UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

FACULTAD 1

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

Actualización de la herramienta *Schema Tool* para el soporte y configuración de sistemas distribuidos en nuevos entornos

Autores: Alexander López Valladares

Ernesto Valdés Sotolongo

Tutores: Msc. Susana María Ramírez Brey

Ing. Vismar Fernández Santana

La Habana, 2015

"Año 57 de la Revolución"

“Siempre es el momento adecuado para hacer lo correcto”

Declaración de autoría

Declaramos ser los únicos autores del presente trabajo de diploma titulado “Actualización de la herramienta *Schema Tool* para el soporte y configuración de sistemas distribuidos en nuevos entornos” el cual fue elaborado durante el curso 2014-2015, además, conferimos los derechos patrimoniales sobre el mismo a la Universidad de las Ciencias Informáticas de manera exclusiva.

Finalmente declaramos que todo lo anteriormente expuesto se ajusta a la verdad, y asumimos la responsabilidad moral y jurídica que se derive de este juramento profesional. Autorizamos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste firmamos el presente a los ____ días del mes de ____ del año 2015.

_____ _____

Ernesto Valdés Sotolongo Alexander López Valladares

_____ _____

Ing. Vismar Fernández Santana Msc. Susana M. Ramírez Brey

Msc. Susana María Ramírez Brey

Graduada de Ingeniería en Ciencias Informáticas en el año 2009. Tiene categoría docente Instructor y es Máster en Informática Aplicada. Se desempeña actualmente como Especialista del Departamento de Componentes del Centro de Identificación y Seguridad Digital.

Correo electrónico: smramirez@uci.cu

Ing. Vismar Fernández Santana

Graduado de Ingeniero en Ciencias Informáticas en el año 2010. Tiene categoría docente Instructor y se desempeña en la actualidad como Jefe del Departamento de Componentes del Centro de Identificación y Seguridad Digital.

Correo electrónico: vsantana@uci.cu

Dedicatoria

A la memoria de mi abuela Delia, quien me crio y me formó como persona. Donde quiera que estés te prometí que me iba a graduar como ingeniero. No hablo más de ti porque no existen palabras para expresar lo que fuiste para mí.

A mi madre Odette por ser la luz de mi camino y la expresión más arraigada de amor, esperanza, esfuerzo y sacrificio.

A mis dos padres de sangre: Ernesto Valdés Capellán y Luis Bringas Guerra.

Ernesto

Quiero dedicarles esta tesis a mis padres, pues siempre han sido mi apoyo y guía desde pequeño, porque su sueño de verme graduado, de convertirme profesional siempre ha sido mi fuerza impulsora, por exigirme siempre los mejores resultados, incitándome al esfuerzo y la dedicación.

Alexander

Agradecimientos

A Alexander, mi compañero de tesis: gracias a ti y a los capitanes del barco mantuvimos esto a flote, a veces parecía que se hundía pero juntos vencimos los obstáculos.

Alisbet: Gracias por estar a mi lado, por ser mi bastón y mi mejor y más grandioso amor. En noches de agobio tan solo con pensar en tu sonrisa todas las mareas se calmaban.

Vismar: No encontré la forma de escribir el apartado tuyo, mi hermano, porque si me pongo a redactarlo ni George R.R. Martin ni Tolkien iban a poder escribir más en sus vidas, aunque si pude hacer un resumen de todo. En ti se juntan todas las cosas buenas de una persona: alegría, amistad, bondad, laboriosidad y lo más preciado que tienes, la inteligencia. Gracias por ser, más que mi mejor amigo, mi hermano.

Susana: Sin tu aporte a este trabajo sinceramente creo que no nos hubiéramos graduado. Eres la mejor en todo lo que haces y dudo que alguien te sobrepase en revisión de documentos científicos. Antes de tener la tesis en mis manos siempre soñé con tener una tutora tan dedicada como tú.

Joel: Sabes que te agradezco con la vida todo lo que hiciste para que se cumpliera este sueño. Eres un ejemplo a seguir. En esta universidad nadie sabe el error que se cometió al dejarte ir.

De manera general gracias a todos mis amigos que siempre estuvieron conmigo, lo mismo en las noches de estudio, que en el estrés del proyecto o el "cuero" en la cola del comedor: a Ander por existir y yo siempre queriendo ser tan inteligente como él; a Carlos y Osvaldo, mis dos "colegones", "ambias", "moninas", socios, amigos, hermanos, que siempre nos pusimos de acuerdo para todo y cito a Carlos: "Déjale lo de los algoritmos a Osvaldito que él es bueno en eso. Tu y yo le metemos a la teoría y después, si eso, cambiamos conocimientos". Mi respuesta fue: "Vamos a seguir inventando, ya hoy es jueves y mañana es la prueba". A mi abuela Isabel, por siempre estar ahí, a mi hermano Osmel, por alegrarme los fines de semana. Al piquete del doble en el comedor de profesores, a mi prima Izette, por tanta ayuda y consejos. A Betty, por haber sido durante mucho tiempo mí segunda mamá, a mis tíos y a mis primos por ser los "míos".

A todos los que se me quedan que de una forma u otra contribuyeron a mi formación como profesional.

Ernesto

No creo poder incluir a todas las personas que de una u otra forma me han ayudado a recorrer el camino en tan poco espacio. Tampoco pienso que con hacer mención a aquellos que tan importantes han sido para mí muestre una medida de lo que realmente quisiera expresar. Aun así, a todos ellos debo y quiero hacer llegar mis agradecimientos.

Agradezco:

A mi familia, por mostrarme el camino y ayudarme a recorrerlo, por apoyarme e inculcarme un grupo de valores sin los cuales hoy no sería quien soy; por ser mí ejemplo a seguir, mi inspiración y mi fuerza. Por cultivar mi carácter y cosechar mis logros, que mucho más que míos, son nuestros.

A mi madre, por su paciencia, su bondad y por guiarme constantemente. A mi padre, por todas sus lecciones y la sabiduría que en todos estos años me ha transmitido. A mi hermano por ser único y espléndido. A mi novia por su ayuda, comprensión y por compartir conmigo todos los momentos difíciles y felices de mi vida. A mis abuelos, por su amor y por ser mí fuerza para crecer. A mis padrino, por siempre estar ahí, por su apoyo, cariño y preocupación.

A todos los profesores que me ayudaron de alguna forma u otra en estos 5 años de carrera, de no haber sido por ellos, no hubiese podido hacer realidad este sueño.

A mis amigos de la UCI, por confiar en mí y darme la seguridad de confiar en ellos; por llenar cada uno de los recuerdos que me llevo; por sus consejos, sus charlas, por tantos ratos inolvidables, a ellos, agradezco.

A mi amigo y compañero de tesis, por ser alguien excepcional desde el primero de estos 5 años y hasta hoy; por hacer en esta última etapa menos compleja la tarea, agradezco.

A mis compañeros que tantos días hemos compartido, nada sería igual sin ustedes.

A mis tutores, por toda su ayuda y entrega, por el conocimiento que nos han transmitido, por guiarme en este difícil camino, por compartir su tiempo, agradezco.

A todos los que alguna vez me preguntaron: "–¿y la tesis…"?

Alexander

Resumen

La mayoría de las soluciones informáticas en la actualidad requieren el almacenamiento de información. Las bases de datos son un elemento predominante en lo que se refiere a técnicas y herramientas para este propósito. La herramienta *Schema Tool* fue creada con el fin de apoyar el proceso de desarrollo de software para el SAIME, ayudando a dar soporte a un sistema con múltiples bases de datos distribuidas. Su principal funcionalidad es la creación de esquemas para la replicación de información entre bases de datos con una estructura similar. Originalmente esta herramienta fue desarrollada sobre la versión 1.1 de la plataforma .NET e integrada únicamente al sistema gestor Oracle, factores que limitan su uso en sistemas operativos actuales y en entornos distribuidos que utilicen otros poderosos y populares sistemas gestores de bases de datos. Por esta razón la presente investigación propone ampliar el entorno de aplicación de la herramienta, integrándola a los principales sistemas gestores de bases de datos relacionales existentes en la actualidad y dotándola de una arquitectura extensible que le permita su crecimiento en el futuro, todo ello a partir de una actualización de su marco de trabajo a la última versión disponible de la plataforma .NET.

El documento expone los resultados de la investigación, describiéndose las principales características de los sistemas analizados. En el mismo se explica la arquitectura y el diseño de la solución propuesta, se describen las herramientas y tecnologías que se utilizan, así como los artefactos que se generan en el proceso de desarrollo guiado por la metodología XP.

Palabras clave: bases de datos distribuidas, replicación, sistemas gestores de bases de datos.

Índice de contenido

Índice de figuras

Índice de tablas

Introducción

Durante los últimos años se ha registrado un incremento considerable a nivel global de la influencia de la informática en los diferentes procesos de la sociedad. Instituciones gubernamentales, entidades empresariales, sectores educativos, sectores de salud, la telefonía móvil o sencillamente el ocio, por solo mencionar algunos, son campos donde el desarrollo constante de nuevas aplicaciones y propuestas informáticas va en índice creciente, modificando procesos y conceptos que hasta entonces se concebían de otra forma.

Cualquier aplicación o sistema informático, independientemente de cuál sea su propósito u objeto social, puede requerir el almacenamiento de información básica para su funcionamiento y configuración. Incluso en un gran número de casos, el fácil manejo de la información relativa a una entidad es precisamente la razón de ser del sistema. Con la evolución de la informática se ha transcurrido un largo camino en lo que respecta a métodos, técnicas y herramientas para el almacenamiento de la información, siendo las bases de datos (BD) un elemento predominante en la actualidad.

Un sistema de bases de datos es un conjunto de datos interrelacionados entre sí, sin redundancias innecesarias e independiente de los programas que acceden a ellos, los cuales son gestionados por diferentes sistemas gestores de bases de datos (SGBD) [1]. El objetivo fundamental de un SGBD consiste en suministrar al usuario las herramientas que le permitan manejar en términos abstractos los datos, sin la necesidad de conocer la forma en que estos son almacenados en los sistemas de información, ni el método de acceso empleado [2]. La evolución de esta tecnología en el tiempo ha dado surgimiento a SGBD relacionales muy confiables e intuitivos de utilizar como PostgreSQL, MySQL, MongoDB, Microsoft SQL Server y Oracle.

Una base de datos distribuida (BDD) consiste en una colección de sitios, conectados por medio de algún tipo de red de comunicación, donde los sitios trabajan en conjunto a fin de que un usuario de cualquier sitio pueda acceder a los datos desde cualquier lugar de la red. Al trabajar con BDD, se hace necesario consultar y obtener información en distintos sentidos, ya que inicialmente la información solicitada puede no encontrarse en un servidor local y sea necesario que se consulte en otro servidor [3]. Para garantizar la disponibilidad de los datos en bases de datos distribuidas es preciso mencionar el término replicación. La

replicación puede definirse como el transporte de datos entre dos o más servidores, permitiendo que cierta información esté almacenada en más de un sitio, y así aumentar la disponibilidad y mejorar el rendimiento de las consultas globales [4]. Es utilizada en BDD para garantizar la sincronización de la información, existiendo una estrecha relación entre ambos conceptos.

La herramienta *Schema Tool* forma parte de un conjunto de aplicaciones creadas con el objetivo de apoyar el proceso de desarrollo de software para el Servicio Administrativo de Identificación, Migración y Extranjería (SAIME) de la República Bolivariana de Venezuela. En el año 2005 se comenzó a concebir todo el proyecto de automatización de los procesos de esta importante entidad venezolana. Dicho proyecto incluía la creación de un gran número de oficinas regionales a nivel nacional, donde los ciudadanos pudieran realizar sus trámites. De igual manera la información debía ser accesible y modificable desde aeropuertos, puntos fronterizos, sedes consulares en el exterior, un centro de impresión de documentos de identificación y una sede central encargada de rectorar los principales procesos de la entidad. Ante un sistema distribuido de tal magnitud, una de las principales preocupaciones del equipo de desarrollo fue garantizar la disponibilidad e integridad de la información en todos los sitios del mismo.

Se decidió entonces la creación de un centro de datos a nivel nacional ubicado en el Distrito Capital y un servidor local por cada oficina regional. El centro de impresión de documentos, los puntos fronterizos, las sedes consulares y los aeropuertos (a excepción del Aeropuerto Internacional de Maiquetía) se comunicarían directamente con el centro de datos nacional. Ante la disyuntiva de qué información almacenar, se decidió colocar en las bases de datos de las oficinas regionales del sistema SAIME solamente la información imprescindible para su funcionamiento, básicamente elementos de configuración. Los datos relativos a los ciudadanos, sus documentos legales, sus trámites, etc., se sincronizarían en tiempo real una vez que un ciudadano decidiera realizar algún trámite en determinada oficina.

De esta manera se garantizó tener en las bases de datos locales solo un mínimo de información, evitando un uso indiscriminado de los recursos de los servidores de oficina. Como ningún sistema puede predecir cuando un ciudadano irá a tramitar de manera espontánea en determinada oficina, esta réplica no podía efectuarse de manera periódica como una tarea programada. Tampoco era viable que la sincronización fuese llevada a cabo manualmente por un equipo de soporte o trabajadores de la entidad. Se decidió que

las aplicaciones del sistema SAIME, efectuaran de manera transparente al usuario la sincronización de la información requerida en cada momento. Con ese propósito surge la herramienta *Schema Tool*.

Las principales funciones de esta herramienta consisten en la creación de esquemas de datos que permitan, mediante la especificación de filtros, la comparación entre fuentes de datos similares, así como la ejecución de los mismos de acuerdo a una planificación previa, lo cual resulta de vital importancia en sistemas distribuidos. Los ficheros generados por la herramienta *Schema Tool*, especifican qué información, con qué filtros y en qué orden debe ser sincronizada hacia determinada base de datos. Estos ficheros son usados por la aplicación de oficinas regionales del sistema SAIME cada vez que un ciudadano desea realizar un trámite. La diversidad de negocios o procesos automatizados en el sistema SAIME, hacen vital la distinción de la información requerida para un trámite determinado. No serán requeridos los mismos datos para un ciudadano que desea solicitar su cédula de identidad, que para aquel que desea solicitar su pasaporte, una visa o un documento con información filiatoria. Por tal motivo, cada vez que se incorporan nuevas funcionalidades o procesos a las aplicaciones existentes, un paso indispensable en dicho proceso de desarrollo es la creación de esquemas de datos que respondan a los mismos.

A pesar de no constituir una solicitud explícita de la entidad venezolana, *Schema Tool* se convirtió en una herramienta primordial en el desarrollo del sistema SAIME. Esta herramienta no solo permite la definición de nuevos esquemas y planificaciones de ejecución de cambios, sino que posibilita además comparar y homogeneizar dos fuentes de datos dadas, mediante la aplicación de operaciones dirigidas a eliminar las diferencias detectadas en la comparación. De ahí que *Schema Tool* también sea ampliamente usada en tareas de soporte. La variedad de situaciones en las que puede ser empleada, la hacen muy útil para cualquier desarrollador de aplicaciones o especialista de soporte de bases de datos. Por esta razón *Schema Tool*, aunque surge a propósito del desarrollo del sistema SAIME, no está atado a él y puede ser utilizado como un producto genérico para el desarrollo y soporte de cualquier sistema con bases de datos distribuidas.

Actualmente la herramienta se encuentra desarrollada bajo la versión 1.1 del *framework* .NET, lo que limita su utilización en sistemas operativos superiores a Windows XP. Por otro lado solo permite la integración con el sistema gestor de bases de datos Oracle, lo que trae como consecuencia que no pueda

ser utilizada como producto genérico de apoyo a los procesos de soporte y configuración en sistemas distribuidos que utilicen otros SGBD.

En el año 2014 el SAIME solicitó la actualización y mejora de algunas de las aplicaciones que componen su sistema, debido a problemas de compatibilidad con versiones superiores de la plataforma .NET 1.1 y el soporte necesario para otros sistemas operativos. El Centro de Identificación y Seguridad Digital (CISED) de la Universidad de Ciencias Informáticas (UCI), principal responsable de estudiar la factibilidad de acceder a dicha petición, ha decidido incorporar la herramienta *Schema Tool* a la lista de aplicaciones a actualizar, aprovechando además este marco para hacerla extensible y posibilitar así su integración con algunos de los principales sistemas gestores de bases de datos existentes en la actualidad.

Teniendo en cuenta los elementos anteriormente planteados se puede identificar el siguiente **problema de investigación:** ¿Cómo ampliar el entorno de aplicación de la herramienta *Schema Tool* para el apoyo al soporte y configuración de sistemas distribuidos?

Para dar solución al problema planteado se define como **objeto de estudio**: Sistemas de bases de datos distribuidas y se delimita como **campo de acción:** El proceso de sincronización de datos en sistemas de bases de datos distribuidas.

El **objetivo general** de la investigación es: Actualizar la herramienta *Schema Tool* utilizando la versión 4.5 del *framework* .NET e integrándola con los principales sistemas gestores de bases de datos relacionales existentes en la actualidad, de manera que sirva de apoyo al soporte y configuración de sistemas distribuidos en nuevos entornos.

A partir del objetivo general, se derivan los **objetivos específicos** que se enuncian a continuación:

1. Elaborar el marco teórico de la investigación.

2. Rediseñar la herramienta *Schema Tool* para permitir la integración de manera extensible con otros sistemas gestores de bases de datos relacionales.

3. Implementar la capa de acceso a datos para integrar la herramienta *Schema Tool* con los principales sistemas gestores de bases de datos existentes.

4. Realizar pruebas de software a la herramienta para validar las funcionalidades que permiten la replicación entre los sistemas gestores de bases de datos relacionales Oracle, PostgreSQL y Microsoft SQL Server.

Para garantizar el cumplimiento de estos objetivos se definen las siguientes **tareas de investigación:**

1. Análisis de la herramienta *Schema Tool*, sus funcionalidades, diseño, arquitectura y limitaciones.
2. Caracterización de los sistemas gestores de bases de datos a integrar con el *Schema Tool* (Oracle, PostgreSQL, Microsoft SQL Server).
3. Análisis de las herramientas existentes en el mercado para la configuración y replicación de BD.
4. Identificación de las mejoras a realizar en la actualización de la herramienta *Schema Tool*.
5. Descripción de la tecnología de acceso a datos, herramientas y metodología, seleccionadas para el desarrollo.
6. Migración del *Schema Tool* a la versión 4.5 del *framework*. NET.
7. Diseño de la capa de acceso a datos garantizando que permita la integración de manera extensible con otros sistemas gestores de bases de datos.
8. Implementación de la capa de acceso a datos, para la integración con Oracle, PostgreSQL y Microsoft SQL Server, según la tecnología de acceso a datos seleccionada.
9. Ejecución de las pruebas de unidad, aceptación y rendimiento, con diferentes sistemas gestores de bases de datos en diferentes entornos, para la validación del correcto funcionamiento de la herramienta.

La solución se sustenta en los siguientes **métodos de investigación:**

Analítico-Sintético: Fue utilizado para el análisis e interpretación de los conceptos más importantes sobre la replicación de datos, así como las herramientas existentes en las cuales la investigación va a sustentarse, para así determinar los requisitos esenciales para la actualización del sistema.

Histórico-Lógico: Fue utilizado para el estudio de los antecedentes, la evolución y el desarrollo que ha tenido la replicación de datos, permitiendo tener un mayor conocimiento para aplicarlo en la solución del problema presentado y realizar una valoración de las tendencias actuales del proceso.

Modelado: Fue utilizado para modelar los múltiples diagramas en las etapas de análisis y diseño de la solución, posibilitando un mejor entendimiento del funcionamiento de la herramienta.

Justificación de la investigación:

Con la actualización del *Schema Tool* se cuenta con una nueva versión capaz de integrarse con algunos de los sistemas gestores de bases de datos relacionales más importantes y no solamente con Oracle. Además se renovó el marco de trabajo que ha quedado obsoleto ante las nuevas versiones del *framework* .NET y se mejoró la interfaz de usuario. De esta manera se garantizó que la herramienta se pudiera utilizar como producto genérico para el apoyo soporte y configuración de sistemas que utilizan BDD. Estos resultados posibilitarán el empleo del software no solo en el SAIME, sino también en cualquier sistema distribuido.

Estructura del documento

El documento está compuesto por tres capítulos que describen el trabajo realizado. Además se presentan las Conclusiones, Recomendaciones, Referencias bibliográficas y Bibliografía consultada de la investigación.

Capítulo 1: "Fundamentación Teórica"

En este capítulo se detallan los conceptos fundamentales relacionados con la temática a abordar, se especifica la metodología a utilizar así como la justificación de su empleo. Además se mencionan las herramientas y tecnologías que fueron empleadas en la actualización del sistema y se muestran los resultados del estudio de los sistemas similares.

Capítulo 2: "Análisis y diseño de la herramienta *Schema Tool*"

En este capítulo se presentan los aspectos fundamentales relacionados con el análisis y diseño de la herramienta, así como la descripción de los mismos según la metodología ágil de desarrollo XP. Se presentan la especificación de requisitos funcionales y no funcionales definidos como punto de partida para la actualización de la herramienta. Además se describen los patrones de diseño utilizados, así como otros artefactos asociados con esta etapa del desarrollo.

Capítulo 3: "Implementación y pruebas de la herramienta *Schema Tool*"

En este capítulo se describen los elementos asociados a la implementación y la validación de la nueva versión de la herramienta. Se presentan el diagrama de componentes y el de despliegue para mostrar la distribución física de los componentes en un ambiente real. Se muestran los resultados de las pruebas unitarias, de aceptación y rendimiento realizadas para verificar y validar el funcionamiento correcto de la herramienta.

Capítulo 1. Fundamentación teórica

Introducción

En este capítulo se analizan un conjunto de aspectos esenciales relacionados con el objeto de estudio y con el objetivo de la investigación. Se analizan sistemas existentes para la replicación de datos que guardan relación con el *Schema Tool* y se exponen sus principales características. Se presentan rasgos distintivos de la versión existente de la herramienta para un mejor entendimiento de su funcionamiento y se exponen las tecnologías que intervienen en el proceso de desarrollo de software. De esta manera se relacionan todos los conceptos que desde el punto de vista teórico fundamentan lo planteado en la situación problemática.

1.1 Bases de datos relacionales

Las bases de datos relacionales son aquellas que utilizan el modelo relacional para la representación de la información que se desea almacenar. Este modelo está basado en tres componentes esenciales: una colección de objetos o relaciones, operadores que actúen sobre dichos objetos o relaciones y métodos para la integridad de los datos.

El modelo relacional provee un lugar para almacenar los datos, una forma de crearlos y recuperarlos, y una manera de asegurarse que sean lógicamente consistentes [5]. Tanto los objetos o entidades, como las relaciones que se establecen entre ellos, se representan a través de tablas, que en la terminología relacional se denominan relaciones [2]. Cada relación tiene asociado un tipo de relación[1] (ver **Figura 1)**.

[1] “De uno a muchos” (1:M); “de muchos a muchos” (M:M); “de uno a uno” (1:1) [46].

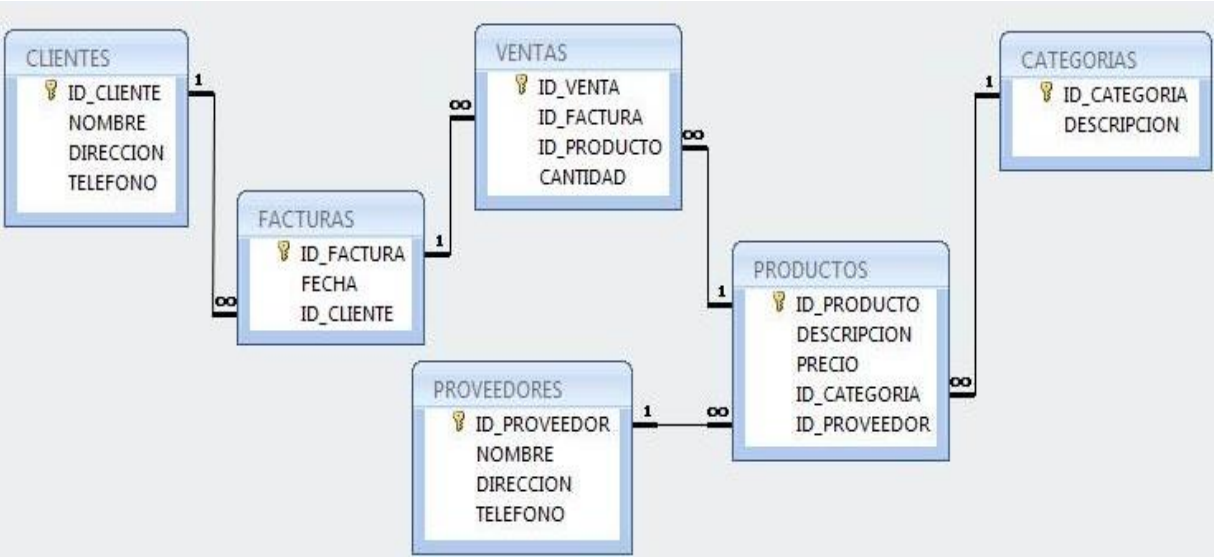

Figura 1. Esquema de una base de datos relacional.

1.2 Bases de datos distribuidas

Una base de datos distribuida es un conjunto de múltiples bases de datos lógicamente relacionadas que se encuentran distribuidas entre diferentes sitios interconectados por una red de comunicaciones. Tienen la capacidad de procesamiento autónomo lo cual indica que puede realizar operaciones locales o distribuidas [6].

Las bases de datos distribuidas tienen dos clasificaciones fundamentales de acuerdo a la forma en que se encuentran distribuidos los datos:

Homogéneas

Donde todos los sitios tienen idéntico SGBD, son conscientes de la existencia de los demás sitios y acuerdan cooperar en el procesamiento de las solicitudes de los usuarios. Además los sitios locales renuncian a una parte de su autonomía en cuanto a su derecho a modificar los esquemas o el software del SGBD [7].

Heterogéneas

Donde sitios diferentes puede que utilicen esquemas diferentes y diferentes SGBD. Algunos de los sitios puede que no sean conscientes de la existencia de otros y que solo proporcionen facilidades limitadas para la cooperación en el procesamiento de las transacciones [7].

Una BDD tiene cuatro ventajas fundamentales:

- Cada grupo o persona que posee un equipo, tendrá control directo sobre sus datos locales, permitiendo una mayor integridad y un procesamiento más eficiente sobre la información.
- Reducción de la sobrecarga de comunicación en comparación con el enfoque centralizado.
- Brinda una solución natural al procesamiento de datos en organizaciones geográficamente dispersas.
- El rendimiento y la fiabilidad pueden ser incrementados mediante la explotación de las capacidades de procesamiento y la redundancia en paralelo de múltiples máquinas [8].

1.3 Replicación de datos

La replicación de datos es el proceso de copiar y mantener objetos de una BD en diferentes bases de datos (ver **Figura 2**), que constituyen un sistema de bases de datos distribuido. Durante este proceso se capturan los cambios aplicados a una BD y luego se aplican a cada una de los demás de acuerdo a ciertas reglas de replicación definidas [9].

Figura 2. Replicación de datos.

Existen tres tipos de clasificaciones para la replicación de datos:

De acuerdo al ambiente de replicación, el proceso de sincronización puede ser maestro-esclavo, donde la replicación se realiza en un solo sentido desde un nodo maestro hacia uno o varios esclavos, o multi-maestro, donde se configuran nodos como maestros para replicar entre sí (ver **Figuras 3 y 4**).

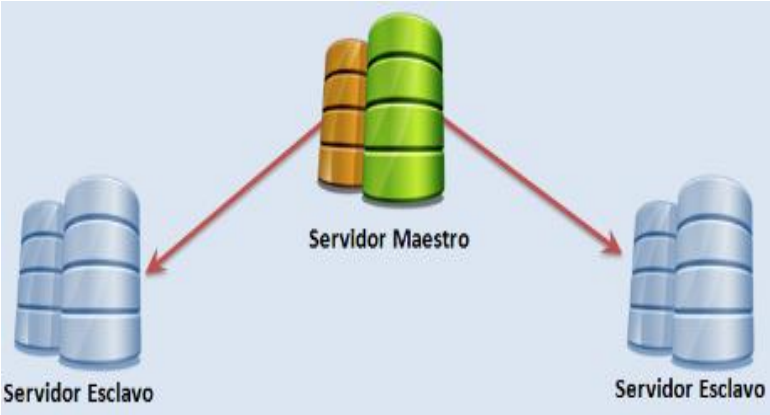

Figura 3. Sincronización de tipo maestro-esclavo

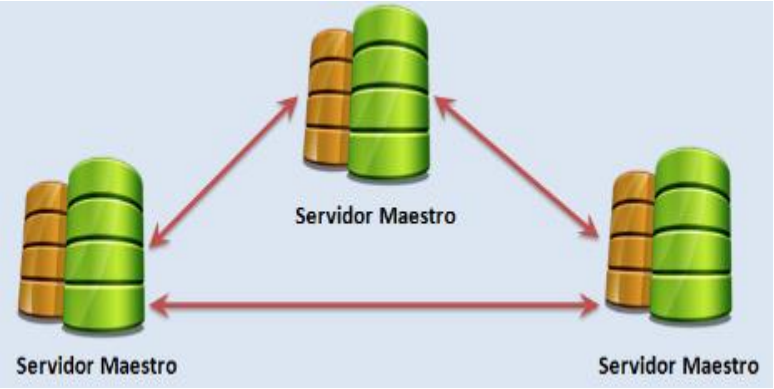

Figura 4. Sincronización de tipo multi-maestro.

De acuerdo a la forma de transmitir los cambios en el entorno de aplicación, la sincronización puede ser síncrona, donde los cambios ejecutados en un nodo maestro son aplicados instantáneamente en el nodo origen y los nodos destino dentro de una misma transacción, o asíncrona, donde los cambios ejecutados en una tabla son almacenados y enviados posteriormente al resto de los nodos del entorno, cada ciertos intervalos de tiempo.

Atendiendo a la forma de capturar y almacenar los cambios a replicar, el proceso de sincronización puede ser basado en *triggers*[2], donde se crean una serie de *triggers* en la BD que permiten capturar las operaciones de inserción, actualización y eliminación realizadas sobre las tablas a replicar, o basado en logs[3] cuyo funcionamiento se sustenta en la lectura de logs de cambios, que proporcionan algunos SGBD como Oracle [10].

Otros sistemas de replicación asincrónicos basan su funcionamiento en la comparación exhaustiva de fuentes de datos, efectuando una extracción de información desde el maestro y auxiliándose de las llaves primarias de cada tupla para detectar diferencias entre la propia extracción efectuada desde el esclavo. La ausencia de cada valor de llave primaria en el esclavo propiciará la inserción de una tupla; su presencia compartida con la del maestro propiciará una actualización y finalmente, en caso de que el valor exista solamente en el esclavo propiciará una eliminación, garantizándose así la homologación de ambas fuentes. Esta estrategia no es ideal para procesos de replicación sincrónicos pues son costosos de realizar, en cambio es muy práctica y eficiente para una replicación inicial, o para la homologación de fuentes de datos afectadas sin un control oportuno de sus versiones.

1.4 Sistemas gestores de bases de datos

Los sistemas gestores de bases de datos son aplicaciones que permiten a los usuarios definir, crear, mantener y proporcionar acceso controlado a las BD.

Un SGBD debe prestar los servicios que a continuación se mencionan: la definición y creación de una BD, la manipulación de los datos a través de consultas, las inserciones y actualizaciones, el acceso controlado a los datos mediante mecanismos de seguridad, el mantenimiento de la integridad y el control de concurrencia. Además debe proporcionar mecanismos de copias de respaldo y recuperación para reestablecer la información en caso de fallos en el sistema [11].

A pesar de que existe un gran número de SGBD en la actualidad, algunos son recurrentes en la revisión de la literatura sobre el tema, dígase artículos de revistas, ponencias en congresos, libros, tesis, etc. La

[2] Traducido del idioma inglés *Triggers*: Disparadores.

[3] Traducido del idioma ingles *Logs*: Registros.

mayoría de estas fuentes coinciden en que Oracle, Microsoft SQL Server y PostgreSQL forman parte de los principales SGBD para el desarrollo de sistemas informáticos existentes en la actualidad, ejemplo de ello es el libro Fundamentos de Bases de Datos, de los autores Abraham Silberschatz y Henry F. Korth [7]. Además en un estudio realizado mensualmente por la prestigiosa web DB Engine se evidencia que estos SGBD han estado durante los años 2013 y 2014 entre los cinco más utilizados por los desarrolladores de aplicaciones. Cabe destacar que en la jerarquía emergen con fuerza algunos sistemas no relacionales como MongoDB y Cassandra, y otros relacionales como MySQL y DB2. Por estas razones se consolida el propósito de ampliar el espectro de SGBD soportador por la herramienta *Schema Tool* [12] .

1.4.1 Principales componentes de los SGBD

Los Sistemas Gestores de Bases de Datos incluyen un conjunto de componentes fundamentales para su correcto funcionamiento, dentro de estos se pueden destacar el diccionario de datos, las herramientas de gestión y las herramientas de programación. A continuación se detallan características de cada uno de ellos.

Diccionario de Datos:

Debido a la complejidad de los SGBD es necesario controlar no solo los datos que forman la BD sino también otro tipo de información interna como usuarios, permisos y estructuras. El diccionario de datos es el lugar donde se almacena toda esta información, es una guía en la que se describe la BD y los elementos que la forman.

Herramientas de Gestión:

Son las herramientas que permiten a los administradores de la BD, la gestión de la misma. Entre otras funciones permiten crear la BD, modificar su diseño, manipularla o crear usuarios y permisos. Estas herramientas han adquirido con el paso del tiempo sofisticadas prestaciones, facilitando la realización de trabajos que hasta hace poco tiempo exigían verdaderos esfuerzos a los administradores.

Herramientas de programación:

Muchos SGBD ofrecen herramientas a los desarrolladores para crear aplicaciones que utilizarán los usuarios finales para acceder al banco de datos ya que con frecuencia los usuarios finales de la BD, tienen pocos conocimientos de informática y es necesario desarrollar aplicaciones que realicen por ellos las consultas o modificaciones en la BD [11].

1.4.2 Lenguajes para la comunicación con el SGBD

Para la comunicación con el SGBD, los usuarios, ya sean programas de aplicación o usuarios directos, utilizan dos lenguajes fundamentales en dicho proceso. Estos lenguajes también constituyen componentes de un SGBD. El primero es el especializado en la escritura de esquemas o descripción de la BD y se conoce genéricamente como DDL[4]. Las sentencias DDL constituyen un subconjunto de las sentencias SQL. Entre ellas se pueden encontrar CREATE, ALTER, DROP y TRUNCATE, que permiten al administrador de BD la gestión de los diferentes objetos que hacen realidad el resultado de un diseño físico determinado. Los resultados del empleo de estas sentencias SQL que componen el DDL quedan registrados en el catálogo del SGBD.

El segundo lenguaje es el encargado de la representación de la información mediante consultas y el mantenimiento y es conocido como DML[5]. Dentro de las sentencias del tipo DML pueden utilizarse las instrucciones SELECT para hacer consultas, e INSERT, UPDATE y DELETE para el mantenimiento de los datos. Lo más frecuente en la comunicación con el SGBD es que el mismo lenguaje posibilite el empleo de ambas funciones, tanto DDL como DML. El lenguaje SQL brinda esta posibilidad y es el más utilizado para gestionar las BD relacionales [13].

1.4.3 Oracle

Oracle es un SGBD que fue desarrollado en 1977 inicialmente con el nombre *Software Development Laboratories*. La compañía cuyo nombre cambió posteriormente a Oracle, se estableció para construir un SGBD como producto comercial y fue la primera en lanzarlo al mercado. Desde entonces Oracle ha

[4] Traducido del idioma inglés *Data Definition Language*: Lenguajes de Definición de Datos.

[5] Traducido del idioma inglés *Data Management Language*: Lenguajes de Manipulación de Datos.

mantenido una posición líder en el mercado de las BD relacionales y con el paso de los años sus productos y servicios han estado en constante auge.

Aparte de las herramientas directamente relacionadas con el desarrollo y gestión de las BD, Oracle también comercializa herramientas de inteligencia de negocio, incluyendo SGBD multidimensionales y un servidor de aplicaciones con una integración cercana al servidor de la BD. Además la compañía ofrece software para la planificación empresarial de recursos y gestión de relaciones con el cliente, incluyendo áreas como finanzas, recursos humanos, manufactura, mercadotecnia, ventas y gestión de cadenas de suministro.

La mayor parte de las herramientas de diseño de Oracle están incluidas en Oracle Internet Development Suite. Se trata de una familia de herramientas para los distintos aspectos de desarrollo de aplicaciones, incluyendo herramientas para el desarrollo de formularios, modelado de datos, informes y consultas. La principal herramienta de diseño de BD en la familia es Oracle Designer, que traduce la lógica de negocio y el flujo de datos en definiciones de esquemas y guiones procedimentales para la lógica de las aplicaciones. Esta herramienta soporta varias técnicas de modelado tales como diagramas entidad-relación; además proporciona herramientas de consulta, generación de informes y análisis de datos, incluyendo OLAP[6].

Oracle tiene soporte extensivo para constructores relacionales orientados a objetos, incluyendo:

- Tipos de objetos: Soporta un único modelo de herencia para las jerarquías de tipos.
- Tipos de colecciones: Soporta varrays, que son arreglos de longitud variable, y tablas anidadas.
- Tablas de objetos: Se utilizan para almacenar objetos mientras se proporciona una vista relacional de los atributos de los mismos.
- Funciones de tablas: Son funciones que producen conjuntos de filas como salida y se pueden utilizar en la cláusula FROM de una consulta. Las funciones de tablas se pueden anidar en Oracle. Si una

[6] Traducido del idioma inglés *On-line Analytical Processing*: Procesamiento analítico en línea es el proceso interactivo de crear, mantener, analizar y elaborar informes sobre datos [47].

función de tablas se utiliza para expresar algún formulario de transformación de datos, el anidamiento de varias funciones permite que se expresen varias transformaciones en una única instrucción.

Oracle tiene dos lenguajes procedimentales principales, PL/SQL y Java. PL/SQL fue el lenguaje original de Oracle para los procedimientos almacenados. Java se soporta mediante una máquina virtual Java dentro del motor de la BD. Oracle proporciona un paquete para encapsular procedimientos, funciones y variables relacionadas en unidades únicas, soporta SQLJ[7] y proporciona una herramienta para generar las definiciones de clases Java correspondientes a tipos de la BD definidos por el usuario [7].

1.4.4 PostgreSQL

PostgreSQL es un SGBD con cerca de una década de desarrollo y constituye el gestor de código abierto más avanzado disponible hoy en día. Liberado bajo la licencia BSD[8], es mantenido por la organización *PostgreSQL Global Development Team*, contando con la colaboración de una amplia comunidad de usuarios y programadores denominada PGDG[9]. Entre sus características fundamentales se encuentra el control de concurrencia multi-versión, soportando casi toda la sintaxis SQL[10], además de subconsultas, transacciones, tipos y funciones definidas por el usuario. PostgreSQL cuenta con un amplio conjunto de enlaces con lenguajes de programación como C, C++, Java, Python, entre otros. Utiliza el modelo cliente-servidor para garantizar la estabilidad del sistema y multiprocesos en lugar de multihilos, permitiendo así que un fallo en uno de los procesos no afecte al resto y el sistema continúe funcionando [14].

Este SGBD cumple con los estándares SQL de interoperabilidad y compatibilidad, destacándose por su rendimiento, seguridad y alta disponibilidad. Además utiliza la tecnología MVCC[11] para conseguir una mejor respuesta en ambientes de grandes volúmenes de datos [15].

[7] SQL incorporado en Java.

[8] Traducido del idioma inglés *Berkeley Software Distribution*: Licencia de software libre permisiva.

[9] Traducido del idioma inglés *PostgreSQL Global Development Group*: PostgreSQL grupo de desarrollo mundial.

[10] Traducido del idioma inglés *Structured Query Language*: Lenguaje de consulta estructurado es el lenguaje estándar utilizado para consultar las bases de datos relacionales, permite (además de opciones más avanzadas) crear, modificar o borrar tablas, así como insertar, eliminar, modificar o consultar los elementos de las mismas [1].

[11] Traducido del idioma inglés *Multi-Version Concurrency Control:* Control de concurrencia multiversión.

PostgreSQL tiene dos formas esenciales de ofrecer metainformación: mediante esquemas de información y mediante catálogos. El esquema de información es un esquema denominado information_schema y existe en todas las BD, su dueño es el usuario inicial en el cluster de BD y ese usuario por defecto posee todos los privilegios en dicho esquema. De manera general son un conjunto de vistas que contienen información sobre los objetos definidos en cada instancia. Los catálogos, están representados en cada BD en PostgreSQL mediante un esquema denominado pg_catalog, contienen todas las tablas del sistema y almacenan los metadatos del mismo; ejemplo de estas tablas son pg_database, pg_class y pg_atribute, las cuales brindan información muy valiosa a los usuarios [16].

1.4.5 Microsoft SQL Server

Microsoft SQL Server es un sistema gestor de base de datos desarrollado por Microsoft basado en el modelo relacional, que utiliza para las consultas el lenguaje T-SQL y ANSI-SQL. Entre sus características fundamentales se encuentran el soporte de transacciones y procedimientos almacenados, la compatibilidad con el modelo cliente-servidor y la administración de información relativa a otros servidores de datos. Es soportado por diversos entornos de desarrollo integrado, entre estos Microsoft Visual Studio y productos de la plataforma .NET. También proporciona servicios de replicación de datos entre varias copias de SQL Server así como con otros SGBD. Su servicio de análisis incluye OLAP y recopilación de datos.

SQL Server proporciona una gran colección de herramientas gráficas y asistentes que guían a los administradores de las BD por tareas tales como establecer copias de seguridad regulares, replicación de datos entre servidores y ajustes de rendimiento. Además proporciona acceso a herramientas visuales de BD que proporcionan tres mecanismos para ayudar al diseño de la BD: el diseñador del banco de datos, el diseñador de tablas y el diseñador de vistas [7].

Este sistema incluye una versión reducida llamada MSDE, con el mismo motor de BD pero orientado a proyectos más pequeños, que en sus versiones 2005 y 2008 pasa a ser el SQL Express Edition que se distribuye de forma gratuita [17].

1.5 Microsoft OLE DB

Microsoft OLE DB[12] es una tecnología desarrollada por Microsoft, utilizada para tener acceso a diferentes fuentes de información o BD, de una manera uniforme. Permite separar los datos de la aplicación que los requiere, ya que diferentes aplicaciones requieren acceso a diferentes tipos y almacenes de datos, y no necesariamente desean conocer cómo tener acceso a cierta funcionalidad con métodos de tecnologías específicas.

Está conceptualmente dividido en consumidores y proveedores. El consumidor es la aplicación que requiere acceso a los datos y el proveedor es el componente de software que expone una interfaz OLE DB a través del uso del COM[13]. Como las diferentes fuentes de datos pueden tener diferentes capacidades, es posible que los proveedores OLE DB no implementen todas las interfaces posible para OLE DB. Las capacidades disponibles son implementadas a través del uso de objetos COM; el proveedor OLE DB asocia la funcionalidad de una tecnología a una interfaz COM particular [18].

1.6 Análisis de soluciones similares

Actualmente en el mundo existen varios sistemas para la sincronización de datos que permiten efectuar operaciones en BD distribuidas. Durante la investigación, se analizaron los principales productos de los que existe información disponible, con objetivo similar a *Schema Tool*, con la perspectiva de que sus características se tengan en cuenta en el desarrollo de la nueva versión de dicha herramienta. A continuación se definen las principales características de dichos productos:

1.6.1 DatabaseSpy

DatabaseSpy es una herramienta desarrollada por Altova [19] (ver **Figura 5**) que permite diseñar y comparar múltiples BD proporcionando la edición de SQL, el diseño de estructuras y la edición de contenido. Para el control de cambios realizados en tablas, incluye una potente herramienta de comparación y combinación, con la que se pueden realizar todas estas operaciones. Además puede

[12] Traducido del idioma inglés *Object Linking and Embedding for Databases*: Enlace e incrustación de objetos para bases de datos.

[13] Traducido del idioma inglés *Component Object Model*: Modelo de objetos componentes.

comparar versiones diferentes de una misma tabla en un mismo tipo de BD y comparar e incluso combinar el contenido de tablas equivalentes y de esquemas de diferente tipo. Es compatible con una amplia gama de SGBD como Oracle en sus versiones 9i, 10g y 11g, PostgreSQL 8, 9.0, 10, 9.1.6 y 9.2.1, Microsoft SQL Server 2000, 2005, 2008 y 2012 entre otros. DatabaseSpy es un producto de código propietario, siendo imposible el acceso a su código fuente; por otro lado no brinda información alguna sobre los procedimientos utilizados en su desarrollo [20].

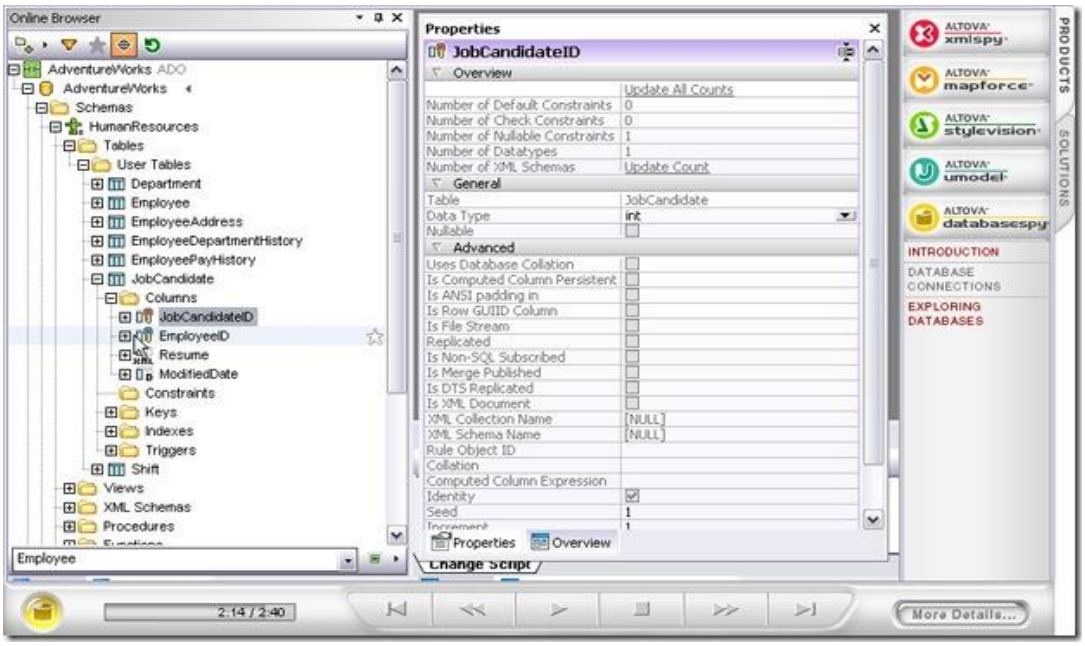

Figura 5. Altova DatabaseSpy.

1.6.2 Daffodil Replicator

Daffodil Replicator es una poderosa herramienta de código abierto desarrollada en Java (ver **Figura 6)** para la integración, migración y protección de datos en tiempo real. Permite realizar la replicación de datos según el modelo cliente-servidor, de acuerdo a las necesidades de los usuarios. Permite la sincronización entre BD homogéneas y heterogéneas, incluyendo Oracle, MySQL, Daffodil DB, PostgreSQL, entre otras.

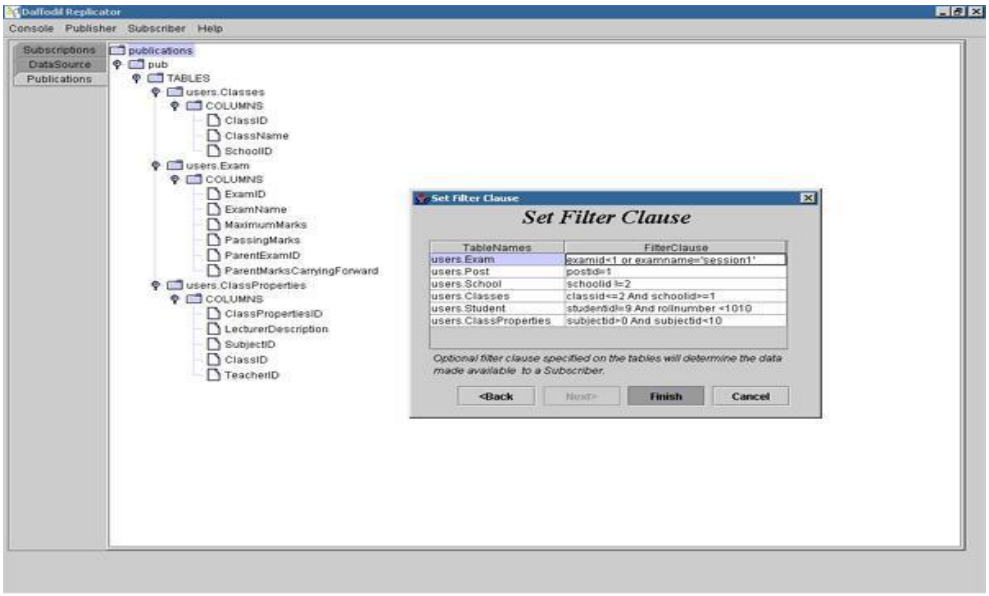

Figura 6. Daffodil Replicator.

Incluye además la capacidad de detectar y permitir resolución de conflictos, facilidades para replicar entre BD con estructuras iguales y soporte para la replicación de datos de gran tamaño. Es compatible con una variedad de topologías de sincronización, modos de sincronización y sincroniza fuentes de datos mediante el uso del protocolo TCP/IP o protocolos de transferencia HTTP [21].

1.6.3 Replicador Reko

Reko ha sido implementado en la Universidad de las Ciencias Informáticas como respuesta a la necesidad de mantener actualizadas un conjunto de BD. Pretende cubrir las principales necesidades relacionadas con la distribución de datos entre los gestores más populares como la protección, recuperación, sincronización de datos, transferencia de datos entre diversas localizaciones o la centralización de la información en una única localización. Facilita la representación de escenarios de replicación complejos al permitir la definición de localizaciones o nodos y la selección de los datos de replicación. Brinda una herramienta web para la administración y el monitoreo de los datos de replicación soportando los SGBD Oracle y PostgreSQL.

Reko ha garantizado desde noviembre del año 2008, la replicación de datos en SIGEP que es la entidad encargada de los procesos de información y gestión del empleo público en la república de Colombia, presentando buenos resultados. Además está certificado por CALISOFT que es Centro para la Excelencia en el Desarrollo de Proyectos Tecnológicos de la Universidad de las Ciencias Informáticas (UCI), por lo cual se considera un software confiable y de alta calidad [22].

1.6.4 Conclusiones del análisis

El análisis realizado anteriormente permitió definir las principales características que engloban los sistemas de este tipo y que por tanto deben ser tenidas en cuenta para el *Schema Tool*.

Por otra parte se concluye que ninguna de las herramientas estudiadas posibilita la creación de esquemas de datos sobre alguna BD para posteriormente ser sincronizados en otra. Además no brindan la posibilidad de efectuar sincronizaciones entre bases de datos que tengan estructuras similares pero que se encuentren en diferentes sistemas gestores, siendo estas algunas de las características fundamentales de *Schema Tool*.

1.7 Metodologías de desarrollo

Una metodología de desarrollo de software es una filosofía o marco de trabajo para estructurar, planificar y controlar el proceso de desarrollo de software; especifica un conjunto de pasos o procedimientos que definen quién debe hacer qué, cuándo y cómo debe hacerlo. Su objetivo principal es guiar a los desarrolladores de software en la obtención de un producto de calidad, que cumpla con los requerimientos establecidos por el cliente, ajustándose a los recursos apropiados y a un costo razonable [23].

1.7.1 Metodología XP (*Extreme Programming*)

Formulada por Kent Beck[14], XP es la metodología más destacada dentro de los procesos ágiles del desarrollo de software, siendo ideal para equipos pequeños. Se diferencia de las metodologías tradicionales principalmente en que pone más énfasis en la adaptabilidad que en la previsibilidad. Sus

[14] Ingeniero de software estadounidense pionero en patrones de diseño de software.

defensores consideran que los cambios de requisitos sobre la marcha son un aspecto natural, inevitable e incluso deseable del desarrollo de proyectos. Creen que ser capaz de adaptarse a los cambios de requisitos en cualquier punto de la vida del proyecto es una aproximación mejor y más realista que intentar definir todos los requisitos al comienzo del proyecto e invertir esfuerzos después en controlar los cambios en los requisitos.

Dentro de sus principales ventajas pueden presenciarse la fácil adaptación a entornos volátiles, la planificación transparente para los clientes, que conocen las fechas de entrega de las funcionalidades y la posibilidad de definir en cada iteración los objetivos de la siguiente. Además es una metodología ideal para la programación en parejas, volátil, con desarrollo iterativo e incremental y simple; por estos motivos fue seleccionada para guiar el proceso de desarrollo de software en la presente investigación.

Algunas de sus características fundamentales consisten en:

Pruebas unitarias continuas: Frecuentemente repetidas y automatizadas, incluyendo pruebas de regresión. Se aconseja escribir el código de la prueba antes de la codificación.

Frecuente interacción**:** Se recomienda que un representante del cliente trabaje junto al equipo de desarrollo.

Propiedad del código compartida: En vez de dividir la responsabilidad en el desarrollo de cada módulo en grupos de trabajo distintos, este método promueve el que todo el personal pueda corregir y extender cualquier parte del proyecto.

Simplicidad en el código: La programación extrema apuesta que es más sencillo hacer algo simple y tener un poco de trabajo extra para cambiarlo si se requiere, que realizar algo complicado y quizás nunca utilizarlo [23].

1.8 Descripción de *Schema Tool*

Schema Tool es una herramienta de apoyo al trabajo con sistemas con BDD que cuenta con varias funcionalidades que optimizan el trabajo con la BD, básicamente en dos sentidos: durante el desarrollo de aplicaciones, donde permite crear nuevos planes, esquemas y generar los permisos sobre éstos, y

además como herramienta para sincronizar determinada información entre distintas BD. Actualmente presenta como principales limitantes que solo se integra con el SGBD relacional Oracle y que se encuentra desarrollada sobre una versión obsoleta del *framework* .NET, lo cual imposibilita su empleo en otros entornos que utilicen BDD.

Con esta herramienta un desarrollador de aplicaciones puede crear esquemas que realizan todo el mapeo de las tablas que desee encuestar en el servidor maestro y por otro lado, organizar el flujo de datos y definir en qué orden serán insertados, actualizados o borrados en el servidor esclavo. Además el *Schema Tool* puede ayudar a un administrador de BD a preparar un entorno de pruebas pasando datos de un servidor a otro, incluso corregir alguna información que falte en algunos casos excepcionales donde por algún error de hardware o réplica, exista pérdida parcial de los datos.

A continuación se explican las principales funcionalidades de la herramienta. En la ventana principal *Schema Editor* se especifican los filtros que intervienen en la sincronización (ver **Figura 7**), los cuales pueden ser por parámetro(s), o por referencia(s).

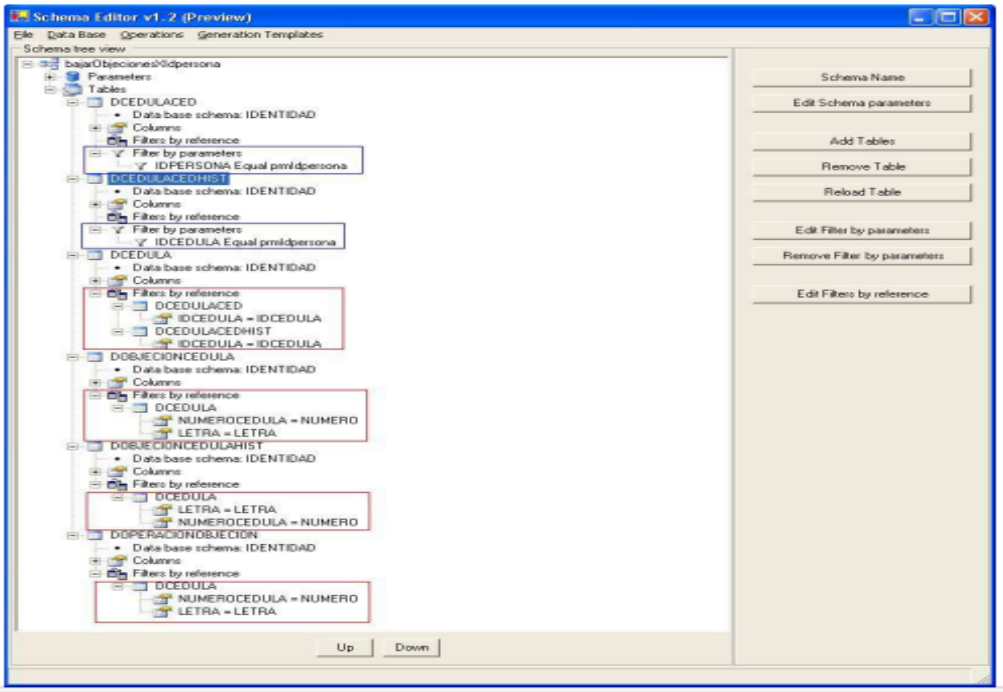

Figura 7. Ventana *Schema Editor.*

Un esquema puede no tener parámetros si se quieren sincronizar todas las tuplas de las tablas que forman parte del proceso. También en esta ventana se establece el orden de las tablas, encabezando la jerarquía con aquellas que se filtrarán por los parámetros de entrada del esquema y estableciendo el orden en base a las dependencias entre tablas.

Una vez seleccionadas las tablas que se desean (y las que se necesitan) sincronizar, corresponde hacer el plan de ejecución, o lo que es lo mismo, determinar el orden de operaciones que debe seguirse para sincronizar la información (ver **Figura 8**), garantizando la integridad de la misma y que no se violen las llaves foráneas. La herramienta propone un plan inicial para cada esquema, pero dicho plan debe revisarse y readaptarse a las condiciones específicas del entorno de replicación.

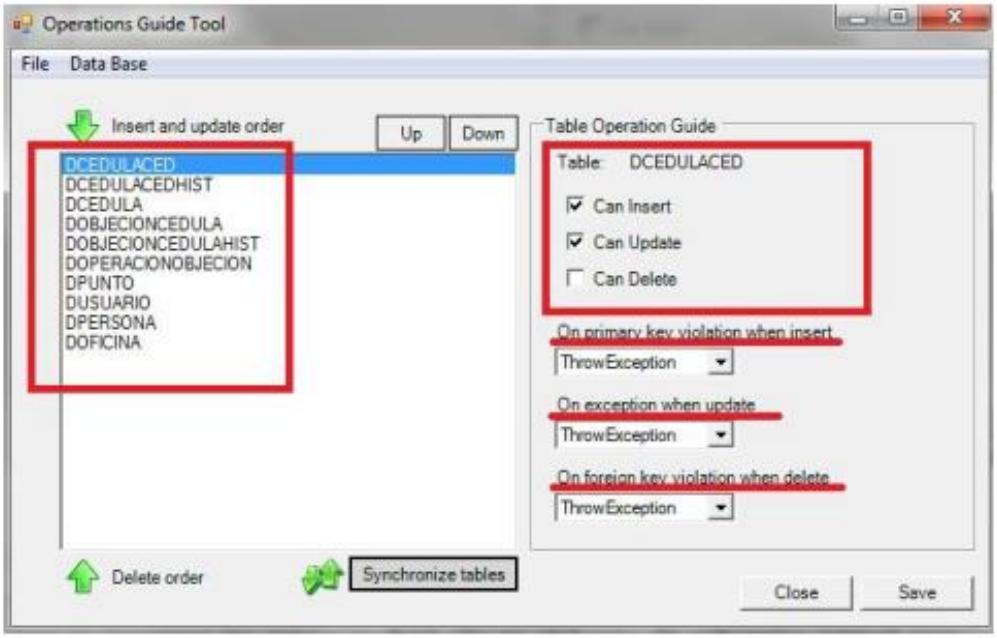

Figura 8. Ventana *Operations Guide Tool.*

Después de definir correctamente el plan de ejecución, se procede a analizar las posibles operaciones que deben ocurrir en la base de datos esclava para garantizar la homologación de ambas fuentes (ver **Figura 9**). Estas operaciones pueden probarse sin ser aplicadas para saber si hay errores por violación de alguna integridad referencial. Si no se detectan errores el proceso de sincronización está listo para ejecutarse.

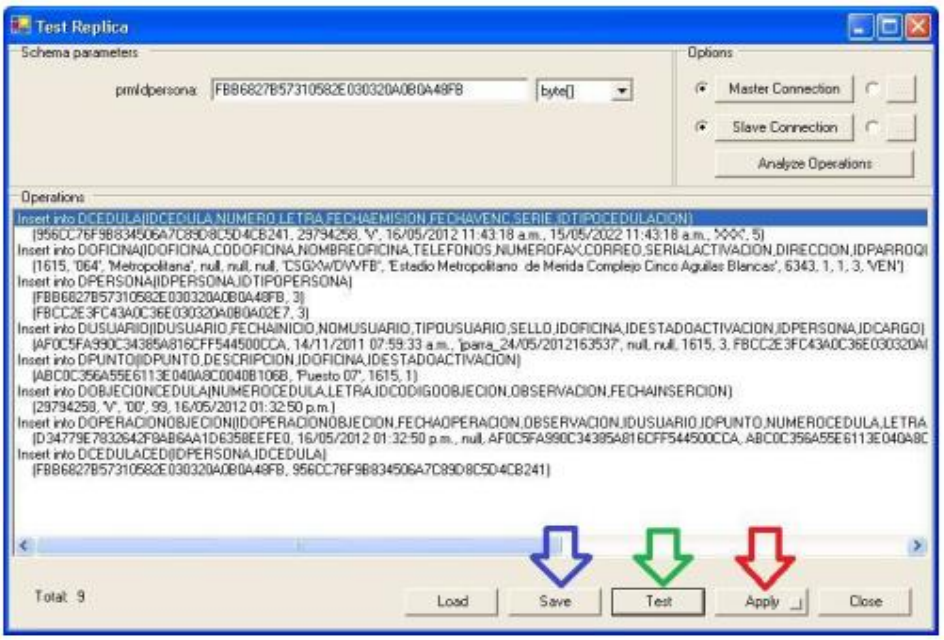

Figura 9. Ventana *Test Replica.*

1.9 Herramientas y tecnologías a utilizar en el proceso de desarrollo de software

A continuación se detallan los lenguajes, herramientas y tecnologías utilizados en el proceso de desarrollo de software.

1.9.1 Lenguaje unificado de modelado (UML)

Para desarrollar la actualización de la herramienta se propone utilizar el lenguaje UML, ya que este tiene una notación gráfica muy expresiva que permite representar en mayor o menor medida todas las fases de un proyecto informático: desde el diseño con los diagramas de clases, hasta la implementación y configuración con los diagramas de despliegue. UML sirve para el modelado completo de sistemas complejos, tanto en el diseño de los sistemas de software como para la arquitectura de hardware donde se ejecuten [24].

Es importante remarcar que UML es un lenguaje de modelado para especificar o para describir métodos o procesos. Se utiliza para definir un sistema, para detallar los artefactos en el sistema y para documentar y construir. En otras palabras, es el lenguaje en el que está descrito el modelo. Se puede aplicar en el desarrollo de software en gran variedad de formas para dar soporte a una metodología de desarrollo de software, pero no especifica en sí mismo qué metodología o proceso usar.

Este lenguaje recibió la aprobación de facto en la industria, pues sus creadores representaron métodos muy difundidos de la primera generación del análisis y diseño orientados a objetos y ha sido adoptado por varias organizaciones dedicadas al desarrollo de software entre ellos los proveedores de herramientas CASE[15] [24].

1.9.2 Herramienta para el modelado de objetos

Las herramientas de modelado de objetos se emplean para la creación de modelos de sistemas que ya existen o que se desarrollarán. Permiten crear un "simulacro" del sistema, a bajo costo y riesgo mínimo. A bajo costo porque, al fin y al cabo, es un conjunto de gráficos y textos que representan el sistema, pero no son el sistema físico real. Además minimizan los riesgos, porque los cambios que se deban realizar se llevarán a cabo más fácil y rápidamente sobre el modelo que sobre el sistema ya implementado.

Las herramientas de modelado, permiten concentrarse en ciertas características importantes del sistema, prestando menos atención a otras. Los modelos resultantes, son una buena forma de determinar si están representados todos los requerimientos, como también de comprobar si el analista comprendió qué hará el sistema [25].

Visual Paradigm

Visual Paradigm es una herramienta profesional para el modelado de objetos que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. Además ayuda a construir aplicaciones de mejor calidad y de menor costo. Permite diseñar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar

[15] Traducido del idioma inglés *Computer Aided Software Engineering*: Ingeniería de Software Asistida por Computadora son diversas aplicaciones informáticas destinadas a aumentar la productividad en el desarrollo de software.

documentación. Entre las características fundamentales de Visual Paradigm se destacan el modelado colaborativo con CVS[16] y Subversion además de la interoperabilidad con modelos UML2 [26].

1.9.3 Plataforma de desarrollo

Una plataforma de desarrollo o marco de trabajo, como también se le conoce, es un conjunto estandarizado de concepciones, prácticas y criterios para enfrentar un tipo común de problema, resolviéndolo de forma rápida y eficaz. Su objetivo principal es proveer una estructura de modo que los desarrolladores no tengan que hacer el código desde cero siempre y puedan volver a utilizar la gran mayoría de este. Ofrece componentes, provee plantillas o esqueletos que precisan el funcionamiento de las aplicaciones. Permite manejar y controlar prácticamente toda la aplicación sin escribir mucho código [27].

.NET

La plataforma .NET de Microsoft es un componente de software que puede ser añadido al sistema operativo Windows. Provee un extenso conjunto de soluciones predefinidas para necesidades generales de la programación de aplicaciones y administra la ejecución de los programas escritos específicamente con la plataforma. Presenta como característica principal la transparencia de redes, la independencia del hardware que se utilice y el diseño rápido de aplicaciones. Soporta más de 20 lenguajes, haciendo posible desarrollar cualquier tipo de aplicación con cualquiera de ellos. Los más utilizados son C#, Visual Basic y Delphi [28]. Durante la etapa de desarrollo se actualizará el marco de trabajo de la herramienta a la versión 4.5 de .NET que constituye la última actualización existente de la plataforma.

1.9.4 Lenguaje de Programación

Según la definición teórica, se entiende como lenguaje a todo aquel sistema de comunicación que posee una determinada estructura, contenido y uso. La programación es, en el vocabulario propio de la informática, el procedimiento de escritura del código fuente de un software. De esta manera puede decirse

[16] Traducido del idioma inglés *Concurrent Versioning System*: Sistema de Versiones Concurrentes.

que la programación le indica al programa informático qué acción tiene que llevar a cabo y cuál es el modo de concretarla [29].

Lenguaje CSharp (C#)

Microsoft C# es un lenguaje de programación diseñado para crear un amplio número de aplicaciones que se ejecutan sobre el *framework* .NET. Supone una evolución de Microsoft C y Microsoft C++; es sencillo, moderno, proporciona seguridad de tipos y está orientado a objetos. El código creado mediante C# se compila como código administrado, lo cual significa que se beneficia de los servicios de CLR[17]. Estos servicios incluyen interoperabilidad entre lenguajes, recolección de elementos no utilizados, mejora de la seguridad y mayor compatibilidad entre versiones.

C# se presenta como Visual C# en el conjunto de programas Visual Studio .NET. Visual C# utiliza plantillas de proyecto, diseñadores, páginas de propiedades, asistentes de código, un modelo de objetos y otras características del entorno de desarrollo [30].

1.9.5 Entorno Integrado de Desarrollo (IDE)

Un IDE es un entorno de programación que ha sido empaquetado como un programa de aplicación, o sea, consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica. Puede ser una aplicación por sí sola o puede ser parte de una aplicación ya existente, además es posible que un mismo IDE pueda funcionar con varios lenguajes de programación. Este es el caso de Eclipse, al que mediante modificaciones se le puede añadir soporte de lenguajes adicionales [31].

Visual Studio .NET

Visual Studio .NET es un IDE para sistemas operativos Windows que soporta diversos lenguajes de programación tales como C++, C#, JavaScript y Visual Basic .NET, aunque al presente se han desarrollado las extensiones necesarias para muchos otros. Posibilita a los desarrolladores crear aplicaciones, sitios y servicios, así como productos web en cualquier medio que soporte la plataforma

[17] Traducido del idioma inglés *Common Language Runtime:* Lenguaje común en tiempo de ejecución.

.NET. Visual Studio 2013 es la última versión para Visual Studio, que suministra nuevas funcionalidades y correcciones, contando con mejoras que abarcan áreas características en todas las ediciones [32].

1.9.6 ADO.NET

ADO.NET es un conjunto de clases que exponen servicios de acceso a datos para el programador de .NET. Ofrece abundancia de componentes para la creación de aplicaciones de uso compartido de BDD. Constituye una parte integral del *framework* .NET y proporciona acceso a datos relacionales en formato XML y de aplicaciones. Satisface diversas necesidades de desarrollo, como la creación de clientes de BD de aplicaciones para usuario y objetos empresariales de nivel medio que utilizan aplicaciones, herramientas, lenguajes o exploradores de Internet.

Dentro de las principales ventajas de ADO.NET se encuentran que al haber una única conexión a la BD por usuario, o incluso a veces por aplicación, establecida permanentemente, puede llegar a resultar más sencillo administrar la seguridad y el acceso al servidor de datos. Lo mismo ocurre con el control de concurrencia ya que en un escenario donde múltiples usuarios se estuvieran conectando y desconectando permanentemente para realizar distintas acciones, este control sería más difícil de llevar, posibilitando siempre el acceso a los datos actualizados.

La arquitectura de ADO.NET está basada en el concepto de proveedores de acceso a datos, siendo un proveedor un conjunto de clases que permiten conectarse a una BD, ejecutar un comando sobre ella y tener acceso a los resultados de su ejecución, ADO.NET soporta el acceso a datos tanto en escenarios conectados como desconectados [33].

1.10 Selección del entorno de trabajo

Como se explicó anteriormente el *Schema Tool* forma parte de un conjunto de herramientas desarrolladas en el marco del proyecto Identidad, para apoyar el proceso de desarrollo de software en el SAIME. El centro de desarrollo CISED, como heredero principal del Proyecto Identidad determina algunos elementos a tener en cuenta para desarrollar sus productos informáticos, como son las herramientas, tecnologías y metodologías a utilizar, teniendo en cuenta el estado actual y el avance de estos a nivel mundial.

Referente a las herramientas y tecnologías para el desarrollo de la solución propuesta se propone que:

- Como herramienta de modelado se seleccione Visual Paradigm en su versión 8.0 con UML en su versión 2.1.

- El lenguaje de programación seleccionado sea C#, ya que la versión original de la herramienta *Schema Tool* se encuentra desarrollada sobre el mismo, razón por la cual la selección de este lenguaje permitirá la reutilización del código existente. De igual manera se facilitará la integración de diferentes proveedores de OLE DB para hacer más extensible la herramienta en su nueva versión.

- El IDE seleccionado para el desarrollo sea Visual Studio 2013, por ser el entorno ideal para la utilización del lenguaje C#, así como para aprovechar las nuevas prestaciones que permiten el fácil acceso y manipulación de BD sobre los sistemas gestores Microsoft SQL Server y Oracle. Otra razón es su fácil integración con la plataforma .NET, que constituye el principal *framework* sobre el que se sustentará la implementación de la herramienta.

- La tecnología para el acceso a datos sea ADO.NET, haciendo especial énfasis en el subconjunto de clases que permitirán el trabajo con la tecnología OLE DB de Microsoft..

- Los SGBD que se seleccionen para integrarse con la herramienta sean PostgreSQL en su versión 9.2.4, Microsoft SQL Server en su versión 2012 y Oracle en su versión 11g, las cuales constituyen las versiones más avanzadas de estos gestores.

Conclusiones parciales

El análisis de herramientas similares para la replicación de datos, permitió delimitar un grupo de características deseadas para el *Schema Tool*, a la vez que corroboró la importancia de dicha herramienta y la necesidad de su actualización. El estudio de las metodologías, herramientas y tecnologías actuales, posibilitó la selección fundamentada de XP como metodología de desarrollo y la herramienta de modelado Visual Paradigm. El lenguaje de programación C#, el *framework* .NET en su versión 4.5 como marco de trabajo, el entorno de desarrollo Visual Studio 2013 y ADO.NET para el acceso a datos, según las pautas establecidas en el centro CISED, garantizando la continuidad del proceso de desarrollo previo de la herramienta y minimizando los contratiempos que pudieran surgir por incompatibilidades con la solución existente.

Se decidió potenciar la integración de la herramienta con los SGBD Oracle 11g, Microsoft SQL Server 2012 y PostgreSQL 9.2.4, ya que a partir del estudio realizado se determinó que se encuentran entre los predominantes en el desarrollo de software en la actualidad.

Capítulo 2. Análisis y diseño de la herramienta *Schema Tool*

Introducción

En el presente capítulo se describe la evolución de la solución en sus fases iniciales de planificación y diseño. Se propone una solución utilizando como base la metodología XP y se muestran los resultados del proceso de levantamiento de requisitos funcionales y no funcionales, también se describen las historias de usuario relacionadas con estos. Además se relacionan los patrones de diseño a aplicar y se presentan el diagrama de clases y las tarjetas CRC correspondientes a las clases de mayor relevancia. Se representan el plan de entrega y el plan de iteraciones, ambos correspondientes a la etapa de planificación del proceso de desarrollo de software, que son de vital importancia para la entrega final de la herramienta *Schema Tool* en su nueva versión.

2.1 Modelo del dominio

El modelo de dominio se utiliza para ilustrar los principales conceptos que se manejan en el ámbito de la herramienta, así como las relaciones entre ellos, ayudando a los usuarios, desarrolladores e interesados a emplear un vocabulario común para poder entender el contexto en que se desarrolla *Schema Tool*. A continuación se muestra el modelo de dominio utilizado como punto de partida para la actualización de la herramienta (ver **Figura 10**).

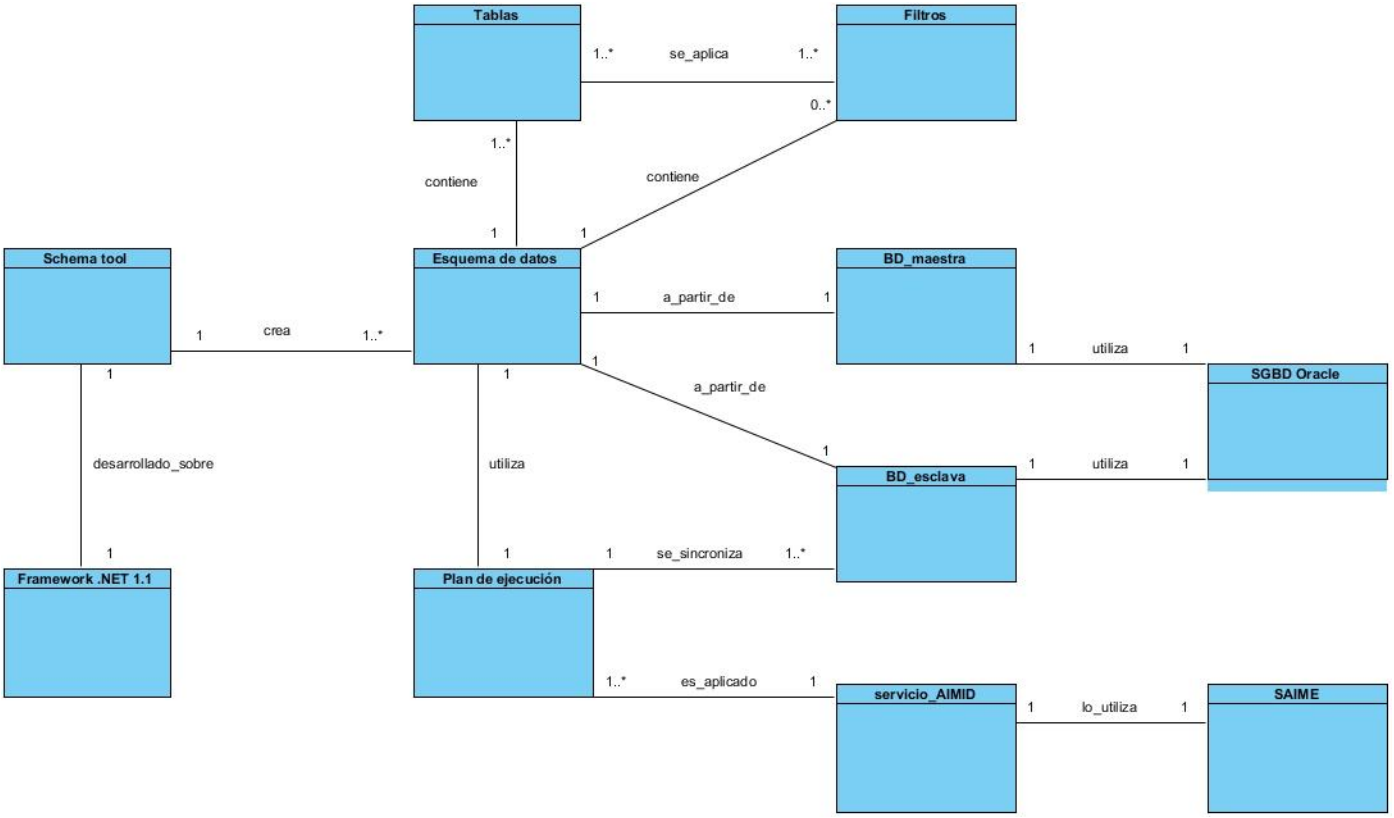

Figura 10. Modelo del dominio.

2.1.1 Descripción del modelo del dominio

La herramienta ***Schema Tool*** actualmente se encuentra desarrollada sobre la **versión 1.1 del *framework* .NET**. El principal objetivo de esta herramienta es llevar a cabo procesos de replicación de información mediante el empleo de **esquemas de datos** que son creados a partir de una BD desarrollada sobre el **SGBD Oracle.** Estos esquemas se componen de **tablas,** y pueden o no contener **filtros** que se apliquen sobre las mismas. Los esquemas permiten la selección de información en una **BD maestra** y su sincronización hacia una **BD esclava** a partir de **planes de ejecución**. Estos planes pueden ser usados por el **servicio AIMID** que está integrado al **SAIME** para la sincronización asincrónica de BD en dicho sistema.

2.1.2 Glosario de conceptos relevantes del modelo del dominio

- ***Schema Tool***: Herramienta para la sincronización de esquemas de datos mediante la aplicación o no de varios filtros.
- ***Framework* .NET**: Plataforma de programación desarrollada por Microsoft que provee de un extenso conjunto de soluciones predefinidas para necesidades generales de la programación de aplicaciones.
- **Esquema de datos**: Agrupación de tablas de las cuales se desea obtener información, sus relaciones y los filtros establecidos sobre ellas.
- **Plan de ejecución**: Fichero donde se especifican elementos esenciales para que ocurra el proceso de sincronización, como el respaldo ante una violación de llave primaria o foránea o qué hacer ante el lanzamiento de una excepción durante el proceso; además de especificar el orden de replicación de las tablas.
- **BD esclava**: BD destino donde se aplican las operaciones de cambio para garantizar la homologación.
- **BD maestra**: BD origen de la que se obtienen los datos para la replicación.
- **Filtros**: Conjunto de restricciones que se le aplica a varias tablas del esquema para seleccionar las tuplas a replicar. Los filtros pueden ser por parámetro si dependen de algún valor especificado por el usuario o por referencia cuando dependen de información relativa a otras tablas involucradas en el proceso de replicación.
- **Servicio AIMID**: Servicio utilizado para la ejecución de los planes elaborados con la herramienta en el sistema SAIME.
- **SGBD Oracle**: Único SGBD integrado a la versión inicial de la herramienta *Schema Tool*.

2.2 Metáfora

Para describir la herramienta es conveniente establecer una metáfora que muestre detalladamente como debe fluir el proceso a automatizar, en este caso el proceso de replicación de datos utilizando la nueva versión del *Schema Tool*. A continuación se describe la metáfora del sistema:

La herramienta debe funcionar como una aplicación de escritorio donde los usuarios se conectan a una BD desarrollada sobre los SGBD Oracle, PostgreSQL o SQL Server. Después de establecerse la conexión, la herramienta listará todas las tablas sobre las que tiene permiso el usuario, permitiendo su selección para ser incorporadas al esquema. Dichas tablas pueden o no tener relaciones con otras tablas en dependencia de como estén relacionados los datos en determinado negocio, por tanto en caso de que existan relaciones entre las tablas a sincronizarse, como sucede comúnmente, debe establecerse un orden de dependencias entre las mismas.

La herramienta permitirá la especificación de filtros mediante parámetros o referencias a tablas superiores en la jerarquía de relaciones, los cuales condicionarán las tuplas que serán seleccionadas en el momento de la replicación. Luego de la creación del esquema, el cual puede ser salvado para su posterior reutilización, la herramienta propondrá una estrategia o plan de ejecución de las operaciones de replicación. Dicho plan puede ser editado, para especificar el orden en que serán actualizadas las tablas que intervienen en el proceso, así como las operaciones DML que pueden efectuarse sobre cada de una de ellas y la estrategia de respaldo ante posibles errores. El plan de ejecución también puede ser guardado para su posterior reutilización.

Una vez establecido el plan de ejecución puede procederse a efectuar un proceso de réplica entre una BD maestra y una BD esclava, especificando los parámetros de conexión a cada una de ellas. Tanto la fuente como el destino pueden ser Oracle, PostgreSQL o SQL Server. Finalmente la herramienta permitirá analizar las operaciones que van a ocurrir y probar la réplica sin ser ejecutada en el destino. Si todo está correcto, se ejecutan las operaciones sobre la BD esclava homologándose así ambas fuentes.

2.3 Especificación de requisitos

Los requisitos funcionales son características del sistema que expresan funcionalidades, es decir, describen lo que el sistema debe hacer. A continuación se enumeran los requisitos funcionales del sistema a desarrollar, los cuales serán descritos con mayor detalle en las historias de usuario.

2.3.1 Requisitos funcionales

RF1 Crear esquema de datos.

RF1.1 Configurar parámetros de conexión a la BD.

RF1.2 Editar tablas del esquema de datos.

RF1.3 Editar filtros por parámetro del esquema de datos.

RF1.4 Editar filtros por referencia del esquema de datos.

RF2 Salvar esquema de datos.

RF3 Cargar esquema de datos.

RF4 Crear plan de ejecución.

RF4.1 Generar propuesta de plan de ejecución.

RF4.2 Editar propuesta de plan de ejecución.

RF5 Salvar plan de ejecución.

RF6 Cargar plan de ejecución.

RF7 Efectuar sincronización entre dos BD.

RF7.1 Definir conexiones a la BD maestra y BD esclava.

RF7.2 Analizar operaciones de cambio.

RF7.3 Aplicar operaciones de cambio sobre BD esclava.

RF8 Generar permisos sobre esquemas.

2.3.2 Requisitos no funcionales

Los requisitos no funcionales delimitan las condiciones que el sistema necesita para poder funcionar correctamente o prestar los servicios a los usuarios [34]. A continuación se especifican los requisitos no funcionales de la herramienta.

Requisitos no funcionales de hardware:

- Estaciones cliente: CPU 1.60 GHz o superior y 512 MB de memoria RAM o superior.
- Se requiere una tarjeta de red en todas las estaciones que intervienen en el proceso.

Requisitos no funcionales de software:

La estación cliente debe tener instalado:

- Windows 7 o superior
- *Framework* .NET version 4.5
- PGNP OLE DB Provider
- Oracle Provider for OLE DB

La estación servidor (maestro o esclavo) debe tener instalado:

- PostgreSQL en su versión 9.2.1 o superior
- Microsoft SQL Server 2008 o superior

- Oracle en su versión 10g o superior

Restricciones en el diseño e implementación:

- Lenguaje de programación: C Sharp 5.0.
- IDE: Visual Studio 2013.
- Plataforma: *Framework* .Net 4.5

Requisito de seguridad:

Se requiere un usuario con permisos de lectura sobre la BD maestra y otro con permisos de lectura y escritura sobre la BD esclava.

Requisito de rendimiento:

El proceso de comparación entre dos fuentes de datos y la aplicación de las operaciones de cambio deben llevarse a cabo en un tiempo inferior a los 3 segundos.

2.4 Proceso de actualización de la herramienta a la versión 4.5 del *framework* .NET

La versión original de la herramienta *Schema Tool* fue desarrollada sobre la versión 1.1 de la plataforma .NET, la cual fue publicada a partir del mes de abril del año 2003. En enero del año 2006 fue liberada la versión 2.0 de dicho *framework*, que contuvo un gran número de actualizaciones y cambios críticos respecto a su predecesor, más allá de la mera incorporación de nuevas funcionalidades y prestaciones.

Los cambios en las funcionalidades ya existentes hasta ese momento, pueden clasificarse dentro de dos categorías: los cambios críticos, que implican la redefinición del nombre o del grupo de parámetros de determinadas funciones, clases o determinados espacios de nombres y los cambios de comportamiento, donde la firma del método se mantiene pero el mismo se desempeña de manera diferente. Algunos de estos cambios condicionaron el proceso de actualización de la versión del *framework* para la herramienta. A continuación se mencionan algunas funciones, clases o espacios de nombres que eran usadas en la versión original de *Schema Tool* y a las cuales, tras sufrir cambios críticos o de comportamiento

considerables, se les buscó una alternativa para garantizar la compatibilidad con la última versión del *framework:*

System.IO.DirectoryInfo.Parent, *System.IO.DirectoryInfo.Name, Assembly.FullName,*

Type.AssemblyQualifiedName, *System.IO.StreamReade*r, *System.Collections.Hashtable,*

DataTable.RejectChanges, *OleDbParameter.Precision*, *OleDbParameter.Scale,*

DataSet.ReadXml (), DataTable.Rows.Clear (), SqlCommand.ExecuteXmlReader (),

DataSet.Merge (DataSet dataSet), XmlSerializer.

La incorporación de importantes prestaciones y funcionalidades en las versiones 3.0, 3.5, 4.0 y 4.5, así como el uso de proveedores de OLE DB que han sido implementados sobre dichas plataformas, posibilitó además el rediseño de la arquitectura de la herramienta para su adaptación a otros SGBD, así como para garantizar su extensibilidad en el futuro.

Otro importante problema que se enfrentó durante el proceso de actualización de la herramienta fue provocado por la dualidad de arquitecturas de CPU existentes en la actualidad. Mientras que para Windows XP todos los programas se ejecutaban sobre 32 bits, en Windows 7 y versiones superiores, una distribución para una arquitectura de 64 bits, puede ejecutar programas que han sido desarrollados para 32 bits. Al ejecutar un archivo UDL (Universal Data Link) sobre una arquitectura de 64 bits, el mismo listará solo aquellos proveedores OLE DB desarrollados sobre dicha arquitectura, obviando aquellos de 32 bits que se tengan instalados en la estación de trabajo. Esto se debe a que la ejecución de un archivo UDL invoca a las bibliotecas rundll32.exe y OleDb32.dll, las cuales dependen de la arquitectura del sistema operativo instalado.

2.5 Historias de usuario

Las historias de usuario son la técnica utilizada en XP para especificar los requisitos del software, representando una breve descripción del comportamiento del mismo, con el empleo de una terminología del cliente sin lenguaje técnico [35]. A continuación se muestran las historias de usuario Analizar

operaciones de cambio y Aplicar operaciones de cambio en la BD esclava, que son dos de las más significativas en la nueva versión de la herramienta (ver **Tablas 1 y 2**).

Historia de Usuario	
Número: HU_12	**Nombre:** Analizar operaciones de cambio.
Usuario: Desarrollador de aplicaciones	
Prioridad en Negocio: Alta	**Riesgo en Desarrollo:** Alto
Programador responsable: Ernesto Valdés Sotolongo	
Iteración Asignada: 3	
Descripción: La presente historia de usuario tiene como objetivo analizar las operaciones de cambio que van a llevarse a cabo durante el proceso de replicación. La herramienta, a partir de una comparación entre la información extraída desde la base de datos maestra y la extraída desde la base de datos esclava, listará las operaciones DML de cambio que deben ser ejecutadas en esta última para garantizar la homologación de ambas. La herramienta debe permitir que estas operaciones sean probadas para detectar posibles errores.	
Observaciones: ---	

Tabla 1. HU_12 Analizar operaciones de cambio.

Historia de Usuario	
Número: HU_13	**Nombre:** Aplicar operaciones de cambio en la BD esclava.
Usuario: Desarrollador de aplicaciones	

Prioridad en Negocio: Alta	**Riesgo en Desarrollo:** Alto
Programador responsable: Ernesto Valdés Sotolongo	
Iteración Asignada: 3	
Descripción: La presente historia de usuario tiene como objetivo aplicar las operaciones de cambio, que fueron analizadas previamente, sobre la base de datos esclava, garantizando así la homologación de la misma con la base de datos maestra.	
Observaciones: ---	

Tabla 2. HU_13 Analizar operaciones de cambio en la BD esclava.

2.6 Plan de iteraciones

El plan de iteraciones determina la duración de cada una de las iteraciones previstas para el desarrollo del sistema, lo que permite la estimación del tiempo requerido hasta la obtención del producto final del proyecto. Con este plan se logra una mayor organización y estructuración del trabajo al mostrar el orden en que serán implementadas cada una de las historias de usuario dentro de cada iteración.

Iteración 1: En esta iteración se implementarán las historias de usuario relacionadas con la configuración de los parámetros de conexión a la BD y la edición de las tablas y los filtros del esquema de datos.

Iteración 2: En esta iteración se implementarán las historias de usuario relacionadas con salvar y cargar los esquemas de datos y los planes de ejecución, además de generar y editar las propuestas de plan de ejecución.

Iteración 3: En esta iteración se realizarán las historias de usuario relacionadas con el análisis de las operaciones de cambio y su aplicación sobre la BD esclava. Una vez concluida esta iteración la herramienta se encontrará completamente terminada y lista para su explotación.

En la **Tabla 3** se resume esta planificación:

Iteración	No. HU	Historia de Usuario	Semanas estimadas
Iteración 1	HU_01 HU_02 HU_03 HU_04	– Configurar parámetros de conexión a la BD. – Editar tablas del esquema de datos. – Editar filtros por parámetro del esquema de datos. – Editar filtros por referencia del esquema de datos.	16
Iteración 2	HU_05 HU_06 HU_07 HU_08 HU_09 HU_10	– Salvar esquema de datos. – Cargar esquema de datos. – Generar propuesta de plan de ejecución. – Editar propuesta de plan de ejecución. – Salvar plan de ejecución. – Cargar plan de ejecución.	16
Iteración 3	HU_11 HU_12 HU_13 HU_14	– Definir conexiones a la BD maestra y BD esclava. – Analizar operaciones de cambio. – Aplicar operaciones de cambio sobre BD esclava.	4

		– Generar permisos sobre esquema	

Tabla 3. Plan de iteraciones.

2.7 Plan de entregas

El plan de entregas es un documento que especifica exactamente qué historias de usuario serán implementadas en cada entrega de la herramienta y sus prioridades, de modo que también permita conocer con exactitud cuáles serán implementadas en la próxima liberación. Debe ser negociado y elaborado en forma conjunta entre el cliente y el equipo desarrollador durante las reuniones de planificación con la idea de hacer entregas frecuentes para obtener una mayor retroalimentación [36].

La **Tabla 4** muestra el plan de entregas pertinente:

Entregable	Iteración	Fin Iteración
-Configurar parámetros de conexión a la BD. -Editar tablas del esquema de datos. -Editar filtros por parámetro del esquema de datos. -Editar filtros por referencia del esquema.	1	Diciembre/2014
-Salvar esquema de datos. -Cargar esquema de datos. -Generar propuesta de plan de ejecución.	2	Abril/2015

-Editar propuesta de plan de ejecución. -Salvar plan de ejecución. -Cargar plan de ejecución.		
-Definir conexiones a la BD maestra y BD esclava. -Analizar operaciones de cambio. -Aplicar operaciones de cambio sobre BD esclava. -Generar permisos sobre esquema.	3	Mayo/2015

Tabla 4. Plan de entregas.

2.8 Diseño

El diseño juega un papel importante en el ciclo de vida del software, ya que permite indicar la forma en que será construido el producto y crea un punto de partida para las fases posteriores de implementación y prueba. La metodología XP por su parte, sugiere que se deben conseguir diseños sencillos y fácilmente entendibles a la hora de implementar. De esta forma costará menos tiempo y esfuerzo el desarrollo de los sistemas.

2.8.1 Diagrama de clases de la herramienta

Los diagramas de clases son los más utilizados en el modelado de sistemas orientados a objetos, puesto que muestran un conjunto de clases, interfaces y colaboraciones, así como sus relaciones. Se utilizan para modelar la vista del diseño estática del software, especificar y documentar modelos estructurales, y para construir sistemas ejecutables, aplicando ingeniería directa e inversa.

A continuación se muestra un fragmento del diagrama de clases, en el que solo se incluyeron las clases más relevantes de la herramienta (ver **Figura 11**). El siguiente diagrama muestra las relaciones entre dichas clases.

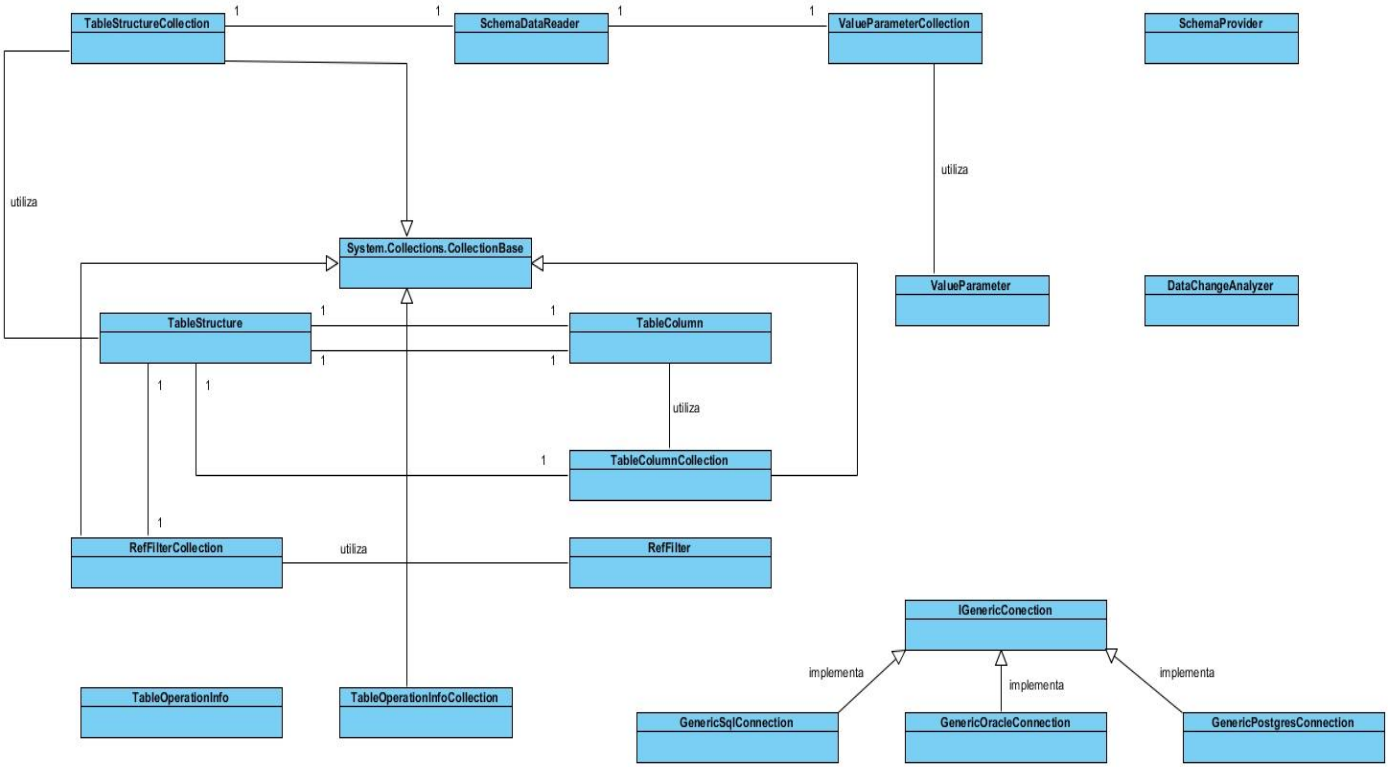

Figura 11. Fragmento del diagrama de clases del sistema.

2.8.2 Tarjetas CRC

La descripción de las tarjetas CRC (clases-responsabilidades-colaboradores) aporta un medio sencillo de identificar y organizar las clases que resulten relevantes al sistema o requisitos del producto. Un modelo CRC es realmente una colección de tarjetas índice estándar que representan clases y están divididas en tres secciones. A lo largo de la cabecera de la tarjeta se escribe el nombre de la clase, en el cuerpo se listan las responsabilidades de la clase a la izquierda, y a la derecha los colaboradores. Las responsabilidades son los atributos y operaciones relevantes para las clases, mientras que los colaboradores son aquellas clases necesarias para completar una responsabilidad [37].

Las **Tablas 5 y 6** muestran las tarjetas CRC de las clases *SchemaDataReader* y *TableStructure*.

Clase *SchemaDataReader*	
Responsabilidades	**Colaboradores**
GetData	*TableStructure, TableStructureCollection, ValueParameterCollection*

Tabla 5. CRC SchemaDataReader.

Clase *TableStructure*	
Responsabilidades	**Colaboradores**
CreateDataTable	*TableColumnCollection, TableColumn*
Fill	*TableColumnCollection, DataTableCollection, RefFilterCollection*
CreateRelation	*RefFilterCollection, TableColumnCollection*
GetPrimaryKeyColums	*TableColumn, TableColumnCollection*

Tabla 6. CRC TableStructure

2.8.3 Arquitectura

En el presente trabajo de diploma se propone como arquitectura base para la organización estructural de la herramienta, una modelo en capas (ver **Figura 12**). La arquitectura basada en capas se enfoca en la

distribución de roles y responsabilidades de forma jerárquica proveyendo una forma muy efectiva de separación de responsabilidades [38].

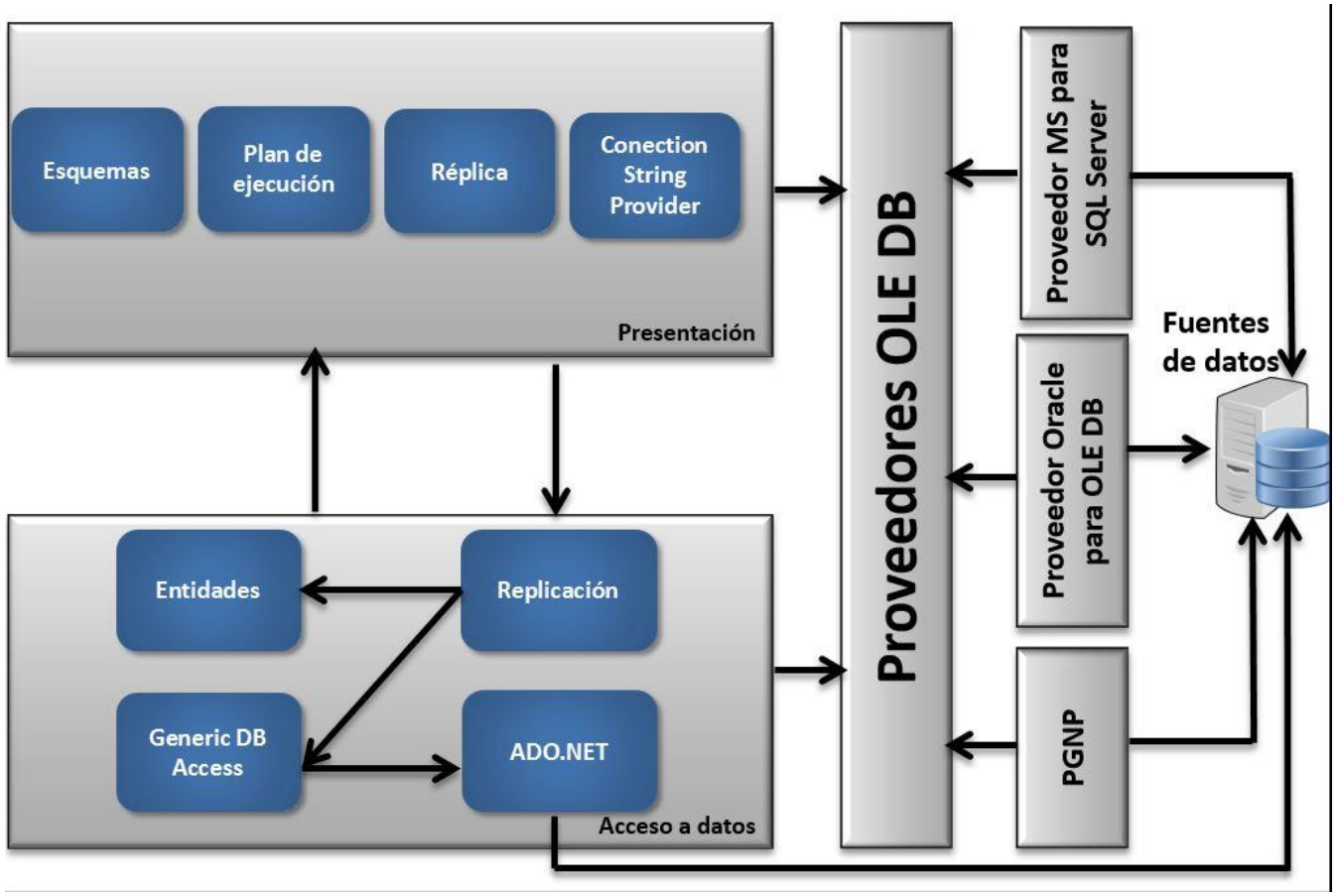

Figura 12. Arquitectura de software.

La arquitectura que presenta la herramienta *Schema Tool* identifica como capas, las siguientes:

Capa de Presentación: Representa la interfaz gráfica para la interacción con los componentes de la herramienta. Es donde se encuentran todas las interfaces de usuario que permitirán los procesos de creación y gestión de esquemas y planes de ejecución, así como la replicación de datos. De igual manera el componente *ConnectiongStringProvider* garantizará a través de su comunicación con los diferentes proveedores OLE DB disponibles, la generación de interfaces dinámicas para conformar las cadenas de conexión a cada uno de los SGBD.

Capa de Acceso a Datos: Será la encargada de efectuar el proceso de extracción y comparación de datos de las BD maestra y esclava, así como de la aplicación de los cambios. El componente de replicación se auxiliará de las entidades para decidir qué tablas y qué tipo de operaciones intervendrán en el proceso. El proceso de sincronización podrá efectuarse para diferentes fuentes de datos gracias al uso del componente de acceso genérico basado en el *framework* ADO.NET y los proveedores OLE DB disponibles, de esta forma se podrán integrar de manera extensible otros SGBD. Los proveedores OLE DB integrados al sistema son PGNP[18] para PostgreSQL, Oracle Provider for OLE DB para Oracle y el Microsoft Provider para SQL Server.

2.8.4 Patrones de diseño

Los patrones de diseño son el esqueleto de las soluciones a problemas comunes en el desarrollo de software. Brindan una solución ya probada y documentada a situaciones en contextos similares [39]. Estos modelos codifican buenos principios y sugerencias relacionados con la asignación de responsabilidades, basados en la recopilación del conocimiento de los expertos en desarrollo de software [40].

Patrones GRASP

Los patrones GRASP (Patrones Generales de Software para Asignar Responsabilidades) describen los principios fundamentales de la asignación de responsabilidades a objetos [41]. A continuación se describen los patrones de este tipo utilizados en el desarrollo de la solución:

Experto: Este patrón se utiliza más que cualquier otro como principio básico de la asignación de responsabilidades en el diseño orientado a objetos, de esta manera se conserva el encapsulamiento y se garantiza que los objetos se valgan de su propia información para llevar a cabo sus tareas [41]. De este modo se mantiene el encapsulamiento, los objetos utilizan su propia información para llevar a cabo sus tareas, se distribuye el comportamiento entre las clases que contienen la información requerida y los sistemas son más fáciles de entender y mantener.

El uso del patrón experto en la solución propuesta se evidencia en múltiples escenarios entre los que se pueden mencionar por ejemplo la clase *TableStructure*, que contiene toda la información relativa a una

[18] Traducido del idioma inglés *PostgreSQL Native OLEDB Provider*: Proveedor OLEDB nativo para PostgreSQL

tabla de una BD (nombre, esquema que la contiene, lista de columnas, relación de filtros por parámetros o referencia) y usa dicha información en la solución de sus principales procesos como son llenar la tabla a través del método *Fill,* obtener las llaves primarias a través del método *GetPrimaryKeyColums* o establecer las relaciones mediante llaves foráneas a través del método *CreateRelation.*

Creador: El patrón Creador guía la asignación de responsabilidades relacionadas con la creación de objetos, tarea muy frecuente en los sistemas orientados a objetos. Plantea que una instancia debe ser creada por la clase que tiene la información necesaria para ello.

El patrón creador ha sido utilizado en toda la solución propuesta. Puede ejemplificarse su presencia en la clase *SchemaProvider*, durante el proceso de creación de un *TableStructure,* a partir de la recopilación de la información necesaria para ello (ver **Figura 13**).

```
        2 referencias
        public TableStructure GetTableStructure(DataRow row)
        {
            string dbSchemaName = null;
            if (row.Table.Columns.Contains("TABLE_SCHEMA"))
                dbSchemaName = row["TABLE_SCHEMA"].ToString();

            string tableName = row["TABLE_NAME"].ToString();
            return new TableStructure(dbSchemaName, tableName, GetColumns(dbSchemaName, tableName));
        }
    }
}
```

Figura 13. Evidencia en el código del patrón Creador.

Bajo Acoplamiento: El acoplamiento es una medida de la fuerza con que una clase está conectada a otras clases, con que las conoce y con que recurre a ellas. Una clase con bajo o débil acoplamiento no depende de muchas otras [41]. El Bajo Acoplamiento soporta el diseño de clases más independientes, que reducen el impacto de los cambios, y también más reutilizables (Ver **Figura 14**).

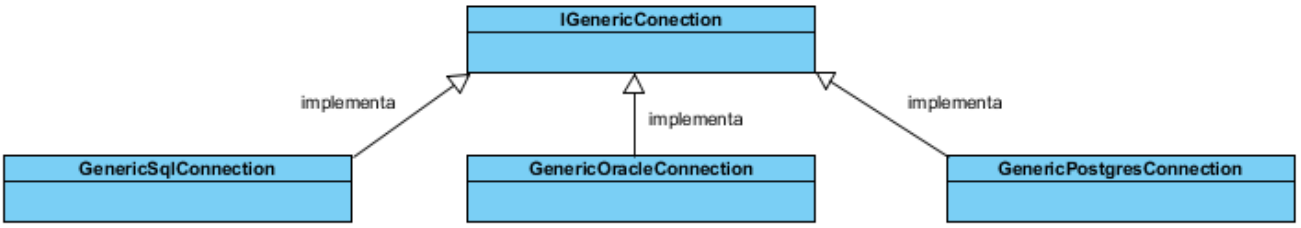

Figura 14. Diagrama de clases que evidencia el patrón bajo acoplamiento.

La interfaz de programación *IGenericConnection* define todas las prestaciones de las diferentes clases que implementan la conexión a los gestores de bases de datos soportados, de modo que al instanciar objetos del tipo de la interfaz, se minimiza la repercusión que pueden tener los cambios efectuados en cada uno de los conectores, así como la incorporación de nuevos conectores en el futuro (ver **Figuras 15 y 16**).

```
2 referencias
private IGenericConnection CreateConnection()
{
    IGenericConnection con = null;
    string conStr = txtConStr.Text;

    if (rdbOracle.Checked)
        con = new GenericOracleConnection( DeleteProvider(conStr) );
    else if (rdbSqlServer.Checked)
        con = new GenericSqlConnection( DeleteProvider(conStr) );
    else if (rdbMAcces.Checked)
        con = new GenericMicrosoftAccesConnection( conStr );
    else if (rdbOther.Checked)
        con = new GenericCustomOldbConnection( conStr, txtPrmPrefix.Text, chbxSupportDbSchema.Checked );

    return con;
}
```

Figura 15. Evidencia en el código del patrón Bajo Acoplamiento.

Una conexión puede abrirse o cerrarse, además de otras operaciones que pueden efectuarse a través de ellas, sin importar específicamente la clase a través de la cual está implementada o la manera en que lo

hacen, denotando un bajo acoplamiento en la solución. Esto se propicia además en la obtención de los esquemas de datos a través de los diferentes proveedores OLE DB utilizados.

```
1 referencia
private void btnTest_Click(object sender, System.EventArgs e)
{
    IGenericConnection con = CreateConnection();
    try
    {
        con.Connection.Open();
        con.Connection.Close();
        MessageBox.Show("Connection OK...");
    }
    catch (Exception ex)
    {
        MessageBox.Show("Error \n" + ex.ToString());
    }
}
```

Figura 16. Evidencia en el código del patrón Bajo Acoplamiento.

Alta cohesión: En la perspectiva del diseño orientado a objetos, la cohesión (o más exactamente, la cohesión funcional) es una medida de cuán relacionadas y enfocadas están las responsabilidades de una clase. Este patrón plantea que la información que almacena una clase debe ser coherente y debe estar en la medida de lo posible, relacionada con la clase. Propone además, que no se debe saturar una clase de métodos, sino asignar las responsabilidades a cada clase correspondiendo a la información que almacena [41].

En la solución propuesta se ha propiciado que todas las clases de la herramienta tengan una alta cohesión, delegando a cada una solo las responsabilidades que le corresponden. Esto puede ejemplificarse en el proceso de obtención de datos de un esquema en la clase *SchemaDataReader.* Como muestra el fragmento de código (ver **Figura 17**), el esquema no llena todas las tablas contenidas en sí, sino que delega en ellas la responsabilidad de llenarse de manera independiente y establecer sus relaciones con otras tablas a través de sus propias llaves foráneas.

```
2 referencias
public DataSet GetData(GenericDbAccess.IGenericConnection gc, bool includeRelations)
{
    try
    {

        gc.Connection.Open();
        int C = _Tables.Count;
        for (int i = 0; i < C; i++)
        {
            TableStructure tbl = _Tables[i];
            tbl.Fill(ds, gc, _Parameters);
            if (includeRelations)
                tbl.CreateRelation(ds);
        }
    }
    catch (Exception e)
    {
        throw e;
    }
    finally
    {
        if (manageConn)
            gc.Connection.Close();
    }
    return ds;
}
```

Figura 17. Evidencia en el código del patrón alta cohesión.

Patrones GOF[19]

Estos son los patrones en el campo del diseño orientado a objetos más conocidos y usados en la actualidad. Describen las formas en las que pueden ser organizados los objetos para trabajar unos con otros [41]. Seguidamente se describen los patrones de este tipo que se utilizan en la actualización de la herramienta:

Singleton: Este patrón garantiza la existencia de una única instancia para una clase y la creación de un mecanismo de acceso global a dicha instancia. El uso de este patrón en la solución propuesta se evidencia en el hecho de que las principales interfaces de usuario de la herramienta comparten una

[19] Traducido del idioma inglés *Gang of Four:* Grupo de cuatro.

misma instancia de la clase *SchemaDataReader*, la cual representa al esquema de extracción de datos que puede crearse, editarse y/o usarse para un ejercicio de sincronización. Esta única instancia se inicializa en la interfaz principal de la herramienta y va siendo transferida para su edición o su uso hacia el resto de interfaces que lo requieren. También se evidencia el uso del patrón en lo referente a los objetos que garantizan la conexión a las diferentes fuentes de datos, los cuales son instanciados una única vez y utilizados luego en múltiples escenarios.

Facade: Este patrón provee de una interfaz unificada para manejar un conjunto de objetos en un subsistema, garantizando un acoplamiento más débil, aumentando la independencia y la portabilidad. Permite además una reducción del número de objetos a utilizar, ya que, en lugar de necesitar un objeto por cada subclase implicada, solo se necesita el objeto Facade. En la solución propuesta puede evidenciarse el uso de este patrón mediante la definición e implementación de las conexiones a los diferentes gestores de bases de datos. El elemento Facade está representado por la interfaz IGenericConnection, al tiempo que las subclases que la implementan son *GenericOracleConnection*, *GenericSqlConnection* y *GenericPostgreConnection*. El uso del patrón permite abstraerse del nivel de complejidad de la conexión a cada gestor de manera independiente, donde intervienen incluso bibliotecas de distintos proveedores.

Conclusiones parciales

El modelo de dominio definido permitió visualizar los principales conceptos manejados en el contexto en que se desarrolla *Schema Tool,* lo que facilitó la identificación de los requisitos funcionales y no funcionales, que fueron descritos en las historias de usuario. La arquitectura en capas propuesta, garantiza la distribución de responsabilidades dentro de la herramienta. Los patrones de diseño utilizados, garantizan el desarrollo de una aplicación que cumpla con las buenas prácticas que exigen estos patrones. El rediseño de la herramienta permitirá la integración de manera extensible con los SGBD relacionales Oracle, Microsoft SQL Server y PostgreSQL, propiciando la incorporación futura de nuevos SGBD.

Capítulo 3. Implementación y pruebas de la herramienta *Schema Tool*

Introducción

En este capítulo se describen los elementos asociados a la implementación y validación de la nueva versión de la herramienta. Se definen los estándares de codificación que debe cumplir el equipo de trabajo, se muestran el diagrama de componentes y el de despliegue. Además se diseñan las pruebas necesarias para comprobar el correcto funcionamiento de la herramienta y se muestran los resultados de la ejecución de las mismas.

3.1 Estándares de codificación

Entre las prácticas a emplear durante el proceso de desarrollo de software que plantea la metodología XP se encuentran la refactorización del código y la propiedad compartida de este de forma que todo el personal pueda corregir cualquier parte del producto. Para complementar estas prácticas, la metodología enfatiza en que la comunicación de los programadores es a través del código, por lo cual es necesario que se sigan ciertos estándares de programación que le provean legibilidad. Un estándar de codificación completo comprende todos los aspectos de la generación de código. Un código fuente completo debe mostrar un estilo armonioso, como si un único programador hubiera escrito todo el código de una sola vez. Al comenzar un proyecto de software, se debe instaurar un estándar de codificación para asegurarse de que todos los programadores del proyecto trabajen de forma coordinada [42].

3.1.1 Estilos para la capitalización de los identificadores

Para la capitalización de los identificadores se utilizaron los siguientes convenios:

Pascal: La primera letra en el identificador y la primera letra de cada subsiguiente palabra concatenada se capitalizan. Este estilo se evidencia en la clase *SchemaDataReade*r.

Camel: La primera letra en el identificador se pone en minúscula y la primera letra de cada subsiguiente palabra concatenada en mayúscula. Este estilo se evidencia en el atributo *columnName*.

3.1.2 Comprensión y legibilidad del código

Con el objetivo de incrementar la legibilidad del código se comentaron todas las declaraciones de clases y funciones más complejas. Los atributos, propiedades y constructores aparecen en la parte superior de las clases, mientras que los métodos privados y públicos están a continuación. Además se organizó el código de forma estructurada en bloques de código, para una mejor comprensión del mismo.

3.2 Diagrama de componentes

El diagrama de componentes representa la división del software en partes más pequeñas denominados componentes. Estos describen los elementos de la herramienta, ya sean código fuente, binario o ejecutable, la organización que presentan y las relaciones entre ellos que se utilizan para indicar que un componente usa los servicios ofrecidos por otro componente [43]. A continuación se describe el diagrama de componentes de la herramienta (ver **Figura 18**).

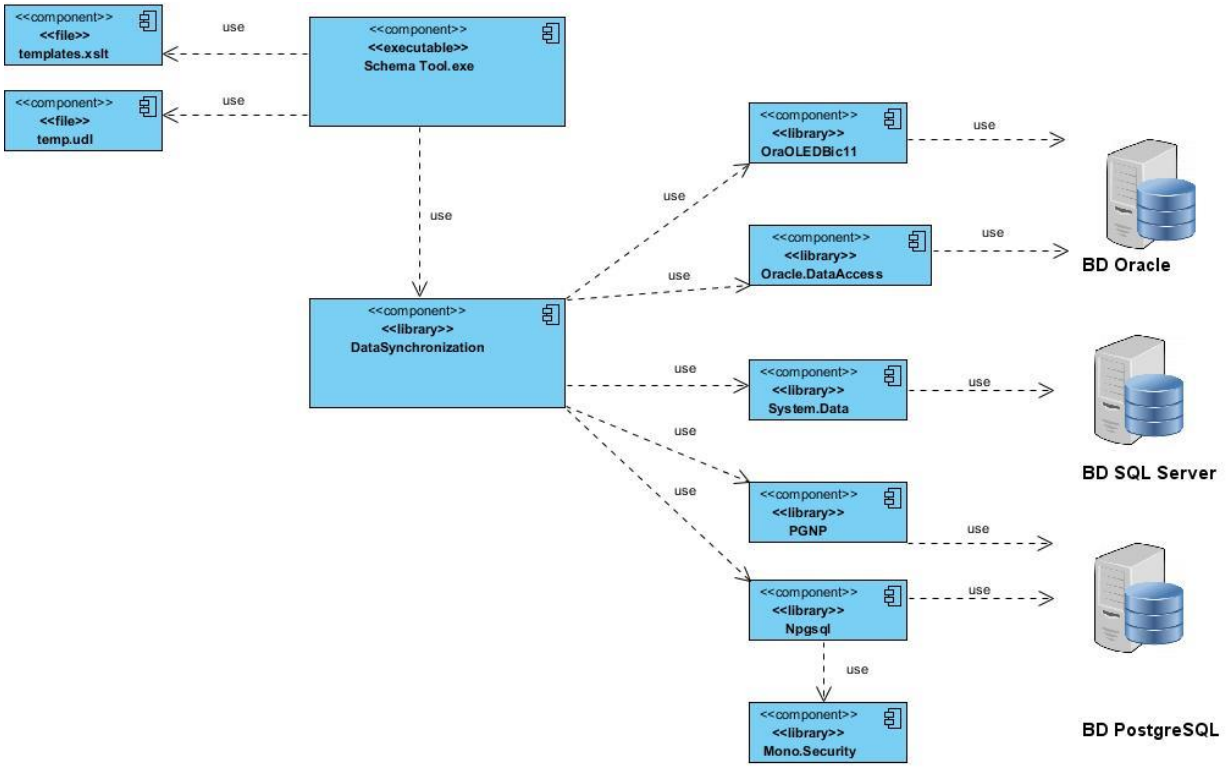

Figura 18. Diagrama de componentes

La herramienta está compuesta básicamente por dos ensamblados principales. El ejecutable *SchemaTool.*exe que contiene la capa de presentación y la biblioteca *DataSynchronization*.dll que contiene toda la lógica inherente al proceso de replicación. El *SchemaTool* usa al archivo *temp*.udl que permite listar todos los proveedores de OLE DB instalados en la estación de trabajo, posibilitando el uso de interfaces dinámicas para el establecimiento de la conexión a distintos SGBD. Por su parte el componente *DataSyncronizacion*.dll es el rector de todo el proceso de extracción, comparación de datos y aplicación de cambios en la BD esclava, proceso para el cual se auxilia de diversas bibliotecas provistas por los diferentes gestores: *OracleOledbic11*.dll y *OracleDataAccess* para el caso de Oracle, PGNP y *Npgsql.*dll para PostgreSQL y *System.Data*.dll, perteneciente al *framework* .NET para el caso de SQL Server. Mediante el uso de todas estas bibliotecas se accede a las BD correspondientes.

3.3 Diagrama de despliegue

El diagrama de despliegue describe la distribución física de un software en un ambiente de producción o prueba (ver **Figura 19**).

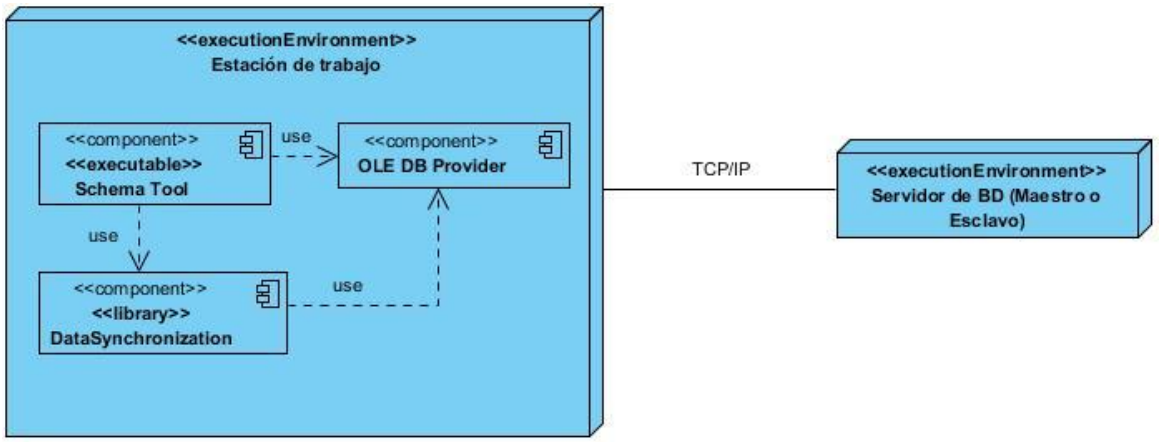

Figura 19. Diagrama de despliegue.

La herramienta se utilizará en una estación de trabajo, donde deberán estar instalados los diversos proveedores de OLE DB, que permitirán la conexión mediante el protocolo TCP/IP con los diferentes servidores de BD. Es importante aclarar que una vez desplegada la nueva versión de la herramienta, será

posible la replicación entre bases de datos de diferentes SGBD, que se pueden encontrar alojadas en múltiples servidores

3.4 Pruebas

Las pruebas de software son la verificación del funcionamiento de un determinado software. Son una serie de acciones que se realizan para encontrar los posibles fallos o vulnerabilidades en la implementación y constituyen un elemento crítico para garantizar el correcto funcionamiento en determinado sistema [44].

3.4.1 Pruebas unitarias

Las pruebas unitarias son escritas por los programadores antes de comenzar la codificación. El objetivo de estas pruebas es aislar pequeñas e individuales porciones del código para verificar que no tengan errores. Para la realización de estas pruebas se utilizó un Add-in de Visual Studio llamado Resharper (R#) que permite detectar errores y brinda soluciones instantáneas para corregirlos. Los resultados de estas pruebas fueron satisfactorios, verificándose de esta manera que en las principales funcionalidades se obtiene en cada caso la respuesta esperada.

3.4.2 Pruebas de aceptación

Las pruebas de aceptación se realizan para determinar el nivel de satisfacción del cliente final con el producto informático. Para que estas pruebas sean lo más objetivas posible, no deben ser diseñadas por los ingenieros de software que desarrollan el producto, sino por el cliente. Las pruebas de aceptación se realizan a partir de las historias de usuario, cada historia de usuario se convierte en un caso de prueba en el que el cliente especifica los puntos a probar, aunque una historia de usuario pudiera tener más de una prueba de aceptación, las que sean necesarias para garantizar la calidad del producto final con el cumplimiento de los requisitos especificados por el cliente. Una historia de usuario no se puede considerar terminada hasta tanto pase correctamente todas las pruebas de aceptación [45]. A continuación se muestran dos de los casos de prueba (ver **Tablas 7 y 8**) que corresponden a las historias de usuario más significativas.

Caso de Prueba de Aceptación	
Código: CPA_12	Historia de usuario: HU_12 Analizar operaciones de cambio.
Responsable: Alexander López Valladares	
Descripción: Se analizan las operaciones de cambio que van a ocurrir durante el proceso de replicación.	
Condiciones de ejecución: • Debe existir una conexión a las BD maestra y esclava y además un esquema de datos y un plan de ejecución creados previamente por la herramienta.	
Entrada / pasos de ejecución: • Dar clic en el botón Analyze Operations. • Dar clic en el botón Test. • Dar clic en el botón Aceptar.	
Resultado esperado: Se listan las operaciones DML que van a ocurrir en la BD esclava y las mismas son probadas. La herramienta muestra un mensaje que dice “Operations can be applied with 0 errors”.	
Evaluación de la prueba: Prueba satisfactoria.	

Tabla 7. CPA_12 Analizar operaciones de cambio.

Caso de Prueba de Aceptación	
Código: CPA_13	Historia de usuario: HU_13 Aplicar operaciones de cambio en la BD esclava.
Responsable: Alexander López Valladares	
Descripción: Se aplican las operaciones de cambio que van a ocurrir en la BD esclava durante el proceso de sincronización.	

Condiciones de ejecución: • Debe existir la conexión entre las BD maestra y esclava y además un esquema de datos y un plan de ejecución creados previamente por la herramienta. Deben estar previamente listadas las operaciones de cambio
Entrada / pasos de ejecución: • Dar clic en el botón Apply. • Dar clic en el botón Aceptar. • Dar clic en el botón Analyze Operations.
Resultado esperado: Ocurren las operaciones especificadas en la BD esclava. Una vez efectuada la comparación nuevamente, la herramienta mostrará un mensaje que diga: “The master and the slave have been already syncronized.”
Evaluación de la prueba: Prueba satisfactoria

Tabla 8. CPA_13 Aplicar operaciones de cambio en la BD esclava.

3.4.3 Pruebas de rendimiento

En lo que respecta al rendimiento de la herramienta se efectuaron pruebas para validar los tiempos de respuesta en los procesos de comparación de la información y aplicación de los cambios para los diferentes SGBD (ver **Figura 20**).

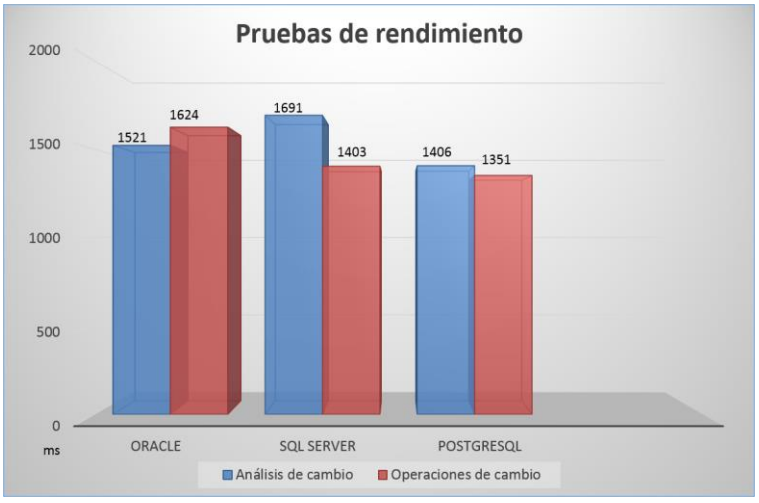

Figura 20. Resultados de las pruebas de rendimiento.

Como es posible observar los tiempos de respuesta para un proceso de sincronización superior a las 1000 tuplas, son inferiores a los 2 segundos tanto para PostgreSQL, Oracle, como para SQL Server. Se valida de esta forma uno de los principales requisitos no funcionales de la herramienta.

3.4.4 Resultados de las pruebas

Después de haber realizado las pruebas de aceptación de la herramienta a partir de los 14 casos de pruebas definidos durante las tres iteraciones de codificaciones planificadas, se detectaron en cada una de estas 3, 5 y 2 no conformidades respectivamente. Esto equivale a un total de 10 no conformidades, que revelaban errores en la codificación que en la mayoría de los casos no incurrían significativamente en las respuestas esperadas (**ver Figura 21**).

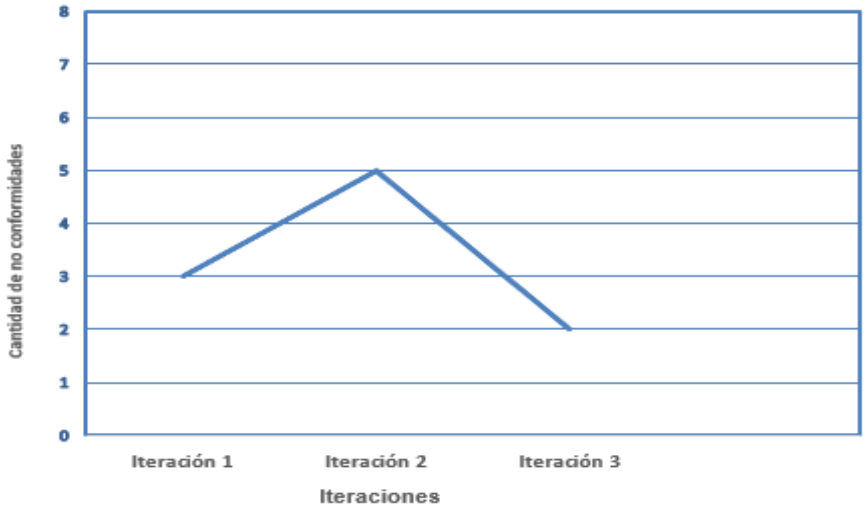

Figura 21. Resultados de las pruebas de aceptación.

Las dificultades fueron solucionadas en un corto plazo, antes de pasar a la iteración posterior. Una vez obtenido el producto final, se verificó el correcto funcionamiento de la herramienta sin detectarse ya nuevas no conformidades, demostrando el correcto cumplimiento de todos los requisitos solicitados por el cliente.

Conclusiones parciales

Durante la etapa de implementación, la representación del diagrama de componentes, el diagrama de despliegue y las interfaces de pruebas creadas, permitieron visualizar la estructura y funcionamiento de la herramienta. Las pruebas efectuadas a las entregas resultantes de cada una de las iteraciones en las que se desarrolló la herramienta, facilitaron la detección y corrección de varias no conformidades que influían en el mal funcionamiento de la herramienta. A partir de las pruebas de aceptación realizadas se pudo dar por terminada la implementación de cada historia de usuario garantizando el correcto funcionamiento de la herramienta

Conclusiones generales

La aplicación de los métodos teóricos y el análisis bibliográfico permitieron concretar el marco teórico referente a la replicación de datos en las bases de datos distribuidas.

El análisis de las principales herramientas existentes para la replicación de datos, permitió constatar los requisitos que se debían considerar en la nueva versión de *Schema Tool*, así como los gestores de base de datos a los que se debía integrar.

Con el rediseño de la capa de acceso a datos y la migración a la versión 4.5 del *framework* .NET, se obtuvo una herramienta extensible que puede ser utilizada para el soporte y configuración de sistemas distribuidos en nuevos entornos.

Las pruebas de software realizadas permitieron validar la correctitud de las funcionalidades implementadas, comprobando que la herramienta cumple con el nivel de calidad requerido para su funcionamiento en un ambiente real.

Recomendaciones

Después de cumplido el objetivo general de la presente investigación, valorando algunos puntos derivados de la misma y con el propósito de mejorar la solución propuesta, se recomienda:

1. Incorporar una capa intermedia que guíe las interacciones entre las capas de interfaz y acceso a datos, mediante la implementación del patrón Controlador, para propiciar un mayor nivel de independencia de los componentes.

2. Implementar un nuevo proveedor OLE DB para el sistema gestor de bases de datos PostgreSQL, de acuerdo a las especificaciones del estándar, para evitar el uso del proveedor propietario PGNP.

3. Integrar nuevos sistemas gestores de bases de datos relacionales a la herramienta, aprovechando su extensibilidad, para seguir ampliando su entorno de aplicación.

4. Implementar una arquitectura basada en servicios que exponga las prestaciones de la biblioteca DataSynchronization, para permitir su uso desde distintos lenguajes y plataformas.

5. Realizar la migración de la herramienta *Schema Tool* a software libre.

Referencias bibliográficas

[1] J. P. Tarancón, “Bases de datos relacionales y el modelo entidad-relación Definición de BD.” p. 32, 2002.

[2] L. I. C. Rosa and M. Mato, *BASES DE DATOS*, Edición 1. Cuba: , 1999, p. 51.

[3] V. Ramirez Santana and I. Deither Cao, “CONFIGURACIÓN SCHEMA TOOL.” Cuba, p. 92, 2005.

[4] J. L. Perez Ramos, “Modelo de replicación de datos en sistemas distribuidos de bases de datos relacionales,” 2009.

[5] G. Gardarin and P. Valduriez, *Relational Databases, Chapter 1*. Addison-Wesley Professional, 1988, p. 12.

[6] Tecnomaestros, “Bases de datos Distribuidas.” [Online]. Available: http://tecnomaestros.awardspace.com/bases_datos_distribuidas.php. [Accessed: 15-Sep-2014].

[7] A. Silberschatz, H. F. Korth, and S. Sudarshan, *Fundamentos de bases de datos*, Edición 4. España: editorial-Interamericana de España, S.A.U., 2002, p. 787.

[8] G. Gardarin and P. Valduriez, *Relational Databases and Knowledge Bases*, Edición 1. Paris: editorial-Addison-Wesley Publishing Company, 1989, p. 237.

[9] J. L. Perez Ramos, “Modelo de Replicación de Datos en Sistemas Distribuidos de Bases de Datos Relacionales.,” Universidad Mayor de San Andrés, 2006.

[10] O. N. Dresky and L. B. Carlos, “Modulo de Reko para la solución de conflictos.,” Universidad de Ciencias Informáticas, 2010.

[11] R. Z. Ramírez and C. F. G. Medio, “Sistemas Gestores de Bases de Datos,” p. 9, 2008.

[12] D. Engine, “Principales SGBD.” .

[13] E. Cornelio Rivero, C. Guardia Rivas, and J. C. Reig Hernandez, *Bases de Datos Relacionales*. 2004, p. 664.

[14] D. De Postgresql and T. Lockhart, *Tutorial de PostgreSQL*. 1996.

[15] PostgreSQL Development Group, “The PostgreSQL Global Development Group,” 2013.

[16] PostgreSQL, “PostgreSQL 8.4.2 Documentation,” 2010. [Online]. Available: http://www.postgresql.org/docs/current/static/information-schema.html. [Accessed: 25-Dec-2014].

[17] Microsoft, “Microsoft SQL Server.” [Online]. Available: http://www.microsoft.com/sql/. [Accessed: 24-Dec-2014].

[18] Microsoft, “OLE DB.” [Online]. Available: http://msdn.microsoft.com/data/. [Accessed: 13-Apr-2015].

[19] Altova, “Altova.” [Online]. Available: www.altova.com. [Accessed: 15-Dec-2014].

[20] Altova, “Altova DatabaseSpy,” 2013. [Online]. Available: www.altova.com/es/databasespy.html. [Accessed: 08-Dec-2014].

[21] Daffodil, “Introducing Daffodil DB Replicator v1.” [Online]. Available: http://www.theserverside.com/. [Accessed: 13-Nov-2014].

[22] L. Edgardo, M. Rodriguez, and D. P. Alfonso, “Módulo de Reko para la replicación entre bases de datos con diferentes estructuras,” vol. 5, no. 4, pp. 1–11, 2012.

[23] A. A. Figueroa, Roberth G., Camilo J. Solís, Cabrera, *Metodologías tradicionales VS Metodologías ágiles*. p. 9.

[24] E. H. Orallo, *Unificado de Modelado (UML)*. 2002, pp. 1–6.

[25] “Herramientas de modelado.” [Online]. Available: http://www.alegsa.com.ar/Dic/herramienta. [Accessed: 11-Jan-2015].

[26] “Visual Paradigm.” [Online]. Available: http://www.identi.li/index.php?topic=168979. [Accessed: 11-Jan-2015].

[27] “Plataforma de desarrollo.” [Online]. Available: http://gnustep.wordpress.com/. [Accessed: 13-Jan-2015].

[28] .NET, “.NET.” [Online]. Available: http://www.dcs.ed.ac.uk/home/stg/Psharp/. [Accessed: 13-May-2015].

[29] “Lenguaje de programación.” [Online]. Available: http://definicion.de/lenguaje-de-programacion/. [Accessed: 12-Jan-2015].

[30] Microsoft, “C#.” [Online]. Available: http://msdn.microsoft.com/es-es/library/aa287483(v=vs.71).aspx. [Accessed: 20-Jan-2015].

[31] “IDE.” [Online]. Available: http://www.ecured.cu/index.php/. [Accessed: 12-Jan-2015].

[32] “Visual Studio.” [Online]. Available: http://www.visualstudio.com/es-es. [Accessed: 12-Jan-2015].

[33] “ADO.NET.” [Online]. Available: https://msdn.microsoft.com/en-us/library/e80y5yhx(v=vs.110).aspx. [Accessed: 17-Feb-2015].

[34] P. Gervás, “Requisitos no funcionales.” [Online]. Available: http://www.sistemas.edu.bo/lsalgado/sis3390/Requisitos/02 captura de requisitos.pdf. [Accessed: 02-Mar-2015].

[35] P. Letelier and M. del C. Penadés, *Métodologías ágiles para el desarrollo de software: eXtreme Programming (XP)*. España: , 2003.

[36] “Plan de entregas.” [Online]. Available: http://www.inf.utfsm.cl/~visconti/xp/. [Accessed: 03-Mar-2015].

[37] P. Roger S, *Ingeniería del Software*, vol. 5. 1997, p. 614.

[38] http://www.juanpelaez.com, “Arquitectura en capas.” [Online]. Available: http://www.juanpelaez.com/geek-stuff/arquitectura/arquitectura-basada-en-capas/. [Accessed: 13-May-2015].

[39] “Patrones de diseño,” 2013. [Online]. Available: http://moodle2.unid.edu.mx/dts_cursos_mdl/pos/TI/IS/AM/04/Patron_de_diseno.pdf. [Accessed: 20-May-2015].

[40] G. Erich, *Patrones de diseño*. Addison-Wesley Professional, 2000, p. 192.

[41] L. Craig, *UML y Patrones*, 2da edició. 2003, pp. 1–23.

[42] Microsoft, “Técnicas de codificación y prácticas de programación.” [Online]. Available: http://msdn.microsoft.com/. [Accessed: 04-May-2015].

[43] B. Campderrich Falgueras, *Ingeniería del Software*. 2003, p. 320.

[44] “Pruebas.” [Online]. Available: http://www.slideshare.net/aracelij/pruebas-de-software/. [Accessed: 06-May-2015].

[45] I. J. Joskowicz, *Reglas y Prácticas en eXtreme Programming*. 2008, p. 22.

[46] P. N. Cabanes, *Introducción a las Bases de Datos*, Edición 1. 2007, p. 46.

[47] U. Federico, S. Maria, and V. Proveedor, *BASES DE DATOS*, Edición 7. México: editorial-Pearson Education, 2001, p. 960.

[48] “BSD.” [Online]. Available: http://www.opensource.org/licenses/bsd-license.php. [Accessed: 23-Dec-2014].

Bibliografía consultada

H. S. Roberto, F. C. Carlos, and B. L. Pilar, *Metodología de la Investigación*. Mc Graw Hill, 2006, p. 882.

B. Mikael, *Thesis Projects. A guide for Students in Computer Science and Information Systems*, Second. 2008, p. 162.

Y. Vald, R. Mar, S. Calder, K. Hurtado, D. Asesora, M. Rosa, and A. Bosh, "Procedimiento general de pruebas de caja blanca aplicando la técnica del camino básico," 2007.

Q. S. Joaquín, "Patrones de diseño seguro para aplicaciones Web," 2010.

P. Roger S, *Ingeniería del Software. Un enfoque práctico*, Quinta., vol. 5. 1997, p. 614.

S. Ian, *Ingeniería del Software*. Addison-Wesley Professional, 2005, p. 712.

J. Ivar, B. Grady, and R. James, *El Proceso Unificado de Desarrollo de Software.pdf*. Addison-Wesley Professional, 2000, p. 458.

A. Y. A. Mohamed, "Aspect Oriented Requirements Engineering," vol. 3, no. 4, pp. 135–155, 2010.

J. José and M. Navarro, "Arquitecturas Software. Curso de Software basado en Componentes."

B. P. Lamancha, "Gestión de las Pruebas Funcionales," vol. 1, no. 4, pp. 2–7, 2007.

H. S. Roberto, F. C. Carlos, and B. L. Pilar, *Metodología de la Investigación*. Mc Graw Hill, 2006, p. 882.