. 8. Diagrama de clases del diseño de los CU Configurar conexiones y Validar conexiones

2.5.3. Diagrama de secuencia

El diagrama de secuencia muestra la forma en que los objetos se comunican entre sí a través de mensajes manteniendo un orden al transcurrir el tiempo.

A continuación se muestra el diagrama de secuencia correspondiente al diagrama de clases del diseño de los CU Configurar conexiones, y Validar conexiones:

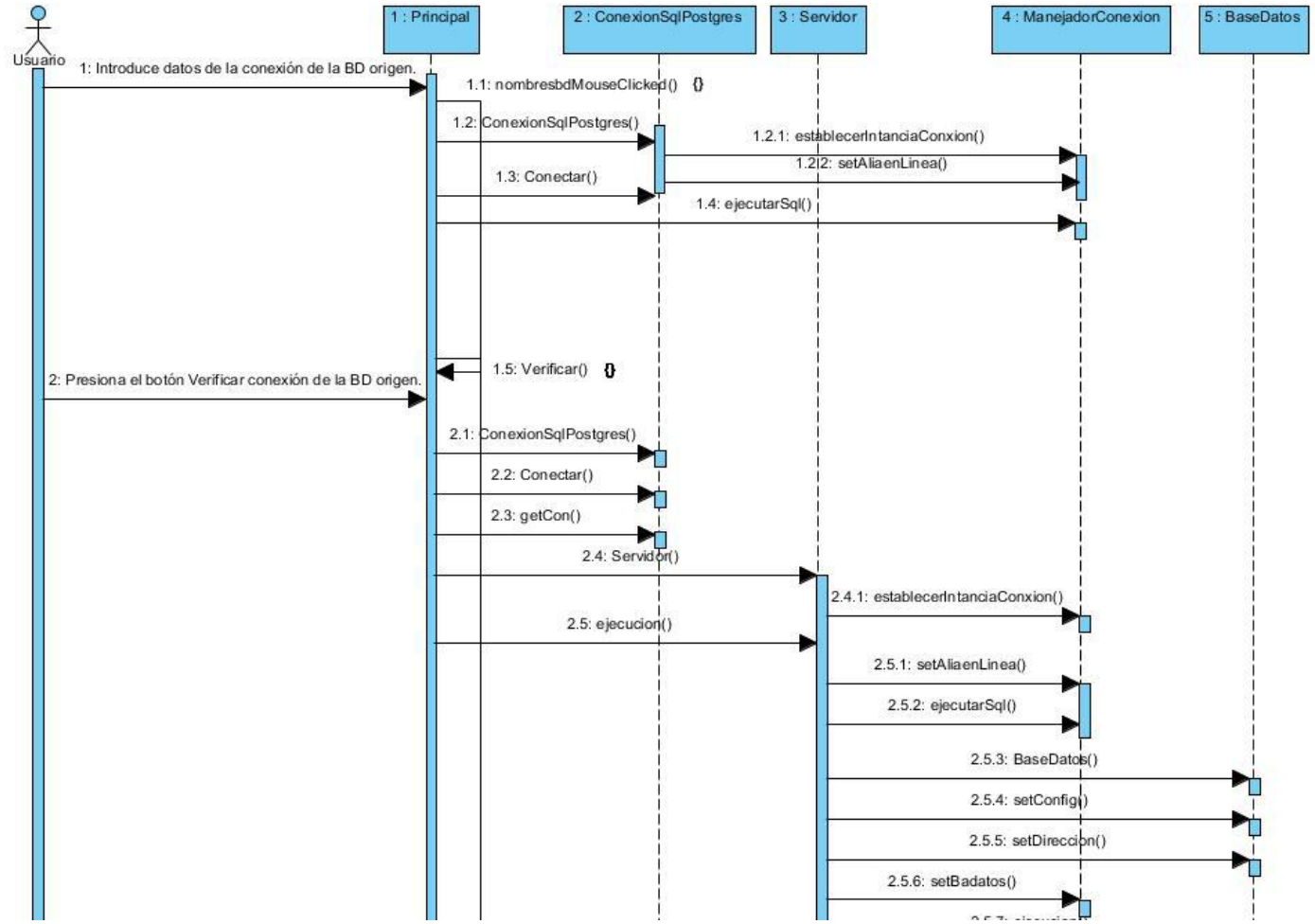


Fig. 9. Diagrama de secuencia de los CU Configurar conexiones y Validar conexiones

2.6. Patrones de diseño utilizados

Los patrones de diseño son los que permiten al arquitecto o creador del software crear una estructura o diseño integrando componentes reusables (Sommerville 2005). Véase el epígrafe 1.5.

A continuación se hará un resumen de cuáles y cómo fueron utilizados.

2.6.1. Patrón Fachada

Provee una interfaz unificada para un conjunto de interfaces de un subsistema. Define una interfaz de alto nivel que hace el subsistema fácil de usar. En el diseño se evidencia en la clase “Principal” que provee una interfaz que sirve de fachada al sistema como tal, propiciando que la navegación sea cómoda pues muestra las vistas necesarias de cada subsistema.

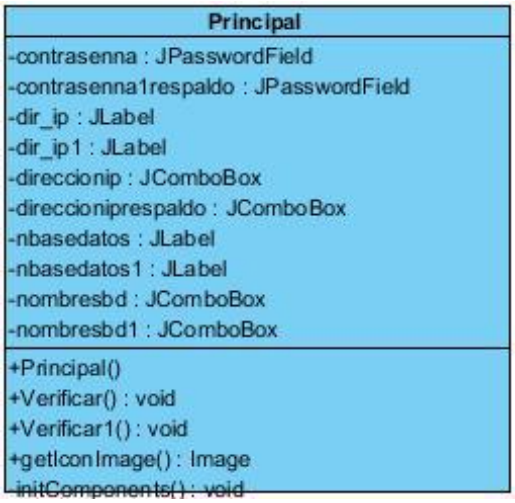


Fig. 10. Clase "Principal" de la herramienta informática

Generalmente el uso de este patrón antes mencionado se ve relacionado con el uso del patrón Instancia única que define la creación una sola instancia para una clase.

La herramienta propuesta utiliza varios patrones de diseño de tipo creación, entre ellos es conveniente destacar el patrón Instancia única que asegura que exista una única instancia de una clase u objeto.

2.6.2. Patrón Instancia única

Se asegura que una clase tiene una sola instancia, y proporcionar un punto de acceso global a ella. Como parte del diseño, se puede observar en la clase ManejadorConexion que contiene los métodos necesarios para asegurar una única instancia de una conexión.

Fig. 11. Clase "ManejadorConexion" de la herramienta informática

2.6.3. Patrón Experto

Indica que la responsabilidad de la creación de un objeto o la implementación de un método, debe recaer sobre la clase que conoce toda la información necesaria para crearlo. Este patrón se ve

referenciado en el diseño de clases que van a utilizar o implementar métodos teniendo en cuenta los atributos que poseen, ejemplo de esto es la clase ConexionSqlPostgres, la cual maneja los datos necesarios para crear los objetos necesarios para cumplir sus funcionalidades.

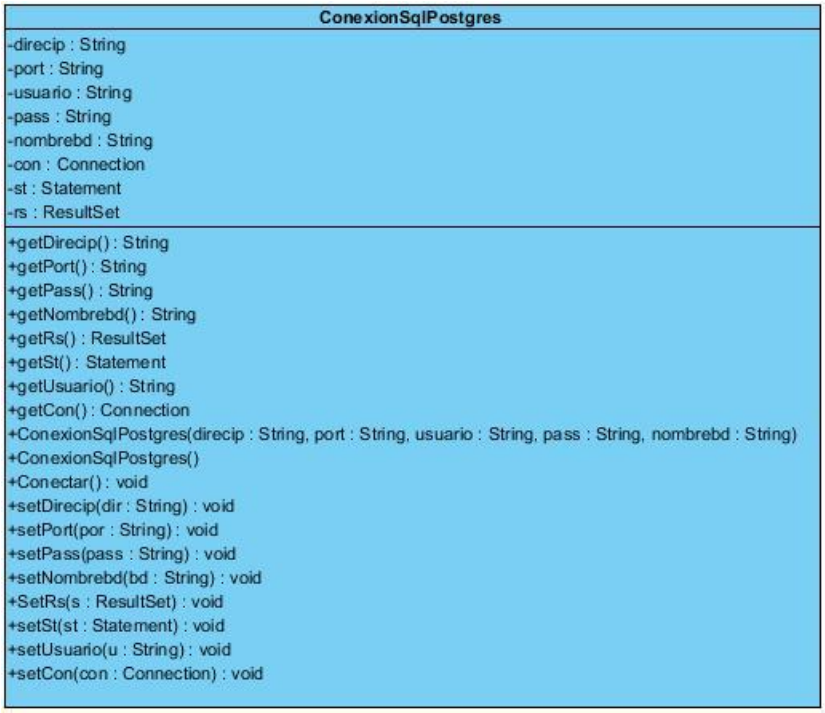


Fig. 12. Clase "ConexionSqlPostgres" de la herramienta informática

2.6.4. Patrón Creador

Generalmente se encuentra estrechamente relacionado con el patrón Experto. Guía la asignación de responsabilidades relacionadas con la creación de objetos. Se aplicará en todos los casos donde una clase tiene la responsabilidad de crear una nueva instancia de la otra (Larman 1999).

Un ejemplo de la aplicación de este patrón es la clase "Principal" la cual crea los objetos necesarios para acceder a los métodos y cumplir con las funcionalidades deseadas. En los métodos señalados se evidencia como la clase "Principal" cuenta con los permisos necesarios para crear objetos, como es el caso del objeto de tipo de clase ConexionSqlPostgres que se utiliza en los métodos Verificar y Verificar1.

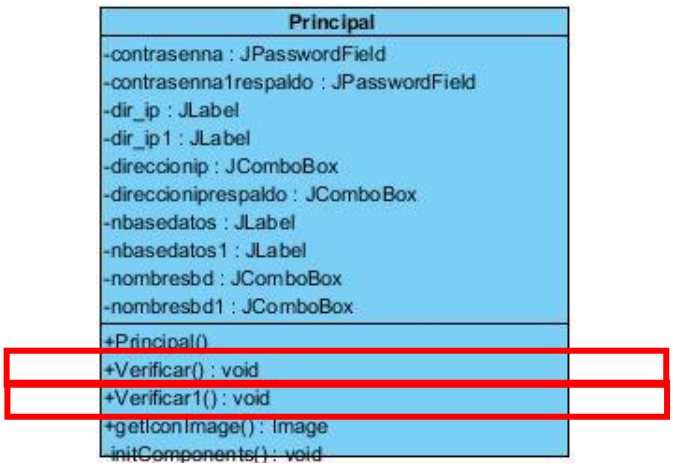


Fig. 13. Clase "Principal" de la herramienta informática

2.6.5. Alta cohesión y bajo acoplamiento

Aunque los conceptos de cohesión y acoplamiento no están íntimamente relacionados, se recomienda tener un mayor grado de cohesión con un menor grado de acoplamiento. De esta forma, se tiene menor dependencia y se especifican los propósitos de cada objeto en el sistema. La alta cohesión representa la coherencia de la información que almacena una clase y la misma debe estar (en la medida de lo posible) relacionada con la clase. El bajo acoplamiento está más ligado a la idea de tener las clases lo menos ligadas entre sí que se pueda. De tal forma que en caso de producirse una modificación en alguna de ellas, se tenga la mínima repercusión posible en el resto de clases, potenciando la reutilización, y disminuyendo la dependencia entre las clases.

Su aplicación se ve ligada en el diseño de clases con los atributos necesarios para su funcionamiento, teniendo en cuenta la relación de los mismos con cada clase, generalmente una alta cohesión implica un bajo acoplamiento pues las clases van a contar con todo lo necesario para cumplir sus funcionalidades. En el modelo del diseño esto se ve ligado a la clase “Pagina” que contiene todo lo necesario para su funcionamiento.

Fig. 14. Clase “Pagina” de la herramienta informática

2.7. Conclusiones del capítulo

Producto de las actividades desarrolladas del flujo de trabajo, modelado, propuesta por la metodología de desarrollo adoptada, AUP; se puede concluir lo siguiente:

- ✓ Se analizaron las funcionalidades y características de la herramienta informática, a través de los

requisitos funcionales y no funcionales. Estos requisitos de software se especificaron y reflejaron en el modelado del sistema, a través de las relaciones entre los actores y CU del sistema descritos en el actual capítulo.

- ✓ Mediante el desarrollo del flujo de trabajo abarcado por AUP se generaron los artefactos correspondientes al modelo de diseño de la herramienta informática, recomendados además por el creador de la metodología. También se alcanzó en la fase elaboración el hito de la arquitectura del sistema con la ayuda de los patrones de diseño orientados a objetos.
- ✓ Se desarrollaron los artefactos definidos para la implementación del sistema: diagramas de clases del diseño.

Capítulo 3: Implementación y pruebas

En el presente capítulo se expondrán los estándares de programación utilizados, así como algunos de los fragmentos de código donde están evidenciados. Finalmente, se le realizarán pruebas de caja negra y caja blanca al software, incluyendo además pruebas de integridad a las BD luego de utilizada la herramienta para garantizar su correcto funcionamiento.

3.1. Estándares de implementación

Los estándares de implementación facilitan el mantenimiento de una aplicación, permiten que cualquier programador entienda y pueda mantener la aplicación, ya que en muy raras ocasiones una misma aplicación es mantenida por su autor original. Los estándares de programación mejoran la legibilidad del código, al mismo tiempo que permiten su comprensión rápida, además permiten definir la nomenclatura de las variables, objetos, métodos y funciones, teniendo en cuenta el orden y legibilidad del código escrito. Seguidamente se muestran las pautas seguidas para la programación del sistema propuesto:

3.1.1. Organización de ficheros

En el fichero java, puede aparecer una instrucción **package**, dicha instrucción lo que hace es indicar que la o las clases definidas en este fichero estarán situadas en la carpeta (paquete), indicado por el nombre del paquete (ANON. 2004 a).

Fichero fuente Java (.java):

El fichero fuente Java contiene una única clase o interfaz pública y el nombre del fichero coincide con el nombre de la clase.

En todo fichero fuente Java se distingue las siguientes secciones:

- **Comentarios de inicio:**

El fichero fuente comienza con un comentario que incluye el nombre de la clase, información sobre la versión del código, la fecha y la declaración de derechos de autor (copyright). El copyright indica la propiedad legal del código, el ámbito de distribución, el uso para el que fue desarrollado y su modificación.

A continuación se muestra el comentario de inicio para la clase "Principal.java".

```
/*
 * @(#)Principal.java 1.0 18/05/12
 *
 * PROPIEDAD DE LA UCI/SOFTWARE BAJO LA LICENCIA GPL.
 */

/**
 * Principal: Es el formulario que contiene, todos los componentes visuales
 * utilizados, además de las funcionalidades requeridas para su uso.
 *
 * @autor Eddy Figueredo Aguilar
 * @autor Odisleysi Martinez Furones
 * @versión 1.0, 18/05/12
 */
```

Fig. 15. Ejemplo en el código del comentario de inicio de la clase "Principal.java"

- **Sentencia de paquete:**

La primera línea no comentada del fichero fuente debe ser la sentencia de paquete, que indica el paquete al que pertenece(n) la(s) clase(s) incluida(s) en el fichero fuente. En este caso:

```
package Visual;
```

Fig. 16. Ejemplo en el código de la sentencia de paquete

- **Sentencias de importación:**

Tras la declaración del paquete se incluyeron las sentencias de importación de los paquetes necesarios. Esta importación de paquetes obligatorios siguió el orden realizado automáticamente por el NetBeans:

```
import java.io.File;
import java.io.FilenameFilter;
import java.awt.CardLayout;
import java.awt.Dimension;
import java.awt.Image;
import java.awt.Toolkit;
import javax.swing.*;
import javax.swing.filechooser.FileNameExtensionFilter;
import javax.swing.tree.*;
import AccesoDatos.ConexionSqlPostgres;
import AccesoDatos.ManejadorConexion;
import DBestructura.Nivel1.Servidor;
```

Fig. 17. Ejemplo en el código de la organización de las sentencias import

3.1.2. Estilo de nombres

- Los nombres de las clases inician con letra mayúscula, utilizando la notación Camello, y si estas tienen más de una palabra, cada palabra nueva debe iniciar con mayúscula y sin utilizar separadores entre ellas, así como los nombres de las funciones, sin embargo, estas deberán brindar una interpretación explícita del contexto para el cual fue creado.

```
public   class ConexionSqlPostgres implements IConexion{
```

Fig. 18. Ejemplo en el código del estilo de los nombres de las clases en la herramienta HIPSI

- Los nombres de las variables inician con letra minúscula, y si estas tienen más de una palabra, cada palabra nueva inicia con mayúscula y sin utilizar separadores entre ellas.

```
private String direcip, port, usuario, pass, nombrebd;
private Connection con;
private Statement st; //Permitir acceder a los datos
private ResultSet rs; //Obtener
```

Fig. 19. Ejemplo en el código del estilo de los nombres de las variables en la herramienta HIPSI

3.2. Diagrama de despliegue

El diagrama se utiliza para modelar el hardware utilizado en la implementación del sistema y las relaciones entre sus componentes.

Los elementos usados por este tipo de diagrama son nodos, componentes y asociaciones. En el UML 2.0 los componentes ya no están dentro de nodos, en cambio, puede haber artefactos (archivo, un programa, una biblioteca o BD) u otros nodos dentro de nodos. Además, el diagrama de despliegue muestra la configuración en funcionamiento del sistema incluyendo su software y su hardware. Para cada componente de un diagrama es necesario que se deban documentar las características técnicas requeridas, el tráfico de red, el tiempo de respuesta, etc.

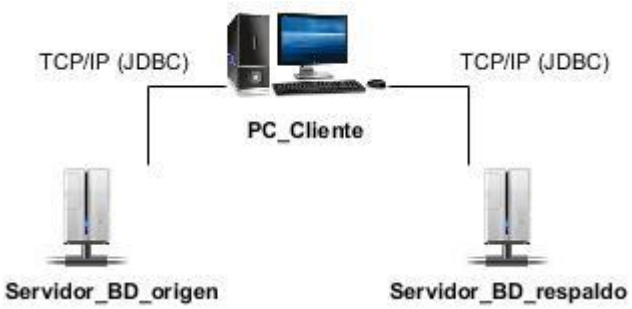


Fig. 20. Diagrama de despliegue

3.3. Aplicación de las funciones de similitud

Teniendo en cuenta el análisis de los algoritmos expuestos, se estableció una guía para la implementación que establece los pasos necesarios para realizar la comparación deseada en la implementación. Se conformó el código a partir de la serie de pasos descritos a continuación:

1. Tratar los objetos seleccionados de las bases de datos como conjuntos independientes.
2. Definir la existencia de un conjunto menor o la igualdad de los mismos.
3. Comparar cada conjunto buscando la ocurrencia del menor en el mayor, definiendo igualdades y diferencias.

Es importante aclarar que este procedimiento se realiza por cada elemento una vez comprobado que son iguales en cuanto al nombre y el tipo, estableciendo una forma de comparación recursiva hacia abajo teniendo en cuenta que cada elemento puede contener un subconjunto.

El código que se refleja a continuación fue obtenido a través de la aplicación de estos pasos:

```
public static List<Tablas> ComparacionDatos(ArrayList<IArbolabstracto> server1, ArrayList<IArbolabstracto>
server2) {
ArrayList<IArbolabstracto> mayor = (server1.size() > server2.size()) ? server1 : server2;
ArrayList<IArbolabstracto> menor = (server1.size() <= server2.size()) ? server1 : server2;
```

```
List<Tablas> resultado = new ArrayList<Tablas>();
boolean[] estavisitado = new boolean[mayor.size()];
for (int i = 0; i < menor.size(); i++) {
boolean esta = false;
if(menor.get(i).Tipo().equals("Tablas")||menor.get(i).Tipo().equals("Esquemas")||menor.get(i).Tipo().equals("BaseDa
tos"))
for (int j = 0; j < mayor.size() && !esta; j++) {
if (!estavisitado[j]) {
if (menor.get(i).Tipo().equals(mayor.get(j).Tipo())&& menor.get(i).getNombre().equals(mayor.get(j).getNombre())) {
esta = true;
estavisitado[j] = true;
resultado.addAll(menor.get(i).obtenerTablas());
}
}
}
if (!esta) {
ConsolaReporte.adicionarInconsistencia(new InconsistenciaDatos(menor.get(i).direccion(),
mayor.get(0).direccionPadre(), menor.get(i).getNombre(), mayor.get(0).direccionPadre(),
Reporte.Enum.Inconsistencia.ESTRUCTURA, Reporte.Enum.FormaInconsistencia.No_Existe,
menor.get(0).getClass() + " no aparecio en la dirección " + mayor.get(0).direccion()));
}
}
for (int i = 0; i < estavisitado.length; i++) {
if (!estavisitado[i]) {
ConsolaReporte.adicionarInconsistencia(new InconsistenciaDatos(menor.get(0).direccionPadre(),
mayor.get(i).direccion(), menor.get(0).direccionPadre(), mayor.get(i).getNombre(),
Reporte.Enum.Inconsistencia.ESTRUCTURA, Reporte.Enum.FormaInconsistencia.No_Existe,
mayor.get(0).getClass() + " no aparece en la dirección " + menor.get(0).direccionPadre()));
}
}
return resultado;
}
}
```

3.4. Tratamiento de inconsistencias en la implementación

Para definir los tipos de inconsistencia en el código se empleó una clase “Enum” que cuenta con los atributos necesarios para establecer las definiciones.

```
public class Enum {
public static enum Inconsistencia {
DATOS, ESTRUCTURA, PROGRAMACION
};
```

```
public static enum FormaInconsistencia {
No_Existe, Son_Incoherente
};
public static enum TipoComparacion {
Datos, Estructura
};
}
```

Dentro de la ejecución de las funcionalidades de comparación tanto de datos como estructura se genera un reporte que describe la inconsistencia. A continuación se coloca un fragmento del código correspondiente.

```
ConsolaReporte.adicionarInconsistencia(new InconsistenciaDatos(menor.get(i).direccion(),
mayor.get(0).direccionPadre(), menor.get(i).getNombre(), mayor.get(0).direccionPadre(),
Reporte.Enum.Inconsistencia.ESTRUCTURA, Reporte.Enum.FormaInconsistencia.No_Existe,
menor.get(0).getClass() + " no aparecio en la dirección " + mayor.get(0).direccion()));
```

3.5. Tratamiento de errores

Se toma en cuenta el tratamiento de errores en la herramienta informática que se implementó, ayudando así al correcto funcionamiento de la misma y el bienestar del usuario. Teniendo como objetivo que se garantice la integridad de la información, se remedian, reducen y tratan los errores. Para su tratamiento se validan las entradas de los datos en las configuraciones de las conexiones por el usuario a la herramienta. Se valida que el formato de los datos sea el esperado, los errores son capturados y gestionados en los métodos implementados, en los cuales se definen los errores y cómo deben ser mostrados de manera que el usuario pueda entenderlos y el manejo de errores se realiza en dependencia del tipo de error lanzado.

3.6. Seguridad

La herramienta no brinda la posibilidad de tener una jerarquía de usuarios, o sea, que se puedan autenticar y estar registrados, que se les muestre las funcionalidades de acuerdo con sus permisos. Esto se debe a que es una herramienta de escritorio para usuarios específicos como administradores de BD, la herramienta garantiza a través de la conexión al servidor que sea el administrador de la BD quien la utilice, de lo contrario no permite conectarse a la misma.

3.7. Métricas propuestas por Chidamber y Kemerer. Aplicación al paquete BDestructura.

Las métricas seleccionadas fueron las propuestas por Chidamber y Kemerer, véase el epígrafe 1.10.

3.7.1. Acoplamiento entre objetos (CBO)

Para aplicar esta métrica es necesario conocer la cantidad de clases con las que se relaciona una clase excluyendo las relaciones por herencia. Para aplicarla a este caso se definió como indicador de

aceptación el tres, por lo que las clases que muestren más de tres relaciones serán consideradas con un alto acoplamiento.

Clases	No. Colaboraciones
Tablas	7
Vistas	4
Esquemas	8
Dominios	2
Secuencias	2
Funcdisparadores	2
Funciones	2
Restricciones	2
Reglas	2
Indices	2
Disparadores	2
Columna	2
Usuarios	0
BaseDatos	3
GruposUsuarios	0
Servidor	3

Tabla 6. Clases y número de colaboraciones

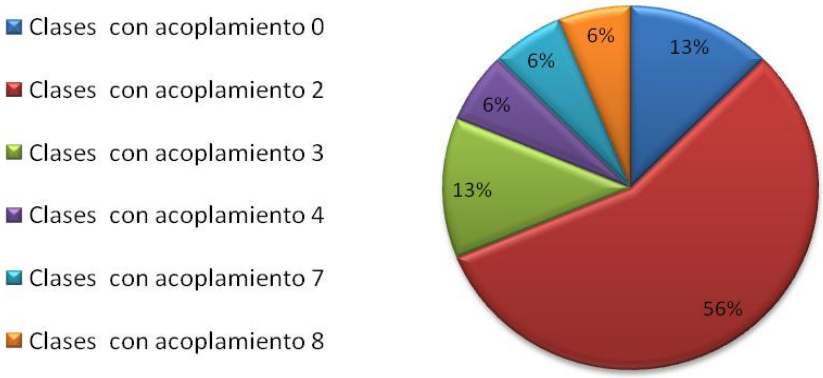


Fig. 21. Acoplamiento entre objetos

Como se muestra en la figura el 82 por ciento de las clases presentan un bajo acoplamiento al no sobrepasar el indicador establecido y el 18 por ciento restante –que lo constituyen solamente tres clases- presentan un acoplamiento alto.

Esto permite identificar que, las clases con bajo acoplamiento (13 de 16) tienen un alto grado de reutilización y que las pruebas o modificaciones que sean necesarias realizarles, serán más sencillas de ejecutar. Mientras que para las restantes tres clases estos valores se invierten.

3.7.2. Profundidad del árbol de herencia (DIT)

Para aplicar esta métrica es necesario conocer la cantidad de relaciones de herencia que tiene una clase. Los autores de la métrica proponen como indicador de permisibilidad cinco niveles de profundidad. Seguidamente se muestran los datos recopilados y su representación gráfica.

Clases	Profundidad árbol de herencia
Tablas	0
Vistas	0
Esquemas	0
Dominios	0
Secuencias	0
Funcdisparadores	0
Funciones	0
Restricciones	0
Reglas	0
Indices	0
Disparadores	0
Columna	0
Usuarios	0
BaseDatos	0
GruposUsuarios	0
Servidor	0

Tabla 7. Clases y profundidad

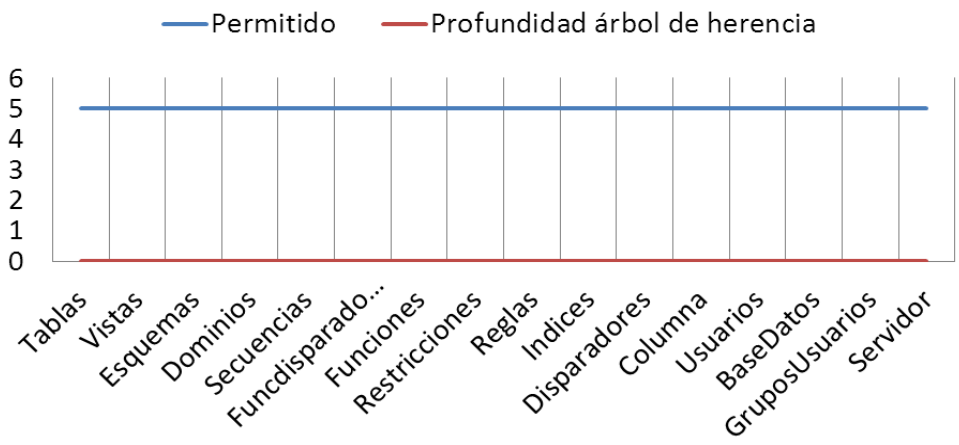


Fig. 22. Profundidad en el árbol de herencia

La imagen muestra en color rojo el nivel que tiene cada una de las clases en el árbol de herencia. Como se observa las 16 clases tienen valor cero lo que indica que no tienen relaciones de herencia. Estos valores son claramente inferiores al 5 sugerido por los autores de la métrica.

Estos resultados indican que las clases presentan una baja complejidad y un alto potencial de reutilización, además la probabilidad de detección de fallos será menor. En resumen, las clases serán fáciles de desarrollar y mantener.

La aplicación de estas dos métricas a las clases del paquete BDestrcutura permite plantear que existe un alto potencial para la reutilización al tiempo que serán fáciles de modificar y mantener.

3.8. Pruebas

Las pruebas son un punto fundamental en el proceso de desarrollo del software debido a que estas dan la garantía de la calidad que va a tener el software. Constituyen una revisión final de las

especificaciones, del diseño y de la codificación(R. S Pressman 2002). Para la realización de las pruebas se diseñan casos de pruebas, las cuales tienen como objetivo verificar el resultado satisfactorio del software.

Un caso de prueba especifica una forma de probar el sistema, incluyendo la entrada o resultado con la que se ha de probar y las condiciones bajo las que ha de probarse. Es un conjunto de entradas de prueba, condiciones de ejecución, y resultados, desarrollados para un objetivo concreto, tal como probar un camino concreto a través de un CU, o verificar que se cumple un requisito específico (Jacobson, Booch, Rumbaugh 2000).

A continuación se exponen las empleadas para probar las especificaciones del diseño y la codificación de la herramienta informática.

3.8.1. Pruebas de caja blanca

Los casos de prueba se basan en el conocimiento que se tenga acerca de la estructura del código fuente (Angélica de Antonio [no date]).

Existen varios tipos de pruebas de caja blanca, ellos son:

- **Prueba de condición:** Es un método de diseño de casos de prueba que ejercita las condiciones lógicas contenidas en el módulo de un programa.
- **Prueba de flujo de datos:** Se seleccionan caminos de prueba de un programa de acuerdo con la ubicación de las definiciones y los usos de las variables del programa.
- **Prueba de bucles:** Es una técnica de prueba de caja blanca que se centra exclusivamente en la validez de las construcciones de bucles.
- **Prueba del camino básico:** Esta técnica permite obtener una medida de la complejidad lógica de un diseño y usar esta medida como guía para la definición de un conjunto básico (Cortés 2000).

En este caso se aplicará la llamada prueba de camino básico con el objetivo de comprobar que cada línea de código se ejecuta al menos una vez, es decir, obtener una medida de la complejidad lógica de un diseño y usar esta medida como guía para la definición de un conjunto básico.

A continuación se definen los pasos necesarios para realizar dicha prueba:

1. A partir del diseño o del código fuente, se dibuja el grafo de flujo asociado.
2. Se calcula la complejidad ciclomática del grafo.
3. Se determina un conjunto básico de caminos independientes.
4. Se preparan los casos de prueba que obliguen a la ejecución de cada camino del conjunto básico.

Un grafo de flujo está constituido por tres componentes fundamentales:

Nodo: Es usado para representar una o más secuencias procedimentales, es decir, representa cada línea del código que pertenece a la funcionalidad a evaluar.

Arista: Es la relación entre cada nodo, e implica el sentido del flujo de la ejecución de cada línea de código.

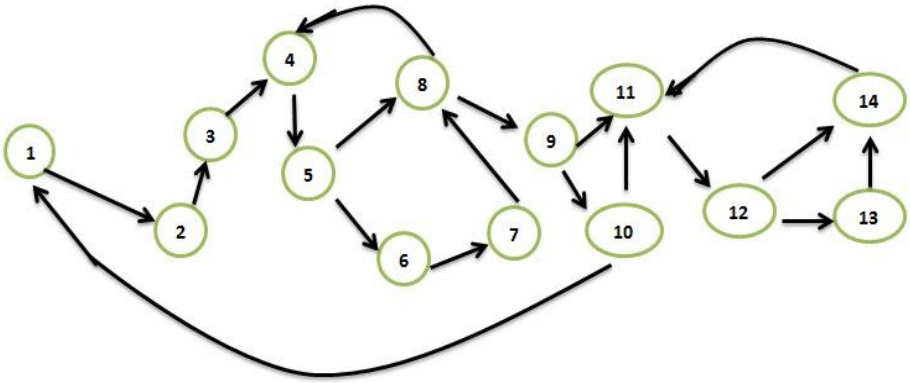
Regiones: Son las áreas cerradas delimitadas por las aristas, estas representan la cantidad de caminos independientes al ejecutarse un programa.

Para realizar la prueba de camino básico es necesario seleccionar una funcionalidad del sistema, en este caso se seleccionó la funcionalidad ComparacionEstructura, por ser considerada una funcionalidad importante para el funcionamiento de la herramienta. A continuación se muestra el código y selección de las áreas que representan cada nodo:

```
public static void ComparacionEstructura(ArrayList<IArbolabstracto> server1, ArrayList<IArbolabstracto> server2) {
    ArrayList<IArbolabstracto> mayor = (server1.size() > server2.size()) ? server1 : server2;
    ArrayList<IArbolabstracto> menor = (server1.size() <= server2.size()) ? server1 : server2;
    boolean[] estavisitado = new boolean[mayor.size()];
     for (int i = 0; i < menor.size(); i++) {
        boolean esta = false;
       for (int j = 0; j < mayor.size() && !esta; j++) {
          if (!estavisitado[j]) {
             if (menor.get(i).comparacion(mayor.get(j))) {
                esta = true;
                estavisitado[j] = true;
             }
          }
       }
       if (!esta) {
ConsolaReporte.adicionarInconsistencia(new InconsistenciaDatos(menor.get(i).direccion(),
mayor.get(0).direccionPadre(), menor.get(i).getNombre(),
mayor.get(0).direccionPadre(),Reporte.Enum.Inconsistencia.ESTRUCTURA,
Reporte.Enum.FormaInconsistencia.No_Existe, menor.get(0).getClass() + " no aparecio en la dirección " +
mayor.get(0).direccion()));
       }
    }
    for (int i = 0; i < estavisitado.length; i++) {
       if (!estavisitado[i]) {
ConsolaReporte.adicionarInconsistencia(new InconsistenciaDatos(menor.get(0).direccionPadre(),
mayor.get(i).direccion(), menor.get(0).direccionPadre(), mayor.get(i).getNombre(),
Reporte.Enum.Inconsistencia.ESTRUCTURA, Reporte.Enum.FormaInconsistencia.No_Existe,
mayor.get(0).getClass() + " no aparece en la dirección " + menor.get(0).direccionPadre()));
       }
    }
 }
```

1 2 3 4 5 6 7 8 9 10 11 12 13 14

Continuando con la aplicación de la prueba se construye un grafo el cual representa el flujo de la funcionalidad y cada arista indica todos los posibles caminos a seguir.



A partir de este nodo se obtiene la complejidad ciclomática, cuenta con tres fórmulas las cuales deben dar el mismo resultado, esta dada por las siguientes fórmulas:

1. V (G) = (A – N) + 2 Donde "**A**" es el número de aristas y "**N**" el de nodos.
2. V (G) = P + 1 Donde "**P**" es la cantidad total de nodos predicados (son los nodos de los cuales parten dos o más aristas).
3. V (G) = R +1 Donde "**R**" es la cantidad total de regiones.

Obteniendo como resultado:

1. V (G) = (19 – 14) + 2 = 7
2. V (G) = 6 + 1= 7
3. V (G) = 6+1 = 7

Como se puede observar se obtienen los mismos resultados, a continuación se definen los caminos independientes:

R1: 1-2-3-4-5-6-7-8-9-10-11-12-13-14
R2: 1-2-3-4-5-8-9-10-11-12-13-14
R3: 1-2-3-4-5-8-9-10-11-12-14
R4: 1-2-3-4-5-4-6-7-8-9-11-12-13-14
R5: 1-2-3-4-5-4-5-6-7-8-4-5-6-7-8-9-10-11-12-13-14
R6:1-2-3-4-5-8-4-5-8-9-10-11-12-14
R7:1-2-3-4-5-8-4-5-8-9-11-12-13-14

Luego de obtener los caminos básicos es necesario realizar los casos de prueba, debe obtenerse al menos uno por cada camino definido, a continuación se muestra un ejemplo de estos:
La realización de los casos de prueba se ejemplificará con la descripción de un caso de prueba para el camino básico #1.

Caso de prueba para el camino básico #1:

Descripción:

Los parámetros de entrada serán listas de objetos de la clase seleccionada y representan la estructura de las BD, las listas no deben estar vacías.

Condición de ejecución:

Se tendrán como objetos de entrada dos listas de objetos que representan a ambos servidores, esto implica la selección de los elementos estructurales a comparar.

Entrada:

ArrayList<Esquema> server1, lista de elementos estructurales correspondientes a la selección de los elementos del primer servidor.

ArrayList< Esquema > server2, lista de elementos estructurales correspondientes a la selección de los elementos del segundo servidor.

Resultados esperados:

Se espera que permita mostrar las inconsistencias encontradas entre las estructuras pasadas por parámetro.

Luego de aplicado los casos de prueba, se pudo comprobar que el flujo de trabajo de la funcionalidad está correcto. Este tipo de prueba fue aplicado además a otras funcionalidades definidas como importantes para el funcionamiento del sistema obteniéndose resultados satisfactorios, validando así el código del sistema.

3.8.2. Pruebas de caja negra

Las pruebas de caja negra están dadas desde el punto de vista de las entradas que reciben y las salidas o respuestas que producen, sin tener en cuenta su funcionamiento interno. En otras palabras, de una caja negra solo interesará su forma de interactuar con el medio que le rodea (en ocasiones, otros elementos que también podrían ser cajas negras) entendiendo qué es lo que hace, pero sin dar importancia a cómo lo hace. Por tanto, de una caja negra deben estar muy bien definidas sus entradas y salidas, es decir, su interfaz; en cambio, no se precisa definir ni conocer los detalles internos de su funcionamiento (Masuelli 2001).

Para la realización de estas pruebas existen diferentes técnicas dentro de las que se encuentra:

- **Métodos de pruebas basados en grafos:** Este método está dirigido a las relaciones que existen entre los objetos del programa y su comportamiento.
- **Partición equivalente:** Esta técnica divide el campo de entrada en clases de datos que tienden a ejercitar determinadas funciones del software.
- **Análisis de valores límite:** Prueba la habilidad del programa para manejar datos que se encuentran en los límites aceptables.
- **Prueba de la tabla ortogonal:** Suministra un método sistemático y eficiente para probar sistemas con un número reducido de parámetros de entrada (Morales Mejía, González, María 2010).

En este caso se realizarán las pruebas de caja negra de particiones equivalentes, ya que mediante esta técnica se ven, por las entradas, las respuestas que da el sistema y así identificar inmediatamente

errores que existan en la herramienta. A continuación se muestra el caso de prueba referente al CU Configurar conexiones:

Aplicación de la técnica de particiones equivalentes CU "Configurar conexiones":

Descripción general:

El CU se inicia cuando inicia el sistema y muestra la interfaz principal de la herramienta, el Usuario introduce los datos correspondientes a la BD que el Usuario determine como la BD origen (BD correcta) e introduce los datos de la BD respaldo (BD incorrecta), para luego invocar el CU incluido Validar conexiones al presionar los botones Verificar conexión de la BD origen y la BD respaldo.

Condiciones de ejecución:

El sistema debe estar ejecutándose correctamente y en caso de ejecutar la aplicación fuera de los servidores de BD, debe existir conexión de red con estos.

Secciones a probar en el CU:

Escenarios del Configurar conexiones	**Descripción de la funcionalidad**	**Flujo Central**
EC 1: **Configurar conexiones**	Configurar las conexiones satisfactoriamente.	Muestra la interfaz para configurar conexiones. Se introducen los datos correspondientes. Se verifican las conexiones de las BD origen y BD respaldo. Se configuran las conexiones. Habilita el botón siguiente, para continuar con la navegación. Se presiona el botón Siguiente. Muestra un mensaje "Seleccione los objetos a comparar.". Se presiona el botón Aceptar del mensaje. Muestra la interfaz Configurar comparación. Ver DCP **Configurar comparación**.
EC 2: **Validar conexión**	Verificar las conexiones de la configuración de las conexiones.	Se presiona el botón Verificar conexión de la BD origen. Muestra mensaje de conexión satisfactoria. Se presiona el botón Aceptar. Se presiona el botón Verificar conexión de la BD respaldo. Muestra mensaje de conexión satisfactoria. Se presiona el botón Aceptar. Se regresa al escenario anterior.
EC 3: **Datos incompletos**	Luego de haber introducido los datos, el sistema los verifica, de haber incompletos muestra un mensaje de datos incompletos.	Muestra la interfaz para configurar conexiones. Se introducen los datos incompletos. Se presiona el botón Verificar conexión. Muestra un mensaje de campos incompletos. Se presiona el botón Aceptar. Se regresa al escenario 1.
EC 4: **Conexión fallida**	Luego de haber introducido los datos, el sistema los verifica la conexión, de haber datos incorrectos o	Muestra la interfaz para configurar conexiones. Se introducen los datos incorrectos o correctos y el servidor no es en la misma máquina y existen problemas con la conexión. Se presiona el botón Validar conexión. Muestra un mensaje de conexión fallida.

	problemas con la conexión muestra un mensaje de conexión fallida.	Se presiona el botón Aceptar. Se regresa al escenario 1.
EC 5: **Salir**	Salir de la herramienta.	Se sale de la herramienta.

Tabla 8. Secciones a probar en el CU

Matriz de datos:

Sección 1 Configurar conexiones:

Id del escenario	**EC 1**	**EC 2**	**EC 3**	**EC 4**	**EC 5**
Escenario	Configurar conexiones	Validar conexión	Datos incompletos	Conexión fallida	Salir
Variable 1 Dirección IP	V	V	I	I	I
Variable 2 Puerto	V	V	I	I	V
Variable 3 Usuario	V	V		V	I
Variable 4 Contraseña	V	V	I	V	V
Variable 5 Nombre BD	V	V		V	V
Botón 1 (Verificar conexión)	V	V	V	V	N/A
Botón 2 (Verificar conexión)	V	V	V	V	N/A
Botón 3 (Salir)	N/A	N/A	N/A	N/A	V
Botón 4 (Siguiente)	V	N/A	V	V	N/A
Respuesta del Sistema	Se configura las conexiones de manera satisfactoria y se visualiza la interfaz Configurar comparación.	Se verifica que las conexiones se realicen satisfactoriamente. Muestra un mensaje para cada botón de conexión satisfactoria. Regresa a la vista anterior.	Muestra un mensaje de información, faltan datos por llenar.	Muestra un mensaje de conexión fallida.	Se sale de la herramienta.
Resultado de la Prueba	Satisfactoria	Satisfactoria	Satisfactoria	Satisfactoria	Satisfactoria

Tabla 9. Matriz de datos de la sección 1: Configurar conexiones

Resultados de las pruebas de caja negra:

Las pruebas de caja negra fueron realizadas a todos los CU del sistema en 4 iteraciones. Una vez concluidas dichas pruebas arrojaron resultados exitosos. Por tanto, el sistema está apto para su utilización, ya que cumple con todas las funcionalidades para las que fue concebido. La presente gráfica da una mejor visión de los resultados de las pruebas de particiones equivalentes.

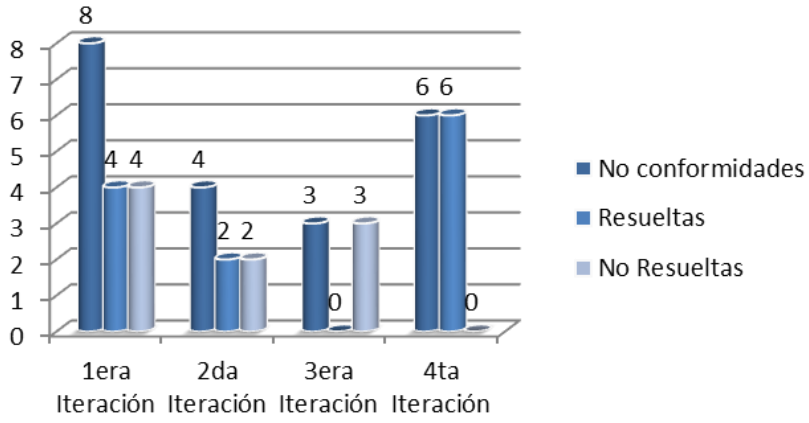


Fig. 23. Resultado de las pruebas de particiones equivalentes

3.8.3. Pruebas de integridad de datos y BD

Este tipo o técnica de prueba se realiza con el objetivo de garantizar que luego de utilizada la herramienta, hayan sido eliminadas las inconsistencias entre las BD.

Objetivo de la técnica:	Verificar la integridad de los datos, mediante inspección en cada una de las BD para evitar inconsistencia en los datos y estructura entre las BD analizadas.
Técnica	Consultas SQL que permitan constatar la integridad de los datos, además de la consistencia entre las BD.
Propósito:	Analizar que los datos y la estructura de ambas BD sean los mismos luego de utilizada la herramienta.
Herramientas requeridas:	Gestor de BD, en este caso PostgreSQL 8.4.
Criterio de éxito	Los datos son persistentes y congruentes en ambas BD pertenecientes a cada servidor. Se obtienen los mismos resultados luego de aplicadas las consultas.
Consideraciones Especiales:	Tener en cuenta que los datos entrados para establecer la conexión sean correctos.

Muestra de consultas SQL utilizadas, para una base datos de prueba y su respaldo:

1. Select denominacion from mod_prueba.t_prueba
2. Select idpersona from mod_persona.t_persona
3. Select ubicacion from mod_ubicacion.t_ubicaciones

Para la ejecución de la prueba se realizaron una serie de consultas a ambas BD por separado, es decir, se le aplica la misma consulta a las BD, luego de ejecutadas las consultas el resultado fue el mismo por cada ejecución, obteniéndose resultados satisfactorios.

3.9. Conclusiones del capítulo

Producto de las actividades desarrolladas de los flujos de trabajo, implementación y prueba, propuestos por la metodología de desarrollo adoptada, AUP; se puede concluir lo siguiente:

- ✓ Se implementaron los CU necesarios, teniendo en cuenta los estándares de programación y el tratamiento de errores. Se desarrolló sobre el lenguaje de programación Java, a través de entorno de desarrollo integrado NetBeans 7.1. Para el uso de la herramienta se utilizará la versión libre del kit de desarrollo de Java OpenJDK 7.
- ✓ Se decidió diseñar los casos de pruebas de integración, de caja negra y caja blanca para el sistema y aplicarlas para validar las funcionalidades implementadas, así como las pruebas de integridad de los datos a la BD.
- ✓ Se realizaron las pruebas a la herramienta informática desarrollada, comprobando de esta forma que las BD que se analicen con la herramienta se encontrarán en un estado íntegro.
- ✓ Se redactó el trabajo de diploma, el cual respalda toda la investigación, donde se exponen los principales resultados del trabajo.

Conclusiones generales

En el presente trabajo se propuso una herramienta para la identificación y reparación de inconsistencias de datos y de estructuras entre BD PostgreSQL, enfocada en el proceso de actualización de las BD con el objetivo de contribuir a su mantenimiento y potenciando la integridad de sus datos. Se cumplieron los objetivos propuestos y se arribó a las siguientes conclusiones:

- ✓ Se realizó un estudio del estado del arte del proceso de actualización de BD incluidos los algoritmos empleados para ello y las herramientas informáticas desarrolladas para este fin, logrando identificar las principales tendencias y concluyendo que no existe ninguna herramienta que solucione el problema identificado en esta investigación.
- ✓ Se aplicaron la metodología y arquitectura definidas generando los entregables previstos para cada fase del proceso de desarrollo, lo que permitió modelar la solución.
- ✓ Se utilizaron estándares de programación y patrones de diseño logrando la implementación de una herramienta informática para solucionar inconsistencias en BD PostgreSQL.
- ✓ La solución informática desarrollada fue sometida a un proceso de validación a través de la aplicación de métricas al diseño así como de pruebas de caja blanca y caja negra al código y a las interfaces del sistema respectivamente, finalmente, se realizaron pruebas de integridad a las BD comparadas para validar la ausencia de inconsistencias. En todos los casos, los resultados finales fueron satisfactorios.

Recomendaciones

Para posteriores versiones de la herramienta es necesario se tomen en cuenta, en función de un mejor desempeño del personal en las próximas versiones del proyecto, las siguientes recomendaciones:

- Extender la herramienta en versiones posteriores para que trabaje con PostgreSQL en su versión 9 en adelante, ya que actualmente la herramienta trabaja con la versión 8.4.
- Ampliar las comparaciones de la herramienta a encontrar inconsistencias dentro de una misma Base Datos.
- Agregar al sistema una funcionalidad que permita la persistencia de los cambios que se hagan en las BD de manera que luego puedan ser consultados.

Bibliografía

- ALEXANDER, C., ISHIKAWA, S. and SILVERSTEIN, M., 1977. *A pattern language: towns, buildings, construction*. S.l.: Oxford University Press, USA. ISBN 0195019199.
- AMBLER, Scott W., 2011. Ambysoft Home Page. In: [online]. 7 July 2011. [Accessed 20 April 2012]. Available from: http://www.ambysoft.com/.
- AMÓN, Iván and JIMÉNEZ, Claudia, 2009. *Funciones de Similitud sobre Cadenas de Texto: Una Comparación Basada en la Naturaleza de los Datos*. 2009. S.l.: s.n.
- ANGÉLICA DE ANTONIO, [no date]. GESTIÓN, CONTROL Y GARANTÍA DE LA CALIDAD DEL SOFTWARE. In: .
- ANON., 2004a. Estructura de un fichero java. In: [online]. 23 April 2004. [Accessed 12 June 2012]. Available from: http://www.emagister.com/curso-java/estructura-fichero-java.
- ANON., 2004b. Tutorial de programación en shell: ¿Qué es un guión (script)? In: [online]. 9 June 2004. [Accessed 19 April 2012]. Available from: http://www.demiurgo.org/doc/shell/shell-2.html.
- ANON., 2010a. Estudio de EMC proyecta gran crecimiento de la información en 10 años. In: [online]. 27 April 2010. [Accessed 22 February 2012]. Available from: http://www.colombianproductions.com/mymit/joom1515/index.php?option=com_content&view=article&id=153:estudio-de-emc-proyecta-gran-crecimiento-de-la-informacion-en-10-anos&catid=38:noticias&Itemid=55.
- ANON., 2010b. Tipos de copia de seguridad. In: [online]. 15 2010. Available from: http://www.ite.educacion.es/formacion/materiales/85/cd/REDES_LINUX/backup/Tipos_de_copia_de_seguridad.html.
- BAKKEN, Stig Sæther, AULBACH, Alexander, SCHMID, Egon, WINSTEAD, Jim, WILSON, Lars Torben, LERDORF, Rasmus, SURASKI, Zeev, ZMIEVSKI, Andrei and AHTO, Jouni, 2001. *Manual de PHP*. 15 March 2001. S.l.: s.n.
- BÖCK, H., 2011. IntelliJ IDEA and the NetBeans Platform. In: *The Definitive Guide to NetBeans™ Platform 7*. 2011. pp. 431–437.
- CÁCERES, P., MARCOS, E. and KYBELE, G., 2002. *Procesos ágiles para el desarrollo de aplicaciones Web.* April 2002. S.l.: s.n. Grupo Kybele, Departamento de Ciencias Experimentales e Ingeniería. Universidad Rey Juan Carlos Móstoles, Madrid (España), Retrieved May. 2002.
- CALVOPIÑA MORILLO, J. C and VELASCO PACHA, V. P, 2012. Artículo Científico-Comparación de los sistemas de gestión de contenidos, de software libre: joomla, drupal, liferay y aplicación al caso práctico para la agencia de viajes Shinegalapagos. In: 2012.
- CANÓS, J. H., LETELIER, Patricio and PENADÉS, Ma Carmen, 2009. *Metodologías Ágiles en el Desarrollo de Software.* 2009. S.l.: s.n. DSIC-Universidad Politécnica de Valencia.
- CARRILLO, Diego Mauricio Paz, 2006. Evolucion de la Filosofia del Conocimiento en las Metodologias de Desarrollo Software - Grupo GNU/Linux de la Universidad del Cauca. In: [online]. 14 February 2006. [Accessed 22 March 2012]. Available from: http://gluc.unicauca.edu.co/wiki/index.php/Evolucion_de_la_Filosofia_del_Conocimiento_en_las_Metodologias_de_Desarrollo_Software.
- CHIDAMBER, Shyam R. and KEMERER, Chris F., 1994. *A Metrics Suite for Object Oriented Design*. S.l.: IEEE Transactions on Software Engineering.
- COMMITTEE, Software Engineering Standards, 2000. *IEEE recommended practice for architectural description of software-intensive systems*. S.l. Technical Report IEEE Std 1471-2000, IEEE Computer Society.
- CORTÉS, E. L, 2000. Pruebaydocumentaciónde programas. Técnicas. In: *Informática: Temario" A" de oposiciones al cuerpo de profesores de enseñanza secundaria*. 2000. Vol. 2, pp. 261.
- DEVART, 2004. dbForge Comparador de Esquemas para SQL Server (dbForge Schema Compare for SQL Server) por Devart - reporte y descarga. In: [online]. 10 August 2004. [Accessed 11 June 2012]. Available from: http://www.freedownloadmanager.org/es/downloads/dbForge_Comparador_de_Esquemas_para_SQL_Server_60182_p/.

- DOMÍNGUEZ, A. H, HERNÁNDEZ, J. G.R, SOTOLONGO, L. H, ACOSTA, Y. A and DÍAZ, A. M, 2010. CENTRO COORDINADOR PARA LA FORMACIÓN Y DESARROLLO DEL CAPITAL HUMANO. FORDES. In: 2010.
- D-SOFTS, 2009. D-Softs Database Comparer Version 2.0 - A powerful MSSQL database compare tool | PRLog. In: [online]. 11 June 2009. [Accessed 11 June 2012]. Available from: http://www.prlog.org/10256335-dsofts-database-comparer-version-20-powerful-mssql-database-compare-tool.html.
- DUNN, H. L, [no date]. Halbert L. Dunn--Please Click to Return to Front Page. In: .
- DUQUE, Raúl González, 2012. *Tutorial de Python* [online]. 20 January 2012. S.l.: s.n. [Accessed 23 February 2012]. Available from: http://mundogeek.net/tutorial-python/.
- EMC CORPORATION, 2008. El nuevo fenómeno mundial: la "Sombra Digital."In: [online]. 2008. [Accessed 12 June 2012]. Available from: http://argentina.emc.com/about/news/press/2008/20080311-01.htm.
- EMS SOFTWARE DEVELOPMENT, 2010. EMS DB Comparer for PostgreSQL 3.3 Descargar Gratis - PostgreSQL DB comparación y sincronización. In: [online]. 17 February 2010. [Accessed 11 June 2012]. Available from: http://www.filecluster.es/programas/EMS-DB-Comparer-for-PostgreSQL-96532.html.
- EMS SOFTWARE DEVELOPMENT, 2011. Página del autor - EMS Software Development - Softpedia en español. In: [online]. 2011. [Accessed 11 June 2012]. Available from: http://www.softpedia.es/autor-EMS-Software-Development-19199.html.
- ESPINOSA, Jose Edgar Juarez, 2009. Respaldo de Información. In: [online]. 30 August 2009. [Accessed 24 February 2012]. Available from: http://sistemasmultiempresa.com/RespaldodeInformacion.aspx.
- ESTRADA, Joel Michel León, 2011. *Software para la conversión de múltiples capas de información geográfica de ESRI Shapefile a PostgreSQL* [online]. Cuba: Universidad de las Ciencias Informáticas. Available from: http://bibliodoc.uci.cu/TD/TD_04364_11.pdf.
El formato ESRI Shapefile es considerado uno de los más difundidos actualmente en el mundo de la cartografía digital. Por otra parte, PostgreSQL se ha convertido en un medio de referencia en lo que respecta al almacenamiento de datos geográficos en Bases de Datos. PostgreSQL cuenta con la extensión PostGIS que permite manejar los datos georreferenciados y realizar un sinnúmero de operaciones sobre estos, facilitando el análisis de la información. Es por ello que el proceso de conversión de datos de ESRI Shapefile a PostgreSQL se ha hecho bastante frecuente en el desarrollo de Sistemas de Información Geográfica (SIG). Debido a que la Universidad de las Ciencias Informáticas (UCI) está inmersa en el desarrollo de varios SIG, en el presente trabajo se propone el desarrollo de una herramienta capaz de transformar múltiples archivos Shapefile a PostgreSQL desde un entorno web, presentando como característica fundamental, proporcionarle más rapidez al proceso de preparación de la cartografía para los SIG desarrollados en el departamento Geoinformática y Señales Digitales (GEYSED).
- F. AL PROCESO DESARROLLO SCRUM and SPADA, D., 2011. *Usabilidad en el proceso de desarrollo de SCRUM.* 5 January 2011. S.l.: s.n.
- FERNÁNDEZ, Noelia Méndez and VALIENTE, Jose Luis García, 2005. Programación con swing. In: [online]. May 2005. [Accessed 21 April 2012]. Available from: http://www.google.com.cu/#hl=es&q=programaci%C3%B3n+con+swing&oq=programaci%C3%B3n+con+swing&aq=f&aqi=g-K2&aql=1&gs_l=serp.3..0i30l2.41634.48754.4.48869.22.21.0.1.1.1.708.4576.6j5j2j2j2j1j1.19.0.gsnos%2Cn%3D10.1.0.0.gNjZLuiWTNk&bav=on.2,or.r_gc.r_pw.,cf.osb&fp=ac2ee27f0e773ba2&biw=1280&bih=629.
- GAMMA, E., HELM, R., JOHNSON, R. and VLISSIDES, J., 1995. *Design patterns: Elements of reusable object-oriented design*. S.l.: Addison-Wesley Reading, MA;
- GARCÍA, María N. Moreno, QUINTALES, Luis A. Miguel, PEÑALVO, Francisco J. García and MARTÍN, M. José Polo, 2010. Obtención y Validación de Modelos de Estimación de Software Mediante Técnicas de Minería de Datos. In: *Revista Colombiana de Computación*. 2010. Vol. 3, no. 1, pp. 53–71.
La medición del software está adquiriendo una gran importancia debido a que cada vez se hace más patente la necesidad de obtener datos objetivos que permitan evaluar, predecir y mejorar la calidad del software así como el tiempo y coste de desarrollo del mismo. El valor de las mediciones aumenta cuando se realiza sobre modelos construidos en las primeras fases del proyecto ya que los resultados obtenidos permiten tenerlo bajo control en todo momento y corregir a tiempo posibles desviaciones. La proliferación

actual de métricas y el gran volumen de datos que se maneja ha puesto de manifiesto que las técnicas clásicas de análisis de datos son insuficientes para lograr los objetivos perseguidos. En este trabajo se presenta la forma en que pueden aplicarse las nuevas técnicas de minería de datos en la construcción y validación de modelos de ingeniería del software, cambiando el análisis tradicional de datos dirigido a la verificación por un enfoque de análisis de datos dirigido al descubrimiento del conocimiento.

- GERACI, A., KATKI, F., MCMONEGAL, L., MEYER, B., LANE, J., WILSON, P., RADATZ, J., YEE, M., PORTEOUS, H. and SPRINGSTEEL, F., 1991. IEEE Standard Computer Dictionary: Compilation of IEEE Standard Computer Glossaries. In: 1991.
- GRUPO DE INGENIERÍA DEL SOFTWARE Y SISTEMAS DE INFORMACIÓN (ISSI), TORRES, Patricio Letelier and LÓPEZ, Emilio A. Sánchez, 2003. *Metodologías Ágiles en el Desarrollo de Software.* 12 November 2003. S.l.: s.n.
- HAMMING, R. W, 1950. Error detecting and error correcting codes. In: *Bell System technical journal*. 1950. Vol. 29, no. 2, pp. 147–160.
- HAMMOND, J. and GOULDE, M., 2007. Rich Internet Apps move beyond the browser. In: *Forrester, June*. 2007.
- HERNÁNDEZ, Raúl, 2012. Lenguajes de Modelado | Utópica Informática. In: [online]. 13 November 2012. [Accessed 25 March 2012]. Available from: http://www.utopicainformatica.com/2010/12/lenguajes-de-modelado.html.
- IEEE, 1990. *IEEE Computer Dictionary - Compilation of IEEE Standard Computer Glossaries.* 1990. S.l.: s.n.
- JACOBSON, I., BOOCH, G. and RUMBAUGH, J., 2000. *El proceso unificado de desarrollo de software*. S.l.: Addison Wesley.
- KOBLUK, C. G, MARIÑO, S. I and VALESANI, M. E, 2003. Sistema de información multiusuario. Aplicación de la programación orientada a objetos en la Dirección Nacional de Migraciones. In: 2003.
- LARMAN, Craig, 1999. *UML y Patrones. Introducción al análisis y diseño orientado a objetos*. México: PRENTICE HALL. ISBN 970-17-0261-1.
- LARMAN, Craig, 2003. *UML y Patrones, Introducción al Análisis y Diseño Orientado a Objetos.* 2da. Prentice Hall: s.n. ISBN 8420534382.
- LARMAN, Craig, 2004. *UML y Patrones, Introducción al Análisis y Diseño Orientado a Objetos.* Prentice Hall: s.n. ISBN 8420534382.
- LEÓN, A. R. Sotolongo, RODRÍGUEZ, M. Gutierrez and VEITIA, B. Piñeiro, 2011. CRUD-PG. In: *Revista Cubana de Ciencias Informáticas.* 2011. Vol. 5, no. 1.
- LEVENSHTEIN, Vladimir I., 1966. Los códigos binarios con la corrección deleciones, inserciones y sustituciones de caracteres. In: 1966. Vol. 163, no. 4, pp. 845–848.
- LÓPEZ, Ángel, 1997. *JAVA la programación del futuro.* 1. Montevideo: Rosgal S.A. ISBN 987-9131-38-X.
- LÓPEZ, Javier Parapar, 2008. *Introducción al lenguaje de programación Java* [online]. 2008. S.l.: s.n. Available from: http://www.dc.fi.udc.es/~parapar/files/java2009.pdf. Universidad de Coruña. Departamento de computación. Tecnología de la Programación.
- MARTÍNEZ, M.C. Beatriz Beltrán, 2002. 2002. S.l.: s.n. Benemérita Universidad Autónoma de Puebla.
- MASUELLI, G. A, 2001. Del Renga al videoblog. In: 2001.
- MENTAT TECHNOLOGIES, 2009. PostgreSQL: DreamCoder for PostgreSQL ver 2.0 is now available. In: [online]. 27 May 2009. [Accessed 11 June 2012]. Available from: http://www.postgresql.org/about/news/1089/.
- MONGE, A. E and ELKAN, C., 1996. The field matching problem: Algorithms and applications. In: *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96)*. S.l.: s.n. 1996. pp. 267–270.
- MORALES MEJÍA, J. A, GONZÁLEZ, S. and MARÍA, A., 2010. Plan de negocio para la creación de una empresa de servicio y asesoría de pruebas de software. In: 2010.
- OLIVARES, Freddy Egdamar Paez, 2009. diseño UML: diagrama de clases. In: [online]. 22 April 2009. [Accessed 14 April 2012]. Available from: http://egdamar877.blogspot.com/2009/05/expocicion.html.
- ORACLE CORPORATION AND/OR ITS AFFILIATES, 2012. NetBeans IDE 7.1.1 Release Information. In: [online]. 2012. [Accessed 5 March 2012]. Available from: http://netbeans.org/community/releases/71/.

- PETER and MENTAT TECHNOLOGIES DATABASE SOLUTIONS, 2012. Herramienta de base de datos para PostgreSQL: Software de administracion y desarrollo - DreamCoder for PostgreSQL - FREE Database Tools, SQL Developer, Database manager, Database IDE and SQL tools for MySQL, Oracle and PostgreSQL. In: [online]. 2012. [Accessed 10 June 2012]. Available from: http://www.sqldeveloper.net/herramientas-base-datos/postgresql/vision-general.html.
- PRESSMAN, R. S, 2002. *Libro: Ingeniería del software: un enfoque práctico.(5a. edicion)*. S.l.: McGraw Hill. Madrid. España, año.
- PRESSMAN, Roger S., 2005. *Ingeniería del Software: Un Enfoque Práctico*. 6ta. S.l.: McGraw-Hill. ISBN 9701054733.
- REYNOSO, Carlos Billy, 2004. Introducción a la Arquitectura de Software. In: *Documento de la Universidad de Buenos Aires. Descargado de http://www. willydev. net/descargas/prev/IntroArq. pdf*. 10 February 2004. Vol. 21, no. 01, pp. 245.
- ROSE, César E., 1993. *Archivos, Organización y procedimientos.* 1993. S.l.: s.n.
- SÁNCHEZ, Ing. María A. Mendoza, 2004. Metodologías De Desarrollo De Software. In: [online]. 7 April 2004. [Accessed 24 January 2012]. Available from: http://www.informatizate.net.
- SOMMERVILLE, Ian, 2005. *Ingeniería del Software*. 7ma. Madrid: Pearson Educación, S.A. ISBN 84-7829-074-5.
- SOTO, W. and PINZÓN, Y., 2010. Sobre el Longest Common Subsequence: Extensiones y Algoritmos. In: *Revista Colombiana de Computación-RCC*. 2010. Vol. 8, no. 2.
- SQL MAESTRO GROUP, 2012a. Descargar PostgreSQL Data Sync 12.2.0.2 versión de prueba gratis - Herramienta fácil de usar para la comparación y sincronización de contenidos de bases de datos - Softpedia en español. In: [online]. 9 April 2012. [Accessed 10 June 2012]. Available from: http://www.softpedia.es/programa-PostgreSQL-Data-Sync-198865.html.
- SQL MAESTRO GROUP, 2012b. Página del autor - SQL Maestro Group - Softpedia en español. In: [online]. 9 April 2012. [Accessed 11 June 2012]. Available from: http://www.softpedia.es/autor-SQL-Maestro-Group-6790.html.
- STONEBRAKER, M. and ROWE, L. A., 1986. *The design of POSTGRES.* S.l.: s.n. ISBN 0897911911. ACM
- SUMATHI, S. and ESAKKIRAJAN, S., 2007. *Fundamentals of Relational Database Management Systems*. Poland: Polish Academy of Sciences. ISBN 3-540-48397-7. Springer Berlin Heidelberg New York.
- SZYPERSKI, Clemens, 1998. *Software de los componentes - Más allá de Programación Orientada a Objetos*. 1era. Nueva York: Addison-Wesley / ACM Press. ISBN 0-201-17888-5.
- THE POSTGRESQL GLOBAL DEVELOPMENT GROUP, 1996a. *PostgreSQL 8.1.0 Documentation.* 1996. S.l.: s.n.
- THE POSTGRESQL GLOBAL DEVELOPMENT GROUP, 1996b. *PostgreSQL 8.1.0 Documentation.* 1996. S.l.: s.n.
- TROWBRIDGE, David, MANCINI, Dave, QUICK, Dave, HOHPE, Gregor, NEWKIRK, James and LAVIGNE, David, 2003. *Enterprise Solution Patterns using Microsoft .NET*. S.l.: Microsoft Corporation.
- VARELA, Feliciana, 2011. On Demand Business Suites Vs Office aplicaciones de escritorio. In: [online]. 2011. [Accessed 24 February 2012]. Available from: http://www.literatura-pretaporter.com/on-demand-business-suites-vs-office-aplicaciones-de-escritorio.html.
- VÁZQUEZ, Cañas, 2009. Sun Microsystems comprada por Oracle | Vázquez Cañas. In: [online]. 20 March 2009. [Accessed 11 April 2012]. Available from: http://www.vazquezcanas.es/2009/04/sun-microsystems-comprada-por-oracle/.
- VELASCO SÁNCHEZ, C., 2010. Reenginyeria inversa d'aplicacions J2EE. In: 2010.
- WEBCAB COMPONENTS COMPANY, [no date]. Component Based Development. In: [online]. [Accessed 11 June 2012]. Available from: http://webcabcomponents.com/componentization.shtml.
- WIECK, Jan, 2002. PL/pgSQL. In: [online]. 27 March 2002. Available from: http://www.ibiblio.org/pub/linux/docs/LuCaS/Postgresql-es/web/navegable/programmer/x1503.html.
- WINKLER, W. E, 1999. The state of record linkage and current research problems. In: *Statistical Research Division, US Census Bureau*. S.l.: Citeseer. 1999.
- YEE, Cristina Manresa, 2005. Informática Aplicada. In: [online]. 2005. Available from: http://dmi.uib.es/~cmanresay/04-BasesDatos.pdf.

Glosario de términos

A

Actividad:

El estado en que se exhibe algún comportamiento (Jacobson, Booch, Rumbaugh 2000).

Actores del Sistema:

Serán los trabajadores del Negocio, que se beneficiarán directamente con el Sistema ya implementado.

Analista:

Conduce y desarrolla la Ingeniería de Requisitos y la modelación de CU perfilando la funcionalidad y el límite del Sistema.

Artefactos de Software:

Pieza de información tangible que (1) es creada, modificada y usada por los trabajadores al realizar actividades; (2) representa un área de responsabilidad y (3) es candidata a ser tenida en cuenta para el control de la configuración. Un artefacto puede ser un modelo, un elemento de un modelo o un documento (Jacobson, Booch, Rumbaugh 2000).

C

Caso de Uso:

Es un fragmento de funcionalidad del Sistema que proporciona al usuario un resultado importante, representan los Requisitos Funcionales.

CSS3:

Hojas de estilo en cascada en su versión tres.

Cookie:

Es la información que guarda un servidor sobre un usuario en su equipo.

Copias de respaldo:

Las copias de respaldo son un duplicado o copia del dato original que va a existir, independiente a este, manteniendo su propia estructura (ANON. 2010 b).

CORBA:

Common Object Request Broker Architecture (Arquitectura común de intermediarios en peticiones a objetos). Es un estándar que establece una plataforma de desarrollo de Sistemas distribuidos, facilitando la invocación de métodos remotos bajo un paradigma OO.

C++:

Lenguaje de Programación OO, diseñado a mediados de los años 1980, por Bjarne Stroustrup como extensión del lenguaje de programación C.

C#:

Es un lenguaje de programación OO desarrollado y estandarizado por Microsoft como parte de su plataforma .NET. Su sintaxis básica deriva de C/C++ y utiliza el modelo de objetos de la plataforma.NET el cual es similar al de Java aunque incluye mejoras derivadas de otros lenguajes.

D

Diagrama:

La presentación grafica de un conjunto de elementos o estereotipos, usualmente representado como un grafo conectado de vértices (elementos) y arcos (relaciones) (Jacobson, Booch, Rumbaugh 2000).

Diagrama de Procesos de Negocio:

Diagrama diseñado para ser usado por las personas que diseñan y administran procesos de negocio, muestra el flujo de las actividades que se desarrollan en el negocio en cual está enmarcado el problema.

Diagrama de Casos de Uso:

Diagramas que muestran un conjunto de CU y de Actores y sus relaciones; los diagramas de Casos de Uso muestran los CU de un sistema desde un punto de vista estático (Jacobson, Booch, Rumbaugh 2000).

DLL:

Dynamic Link Library o Dynamic-Link Library (Bibliotecas de Enlace Dinámico), término con el que se refiere a los archivos con código ejecutable que se cargan bajo demanda del programa por parte del SO.

F

Fase:

Período de tiempo entre dos hitos principales de un proceso de desarrollo (Jacobson, Booch, Rumbaugh 2000).

Framework:

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

Freeware:

Se define como Freeware todo aquel programa que se distribuya gratuitamente, con ningún coste adicional. Uno de los grandes ejemplos es la *suite* de navegador y cliente de correo y noticias de Mozilla, distribuido también bajo licencia GPL (Software Libre).

H

HTML:

HyperText Markup Language o Lenguaje de Marcado de Hipertexto, es el lenguaje de marcado predominante para la elaboración de páginas web.

HTTP:

Es un protocolo de transferencia de hipertexto.

I

IEEE:

Instituto de Ingenieros Eléctricos y Electrónicos, fundado en 1963, IEEE es una asociación técnico-profesional mundial integrada por ingenieros, científicos y estudiantes, dedicada a la estandarización, entre otras cosas. Es la mayor asociación internacional sin fines de lucro formada por profesionales de las nuevas tecnologías, como ingenieros eléctricos, ingenieros en electrónica, científicos de la computación, ingenieros en informática e ingenieros en telecomunicación.

IMAP:

Es un protocolo de red de acceso a mensajes electrónicos almacenados en un servidor.

Ingeniería de Software:

La aplicación de un enfoque sistemático, disciplinado y cuantificable hacia el desarrollo, operación y mantenimiento del Software; es decir la aplicación de Ingeniería al Software. Es una tecnología multicapa (Roger S. Pressman 2005).

J

JavaEE:

Java Platform, Enterprise Edition o Java EE (anteriormente conocido como Java 2 Platform, Enterprise Edition o J2EE hasta la versión 1.4), es una plataforma de programación (parte de la Plataforma Java) para desarrollar y ejecutar software de aplicaciones en el lenguaje de programación Java con arquitectura de N capas distribuidas y que se apoya ampliamente en componentes de software modulares ejecutándose sobre un servidor de aplicaciones (Velasco Sánchez 2010).

JavaFX:

Es una familia de productos y tecnologías de Sun Microsystems, adquirida por Oracle Corporation, para la creación de Rich Internet Applications (RIAs), esto es, aplicaciones web que tienen las características y capacidades de aplicaciones de escritorio, incluyendo aplicaciones multimedia interactivas. Las tecnologías incluidas bajo la denominación JavaFX son JavaFX Script y JavaFX Mobile, aunque hay más productos JavaFX planeados (Hammond, Goulde 2007).

M

Maven:

Es una herramienta de automatización para construir en la gestión de proyectos Java (Böck 2011).

Modelos:

Es una descripción de (parte de) un Sistema, descrito en un lenguaje bien definido. Un lenguaje bien definido es un lenguaje con una sintaxis y semántica precisa y que es interpretado y manipulado por un ordenador.

N

Negocio:

Cualquier ambiente o entorno en cual está enmarcado el problema.

.NET:

Plataforma de desarrollo de Software creada por Microsoft, con énfasis en transparencia de redes, con independencia de plataforma y permite un rápido desarrollo de aplicaciones.

NNTP:

Protocolo para la Transferencia de Noticias en Red es un protocolo creado para la lectura y publicación de artículos de noticias en la red de usuarios.

Normalización:

El proceso de normalización de BD consiste en aplicar una serie de reglas a las relaciones obtenidas tras el paso del modelo entidad-relación al modelo relacional.

Las BD relacionales se normalizan para evitar la redundancia de los datos, evitar problemas de actualización de los datos en las tablas y proteger la integridad de los datos.

O

OMG:

Object Management Group (Grupo de Gestión de Objetos), es una asociación fundada en 1989 sin fines de lucro formada por grandes corporaciones, muchas de ellas de la industria del software, como IBM, Apple, Sun Microsystems y HP entre otros. Este grupo gestiona los estándares relacionados con la tecnología OO.

Oracle:

Sistema de gestión de bases de datos relacional (o RDBMS por el acrónimo en inglés de Relational Data Base Management System), desarrollado por Oracle Corporation.

P

Patrón:

Solución común a un problema común de un determinado contexto. Un patrón es una pareja de problema / solución con un nombre, que codifica (estandariza) buenos principios y sugerencias relacionados frecuentemente con la asignación de responsabilidades (Larman 2004).

POP3:

Es un protocolo de correo.

Proceso:

Conjunto de operaciones, acciones, cambios o funciones que se realizan para alcanzar un fin.

Proceso de Negocio:

Conjunto total de actividades necesarias para producir un resultado de valor percibido y medible para un cliente individual de un negocio (Jacobson, Booch, Rumbaugh 2000).

Proceso de Desarrollo de Software:

Proceso de Negocio o Caso de Uso de Negocio, de un Negocio de Desarrollo de Software. Conjunto total de actividades necesarias para transformar los Requisitos de un cliente en un conjunto consistente

de artefactos que representan un producto Software y, posteriormente, para transformar cambios en dichos Requisitos en nuevas versiones del producto Software (Jacobson, Booch, Rumbaugh 2000).

Protocolo:

Es un conjunto de reglas usadas por computadoras para comunicarse unas con otras a través de una red por medio de intercambio de mensajes.

R

Requisito:

1. Una condición o capacidad necesaria para un usuario para resolver un problema o alcanzar un objetivo.
2. Una condición o capacidad que debe ser alcanzada o poseída por un sistema o componente de un sistema para satisfacer un contrato, estándar, especificación u otro documento formalmente impuesto.
3. Una representación documentada de una condición o capacidad dada en los puntos 1 ó 2 (Geraci, Katki, McMonegal, Meyer, Lane, P. Wilson, Radatz, M. Yee, Porteous, Springsteel 1991).

Rol:

Papel que desempeña una persona en un determinado momento; una misma persona puede desempeñar distintos roles a lo largo del proceso. Comportamiento especifico de una entidad que participa en un contexto particular (Jacobson, Booch, Rumbaugh 2000).

S

Scripts:

Un guión o *script* es un fichero de texto que contiene una serie de instrucciones que se pueden ejecutar en la línea de órdenes, y que se ejecutarán seguidas. El único requisito es que ese fichero de texto tenga permiso de ejecución para la persona que intenta ejecutarlo. Alternativamente, se puede llamar al intérprete y darle como parámetro el nombre del guión (ANON. 2004 b).

Sockets:

Los sockets de Internet constituyen el mecanismo para la entrega de paquetes de datos provenientes de la tarjeta de red a los procesos o hilos apropiados.

Shareware:

El Shareware es otra modalidad de comercialización, el programa se distribuye con limitaciones, bien como versión de demostración o evaluación, con funciones o características limitadas o con un uso restringido a un límite de tiempo establecido (por ejemplo 30 días). Así, se le da al usuario la oportunidad de probar el producto antes de comprarlo y, más tarde, adquirir la versión completa del programa.

SNMP:

Protocolo Simple de Administración de Red es un protocolo de la capa de aplicación que facilita el intercambio de información de administración entre dispositivos de red.

Software:

Es la suma total de los programas de cómputo, procedimientos, reglas, documentación y datos asociados que forman parte de las operaciones de un sistema de cómputo y que “un producto de Software es un producto diseñado para un usuario” (Geraci, Katki, McMonegal, Meyer, Lane, P. Wilson, Radatz, M. Yee, Porteous, Springsteel 1991).

Software Libre:

El Software Libre no tiene por qué ser gratuito. De hecho su denominación de Libre se debe a que se tratan de programas de Código Abierto (Open Source) y es ahí donde reside la esencia de su *libertad*: los programas bajo licencias GPL, una vez adquiridos, pueden ser usados, copiados, modificados y redistribuidos libremente.

Stakeholders:

Son participantes, pueden implicar a usuarios finales, encargados, ingenieros implicados en el mantenimiento, expertos del dominio, entre otros.

Swing:

Librería que ofrece el desarrollo de interfaces gráficas de usuario (Fernández, Valiente 2005).

T

Trabajadores:

Son las personas o entes involucrados en cada proceso (Sánchez 2004).

X

XML:

Extensible Markup Language o lenguaje de marcas extensible, es un metalenguaje extensible de etiquetas desarrollado por el World Wide Web Consortium.