Universidad de las Ciencias Informáticas

Curso 2009-2010

Facultad 6

Título: Módulo de Reko para la resolución de conflictos.

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

Autores:

Dresky Ortíz Noriega.

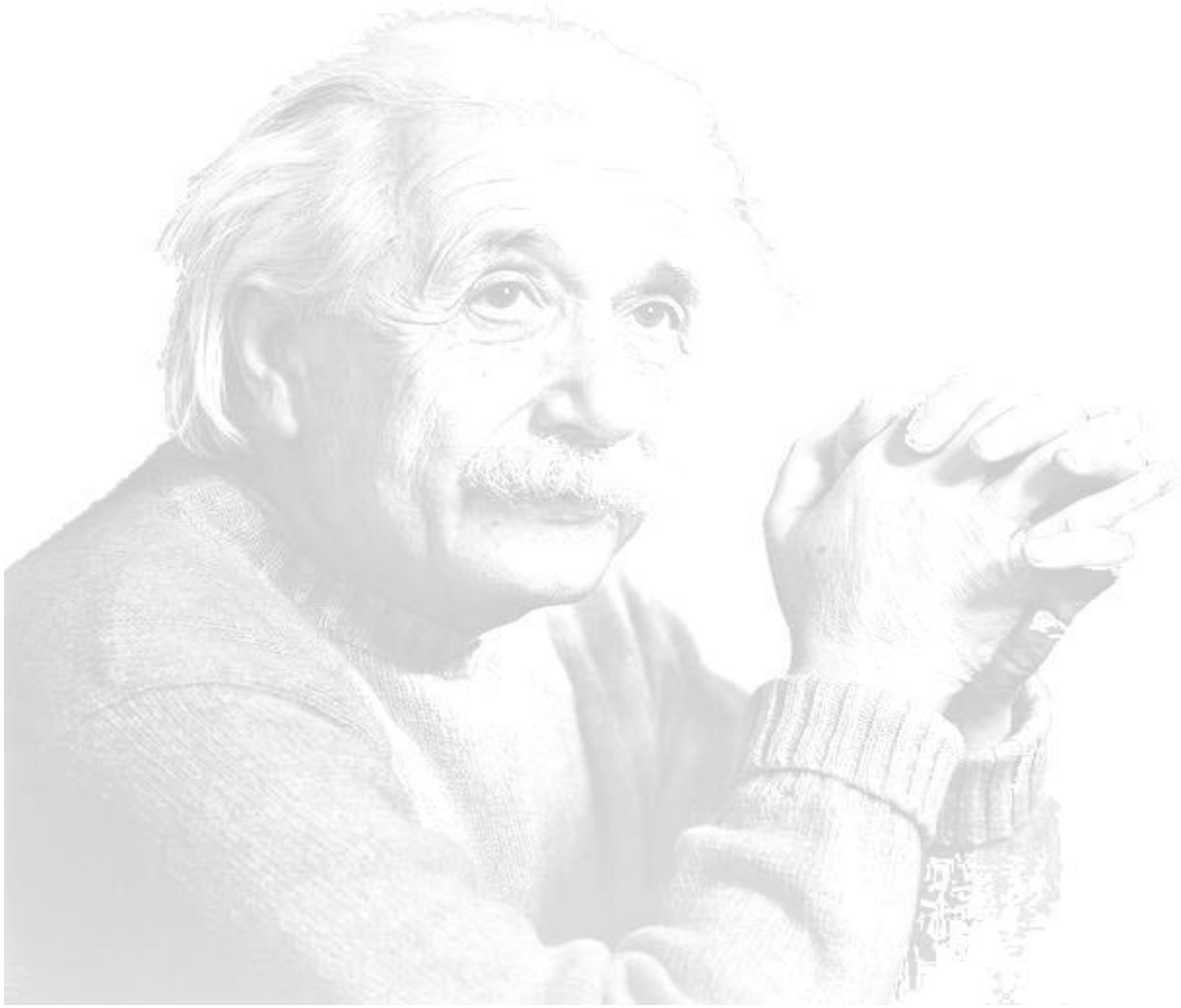
Carlos Javier López Boullón.

Tutores:

Lic. Yoemny González Almaguer.

Ing. Esley León Valdés.

Ciudad de la Habana, Julio 2010.



El misterio es la cosa más bonita que podemos experimentar. Es la fuente de todo arte y ciencia verdaderos.

Albert Einstein

Declaración de Autoría

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

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

_____ _____

Dresky Ortíz Noriega Carlos Javier López Boullón

_____ _____

Lic. Yoemny González Almaguer Ing. Esley León Valdés

Datos de Contacto

Lic. Yoemny González Almaguer
Licenciado en Ciencias de la Computación, Profesor Asistente.
E-mail: yoemny@uci.cu

Ing. Esley León Valdés
Ing. en Ciencias Informáticas, Instructor Recién Graduado.
E-mail: elvaldes@uci.cu

Agradecimientos:

Ante todo agradecemos al país donde vivo y a la Revolución, por darnos la posibilidad de estudiar en una universidad como esta y confiar siempre en nosotros como la tropa del futuro.

A nuestra familia y a nuestros amigos por siempre estar ahí cuando más los necesitamos.

A nuestros tutores por confiar en nosotros, tolerarnos y brindarnos su apoyo. En especial a Esley por su gran capacidad profesional y sus recomendaciones que nos guiaron en todo este proceso de tesis.

A Mena y a Eliecer por dedicarnos su tiempo y su apoyo incondicional, que a pesar de ser estudiantes como nosotros, fueron ellos quienes nos encaminaron, nos dieron las explicaciones más especificas y nos ayudaron a entender la estructura y el funcionamiento de Reko, así como lo que debíamos hacer y como trabajar.

A Pimentel que a pesar de estar lejos también nos brindó su apoyo y siempre mostrando preocupación por el logro de la aplicación.

Agradecer de manera especial a Rolando Bermúdez por su entrega en todo momento, por darme su apoyo cada vez que lo necesitaba; por transmitirnos y aportarnos sus amplios conocimientos de “Dojo”. Sin él no hubiese sido posible que el módulo de resolución de conflictos contara con una interfaz mucho más agradable y con buena presentación.

A Rachid por su dedicación y gran ayuda brindada en nuestros inicios, que más que un amigo se comportó como un hermano mayor exigiéndonos siempre como trabajar para aprovechar el tiempo; que debíamos investigar y estudiar para poder lograr nuestros objetivos.

Agradecemos enormemente a Andrés Miguel Bores (el mellizo) que en los últimos tiempos ha sido la persona más entregada a nuestra causa, ese amigo que siempre estaba ahí cada vez que lo necesitábamos, entregando incluso horas de sueño junto a nosotros.

Al tribunal por sus señalamientos y críticas, que nos ayudaron a guiar nuestra investigación previa por el camino correcto y elaborar la tesis con mejor calidad.

Al oponente, que fue como otro tutor para nosotros, por su cooperación y porque sus críticas, para nosotros constructivas, nos ayudaron a exigirnos cada día más en la confección de la tesis.

A todos los compañeros y amigos que nos ayudaron de una forma u otra, entre los que merecen la pena destacar a Yoan Manuel, Dairon, Daniel, Enrique, entre otros. A todos muchas Gracias.

Dedicatoria Dresky:

A mi madre y a mi padre por ser mi razón de ser, por darme todo su amor, por su entrega y sacrificio. Sólo le pido a la vida merecerlos y que me de salud para demostrarles cuanto los quiero.

A mi abuela Blasita, por ser mi segunda madre y la persona más importante de mi vida, por darme tanto cariño y por toda su dedicación durante todos estos años y enseñarme que a veces la vida nos puede golpear muy fuerte y ponernos en situaciones difíciles, pero que hay que sacar fuerzas y saber reponerse cuanto antes.

A mi abuelo por ser la persona que más admiro en el mundo, por sus buenos consejos, su preocupación y por estar siempre ahí cada vez que lo necesito y por regalarme tantos buenos momentos en su compañía, por eso hoy le puedo decir: “Pipo, te tengo una buena noticia…” y es la satisfacción de hoy poder concretar uno de mis sueños.

A mi hermano Heribe, a mi hermana Greisy y a todos mis primos, que son muy especiales para mí, que han sido mi inspiración para lograr este sueño hecho realidad quiero que sirva de ejemplo cuanto sacrificio es necesario hacer para regalarle a la familia y a uno mismo este momento de enorme satisfacción.

Dedico este logro a mis tíos que siempre han estado al tanto de mi carrera, en especial a mi tía Migday por su preocupación y cariño durante toda mi vida.

A mi familia de Marianao que siempre han estado ahí en todo momento y tanto apoyo me han dado, a mi abuela Vilma, que la adoro muchísimo le agradezco este logro tan importante; a mi tío Yusnel, que ha pesar de no poder lograr este sueño todavía, sé que en algún momento lo logrará.

A mi otra familia de Bayamo que tan buenos momentos pasé a su lado y que mucha muestra de preocupación y apoyo me han dado siempre a pesar de estar lejos.

A mi amigo y compañero de tesis por su optimismo, por demostrarme que siempre se puede y que todo se puede lograr. Gracias por soportarme todos estos meses.

A todos esos profesores durante mi vida de estudiante que me regalaron su granito de arena para que hoy llegara a ser quien soy.

A mis amistades de la UCI por darme tantos momentos de alegría y poder compartir juntos durante todo este tiempo, a esos amigos que por un motivo u otro hoy no están entre nosotros. A mis amigos del barrio, los del Infodrez, los del proyecto, los del apartamento y todos los que he conocido durante mi vida.

A todos los que hicieron posible que este sueño se hiciera realidad y a los que no también.

Dedicatoria Carlos Javier:

A mi abuela Gladys por saberme guiar por el buen camino y quiero que sepa que la quiero mucho y estoy eternamente agradecido por todo lo que ha hecho por mí, que todo lo que soy hoy es por ella. Mima te quiero

A mi abuelo Nico que hoy es su cumple y creo que este es el mejor regalo que le puedo dar en su día.

A mami lola y a boullón por lo tanto que me han querido y lo que han hecho por mí, a ustedes que son mis padres les dedico todo lo que soy.

A mi mamá y mi papá que de no ser por ellos no estuviera hoy aquí, los quiero cantidad y estoy súper orgulloso de ustedes.

A mi tío panchy que es como un padre para mi, estaré agradecido siempre por todo lo que me has ayudado y lo que has hecho por mi, espero que este orgulloso de mi como yo lo estoy de ti.

A mi tía Vicky, tío Ernesto y tío Ferna por quererme tanto y ser los mejores tíos del universo.

A mis hermanitos Alina Patricia, Nei, Danielita, tito, y Ana karliña, ojalá estén orgullosos de mi y les sirva de ejemplo. Los quiero cantidad.

A mi dúo de tesis y mi hermano Dresky, mano lo logramos, sin ti no hubiera sido posible.

A landy, Ingrid, erislan, el pache, el mikillo, el chupao, el pikete, Joan Maikel, pascual, el tavo, yamisel, noralys, alex, dayana, mayli, angel, los parecidos luis, dairon, el robe, el yuma por su apoyo, su confianza, por hacer de cada día el mejor de todos.

A todas mis amistades de aquí de la UCI, los que están y los que ya no están, la gente mía que se graduó el año pasado, la gente del bailoteo, mi piquete de la rueda de casino y la kisomba. Gracias por esos momentos inolvidables.

A todos los profes que me han impartido clases en estos años

A todos lo que me preguntaron por la calle: ¿dime pa´ cuando? ¿Cómo va la tesis esa? ¿nos graduamos?

A todos los que de una forma u otra han formado parte de mi vida en esta universidad y han hecho que este lugar sea inolvidable para mí.

Resumen

Reko es un replicador de bases de datos, el cual no cuenta con un mecanismo para solucionar los conflictos que pueden surgir en el momento de la réplica. El objetivo del presente trabajo es desarrollar un módulo que permita dar solución a los conflictos de replicación de forma manual y automática. En el primer caso el usuario tiene la oportunidad de cambiar la sentencia SQL que tiene problemas manualmente, y en la segunda opción el sistema de manera transparente al usuario tratará de dar solución haciendo uso de reglas creadas con anterioridad Si el conflicto capturado y listo para tratar coincide con alguna regla ya definida se le dará solución automáticamente. Para la definición de las reglas se utiliza Drools, un poderoso motor de reglas de negocio, utilizado con anterioridad en el Centro de Base de Datos de la universidad. Los conflictos pueden ocurrir cuando se trabaja en un ambiente de replicación que permite transferencia de datos entre diversas localizaciones en múltiples sitios. Existen varios tipos de conflictos a tener en cuenta durante la replicación de datos, de actualización, de unicidad, de supresión, por mencionar algunos; serán tratados mediante el módulo que se desarrolla para la aplicación Reko, ofreciendo una solución más completa.

Palabras Clave
Bases de datos, Conflictos, Reko, Réplica de datos.

Contenido

Introducción

La replicación de datos es el proceso de copiado y mantenimiento de objetos de la base de datos, tales como las tablas, en bases de datos múltiples y simulan un sistema de bases de datos distribuido. Los cambios hechos en un sitio son capturados y aplicados en cada una de las locaciones remotas.

Hoy día, con la tendencia al desarrollo de sistemas distribuidos, ha aumentado la necesidad de la utilización de réplicas. Las soluciones robustas en esta temática son muy costosas y las que existen en el software libre están incompletas y son muy difíciles de configurar.

Reko es un software de réplica de datos desarrollado en la Universidad de las Ciencias Informáticas. Con él se pretende brindar una herramienta multiplataforma que le permita al usuario con el menor esfuerzo cubrir las necesidades fundamentales de replicación de datos.

En este software (Reko) aún existen algunas deficiencias y precisamente una de ellas es la resolución de los conflictos que ocurren durante la replicación de los datos, lo cual es fundamental en el proceso. El desarrollo de un nuevo módulo dedicado a la resolución de los conflictos le permitiría al usuario visualizar los conflictos y tomar decisiones en su resolución, decisiones que se limitan a cambios manuales sobre la acción y la aplicación de reglas previamente creadas que permita tratar los conflictos de forma automática.

Por las razones anteriormente expuestas surge como **problema** científico el siguiente: ¿Cómo solucionar los conflictos en la replicación de datos en el software Reko?

Por tanto, el **objeto de estudio** de la investigación es: la replicación de datos usando Reko, definiendo como **campo de acción** del mismo: la resolución de conflictos en el software de replicación de datos Reko.

De esta manera la **idea a defender** con el presente trabajo, es la realización de un módulo para la resolución de conflictos en el software de replicación Reko, ajustado a las necesidades requeridas por dicha aplicación, estructura y estrategia de trabajo, posibilitando que éste mejore el proceso de replicación y control de objetos de una base de datos.

El **objetivo general** de este trabajo es desarrollar un módulo para solucionar los conflictos que ocurren en la replicación de datos usando Reko.

Para desarrollar el objetivo general se plantean los siguientes **objetivos específicos**:

- Analizar una solución para la resolución de conflictos en la replicación de datos del software Reko.
- Diseñar una solución para la resolución de conflictos en la replicación de datos del software Reko.

- Implementar una solución para la resolución de conflictos en la replicación de datos del software Reko.
- Validar la solución propuesta para la resolución de conflictos en la replicación de datos del software Reko.
- Integrar la solución al software Reko como un módulo del mismo.

Para darle solución al problema científico planteado y lograr el cumplimiento de los objetivos se proponen las siguientes **tareas**:

- Realización de un estudio del estado del arte teniendo en cuenta los trabajos anteriores relacionados con el tema.
- Definición de los requisitos funcionales y no funcionales de la aplicación.
- Elaboración del modelo conceptual.
- Desarrollo del diagrama de casos de uso del sistema.
- Desarrollo del diagrama de clases del análisis.
- Desarrollo del diagrama de clases del diseño.
- Implementación del sistema.
- Realización de pruebas al sistema desarrollado

Diseño Metodológico

Métodos Teóricos:

1- Método Analítico-Sintético:

En el trabajo se utilizará este método ya que se realizó un análisis sobre las diferentes teorías que existen del concepto de resolución de conflictos (en la replicación de bases de datos), permitiendo escoger los elementos principales que giran entorno al objeto que se está analizando.

Métodos Empíricos:

1- Entrevista:

Se realizarán entrevistas a los usuarios e implementadores del software de replicación Reko que son los principales conocedores del tema, para así estructurar todo el trabajo de acuerdo a sus exigencias. Este método se utilizará con el fin de recoger la información necesaria para llevar a cabo la constitución de un producto con verdadera calidad y que cumpla las expectativas del cliente.

Para lograr una mejor comprensión el presente documento se divide en introducción, cuatro capítulos de contenidos donde se incluye todo lo relacionado con el trabajo desarrollado, conclusiones, recomendaciones y bibliografía.

Capítulo 1 "*Fundamentación Teórica*" Aborda el estudio del arte del tema relacionado con el trabajo que se desarrolla. Se destacan además las tendencias y tecnologías actuales sobre las cuales se apoya la propuesta del sistema informático, así como las diferentes metodologías de desarrollo de software y herramientas a utilizar.

Capítulo 2 *"Descripción del sistema"* Se realiza una breve descripción de la solución propuesta, sus requisitos funcionales y no funcionales, así como los actores que intervienen en ella. Se representa el diagrama de Casos de Usos del Sistema, la descripción de los mismos y el modelo de dominio.

Capítulo 3 *"Análisis y Diseño"* Incluye la descripción y análisis de la solución que se propone para darle respuesta a la problemática planteada.

Capítulo 4 *"Implementación y Pruebas"* Contiene las estrategias de codificación, descripción del flujo de implementación y estilos a utilizar, además de las pruebas que le serán aplicadas al sistema para demostrar la robustez del mismo.

Capítulo 1: Fundamentación Teórica.

1.1 Introducción.

En este capítulo se abordan temas relacionados con la importancia de la réplica de información almacenada en distintas bases de datos ubicadas en distintos lugares y los conflictos que pueden surgir a la hora de replicar estas bases de datos. Se realizará un estudio del estado del arte analizando la forma de resolucionar conflictos que poseen otros replicadores. Se abordarán conceptos fundamentales relacionados con la replicación. Se realizará una breve descripción de las herramientas y metodologías de desarrollo a utilizar para alcanzar los objetivos trazados.

1.2 Estudio del Estado del Arte

1.2.1 Réplica de datos.

La replicación es el proceso de copiado y mantención de los objetos de una base de datos, tales como tablas, en múltiples bases de datos que componen un sistema de bases de datos distribuidas. Los cambios aplicados en un sitio, son capturados y almacenados localmente antes de ser transmitidos y aplicados en cada una de las ubicaciones remotas. Estas acciones se llevan a cabo desde una base de datos o nodo, como comúnmente se conoce, a otra, para luego sincronizar ambas y mantener la coherencia.

Por lo general, existen tres tipos de clasificaciones para las réplicas de datos atendiendo a:

- Ambiente de replicación:

  ✓ **Maestro-esclavo** (master-slave)**:** En este caso la replicación se realiza en un sólo sentido: desde un nodo maestro a uno o varios esclavos.

  ✓ **Multi-Maestro** (multi-master)**:** Se configuran varios nodos como maestros, estos pueden replicar entre sí.

- Forma de transmitir los cambios en el entorno de replicación:

  ✓ **Síncrona:** Los cambios ejecutados en un nodo maestro son aplicados instantáneamente en el nodo origen y los nodos destino dentro de una misma transacción. En caso de no poder

ejecutarse la acción en cualquiera de los nodos, la transacción completa es retrotraída en todos los nodos. Requiere una alta disponibilidad de recursos de red.

- ✓ **Asíncrona:** Los cambios ejecutados en una tabla son almacenados y enviados posteriormente al resto de los nodos del entorno cada ciertos intervalos de tiempo.

- Forma de capturar y almacenar los cambios a replicar:

  - ✓ **Basada en triggers (Disparadores):** Se crean una serie de triggers, en la base de datos, que permiten capturar las operaciones de inserción, actualización y eliminación realizadas sobre las tablas a replicar. Es la que emplean generalmente las herramientas de replicación de terceras partes.
  - ✓ **Basada en logs (Registros):** Se sostiene en la lectura de logs de cambios que proporcionan algunos gestores como Oracle.

Reko utiliza el ambiente de replicación Multi-maestro, la forma de transmitir los cambios en el entorno de replicación es asíncrona y la forma de capturar los cambios a replicar es basada en triggers(disparadores).

1.2.2 Conflictos en la replicación de las bases de datos.

Se deben diseñar las aplicaciones de manera tal que eviten los conflictos siempre que sea posible, ya que la detección y resolución de los conflictos aumenta la complejidad, el procesamiento y el tráfico de red. Reko utiliza la replicación de mezcla, la cual permite que varios sitios funcionen en línea o desconectados de manera autónoma, y mezclar más adelante las modificaciones de datos realizadas en un resultado único y uniforme. Los datos se sincronizan entre los servidores a una hora programada o a petición. Las actualizaciones se realizan de manera independiente, sin protocolo de confirmación, en más de un servidor, así el publicador o más de un suscriptor pueden haber actualizado los mismos datos. Por lo tanto, pueden producirse conflictos al mezclar las modificaciones de datos. Un conflicto se produce al insertar, actualizar o eliminar la misma fila en más de un lugar o cuando más de un nodo cambia los mismos datos o pudiera ser que al replicar cambios, la réplica encuentre determinados tipos de errores. Existen varios tipos de conflictos a tener en cuenta, al tratar la replicación de datos. Estos pueden ocurrir cuando estamos trabajando en un ambiente de replicación que permite actualizaciones concurrentes sobre los mismos datos en múltiples sitios [1]. A continuación, algunos de los conflictos identificados de forma general y algunas de las posibles resoluciones:

Conflicto de Actualización: Un conflicto de actualización ocurre cuando se produce la replicación de una actualización (update) sobre un registro con otra actualización (update) sobre el mismo registro. Este conflicto ocurre cuando dos transacciones originadas desde distintos sitios actualizan el mismo registro, en forma cercana en el tiempo.

Resolución

- Prioridad: Cada servidor obtiene una prioridad única, y el servidor de mayor prioridad "gana", respecto de aquellos con prioridad menor.
- Timestamp: La más nueva o la más antigua de las modificaciones es la considerada correcta, y por defecto, sino se eligió ninguno de los criterios "gana" la más nueva.
- Particionamiento de datos: Se garantiza que cada registro sea manipulado por un único servidor, lo que simplifica la arquitectura.

Conflicto de unicidad: El conflicto de unicidad sucede cuando la replicación de un registro intenta violar una restricción de integridad, ya sea por llave primaria o única (Primary Key o Unique). Por ejemplo considere lo que sucede, cuando dos transacciones originadas de dos sitios diferentes, cada una inserta un registro, en su respectiva tabla replicada, con el mismo valor de clave primaria. En ese caso sucede un conflicto de unicidad.

Resolución

- Para cada servidor brindar un rango distinto de números para los generadores de clave (secuencias).
- Agregar el identificador del servidor a la clave primaria.
- Replicar en tablas separadas, y acceder a los datos a través de una vista formada por la unión de ellas. Para resolver el conflicto de potenciales claves duplicadas en la unión se usará una pseudo columna que representa la base de datos fuente.

Conflicto de supresión: Un conflicto de supresión ocurre cuando dos transacciones originadas de sitios diferentes, una de ellas intenta borrar un registro, y la otra actualizar o borrar el mismo registro, ya que en este caso el registro no existe, tanto para ser actualizado como borrado.

Resolución

Para evitar este tipo de conflictos, una posible solución es que los sitios marquen lógicamente los registros a ser borrados y que periódicamente el sitio maestro corra un proceso que realice el borrado ("delete") físico de los datos, es decir desde los sitios replicados no se puede ejecutar una sentencia para hacer el borrado de los datos (delete).

Conflicto de orden: Los conflictos de orden pueden ocurrir en ambientes de replicación con tres o más sitios maestros. Si la propagación al sitio maestro X, está bloqueada por alguna razón, entonces la replicación de modificaciones en datos puede seguir siendo propagada a través de los otros sitios maestros; al finalizar la propagación estas modificaciones debieron ser propagadas al sitio X en un orden diferente a como ocurrieron en los otros sitios maestros, pudiendo producirse un conflicto.

Resolución

Este tipo de conflictos suelen resolverse asignándole distintas prioridades a los sitios maestros, de forma de ordenar las transacciones de acuerdo a esta.

En el mundo de la informática existen varios replicadores que tienen implementado la resolución de conflictos desde cierto punto. Algunos de los encontrados son:

Daffodil Replicator: Es una poderosa herramienta de Java de código abierto para la integración, migración y protección de datos en tiempo real. Permite bidireccional datos de replicación y sincronización entre bases de datos homogéneas y heterogéneas, incluyendo Oracle, MySQL, Daffodil DB, PostgreSql, entre otras [2].
Incluye además:

- La capacidad de detectar y permitir resolución de conflictos.
- Independencia de la plataforma.
- Facilidades para replicar entre bases de datos con estructuras iguales.
- Soporte para la replicación de datos de gran tamaño.

La principal limitante detectada en el estudio de este replicador es que no es capaz de replicar en un escenario multi-maestro y la necesidad de la universidad era utilizar un replicador que pudiera replicar en dicho escenario.

DbReplicator: Básicamente en este replicador la resolución de conflictos esta implementada como publisher wins/subscriber wins, esto quiere decir se actualiza con la última información ya sea por parte del servidor o del cliente, el nodo ganador se define antes de llevar a cabo la replicación. En futuras versiones se le agregarán reglas más complejas basadas en algoritmos de resolución de conflictos [3].

Pyreplica: No posee Resolución de conflictos ni tolerancia a fallos (Fail over) automático pero advierte al detectar conflictos de actualización/eliminación y falla en conflictos de inserción o errores de integridad de datos mediante señales NOTIFY [4].

Reko. Características.

El software de réplica de datos, Reko, fue diseñado y construido por un grupo de desarrolladores procedentes de la Universidad de las Ciencias Informáticas. El 2 de noviembre de 2009, fue liberada su versión 1.0, luego de ser aprobado por los Laboratorios de Pruebas y el grupo de Calisoft. Soporta la replicación de transacciones a través de *Hibernate* y la replicación de acciones a través de *triggers* [5].

Actualmente, Reko cuenta con un conjunto de componentes que permiten la replicación de datos entre bases de datos de iguales estructuras y distintas locaciones.

Núcleo: Maneja las configuraciones fundamentales del software y agrupa las principales funcionalidades de procesamiento de información.

Capturador de Cambios: Captura los cambios que se realizan sobre la Base de Datos y se los entrega al Distribuidor de Cambios.

Aplicador de Cambios: Ejecuta sobre la Base de Datos los cambios que sean replicados hacia la misma.

Distribuidor de Cambios: Determina para dónde debe ser enviado cada cambio realizado en la Base de Datos, los envía y es responsable de que cada cambio llegue a su destino.

BD Local de Réplica: Es utilizada Berkeley DB para guardar las configuraciones propias de la réplica, las acciones sobre la Base de Datos que han dado conflicto al aplicarse y las acciones o transacciones que no han podido llegar a su destino.

Consola Web de Administración: Representa la interfaz del software. Permite realizar las configuraciones principales del software como el registro de nodos, configuración de las tablas a replicar, datos a replicar y el monitoreo del funcionamiento del software.

Servidor JMS (Java Messenger Service): Servidor de Mensajería bajo la especificación Java Message Service, es utilizado como punto intermedio en la distribución de la información enviada bajo JMS.

Base de datos: Es la base de datos que se está replicando, el software de réplica envía los cambios que se realizan sobre ella y aplica los cambios que provienen de otros nodos de réplica.

Este módulo, como se dijo anteriormente, está encargado de solucionar los conflictos que surgen en la réplica de datos. Estos son tratados de dos formas, manual y automática. El tratamiento manual consiste en aplicar cambios directamente sobre el SQL con conflicto mediante un componente implementado, el cual se encarga de dividir el SQL por partes, para que el usuario pueda modificar sin problemas cualquiera de ellas y luego mediante otro componente se une la sentencia y queda lista para enviarse. El tratamiento manual sirve como base para el tratamiento automático, ya que la misma solución que se le da al conflicto de forma manual debe ser implementada para la forma automática y así se le ahorra trabajo al usuario, porque sería demasiado engorroso solucionar conflictos del mismo tipo varias veces.

Luego de investigar cómo se realiza la resolución de conflictos en el mundo de las réplicas de manera automática se llega a la conclusión de que se debe utilizar un motor de reglas, y después de analizar algunos motores el que se propone utilizar es JBoos Drool (JDrools) ya que en el centro de bases de datos se ha trabajado antes con él y hay mas bibliografía y experiencia con el uso de este motor de reglas y así sería más fácil entenderlo. ¿Y qué es un motor de reglas de negocio? Es un componente que, a partir de una información inicial y un conjunto de reglas, detecta qué reglas deben aplicarse en un instante determinado y cuáles son los resultados de esas reglas. Básicamente un motor de reglas de negocio está compuesto de tres elementos: un conjunto de reglas, el espacio de trabajo (o el conocimiento que tiene),

y el procesador de reglas. Las reglas son sentencias de la forma IF-THEN, de tal manera que si se cumplen todas las condiciones del IF se ejecutan todas las acciones del THEN. El espacio de trabajo es donde se guarda el conocimiento (objetos Java) que el motor utilizará para decidir que reglas deben activarse. Aunque el motor de regla escogido es JBoos Drool existen otros como Jess, el cual es un sistema basado en reglas de producción y un entorno de scripting (archivo de órdenes o archivo de procesamiento por lotes) implementado enteramente en Java por Ernest Friedman-Colina en Sandia National Laboratories en Livermore, California [6].

1.3 Herramientas y Metodologías

1.3.1 Metodología de Desarrollo de Software. Open Up.

En la actualidad existe un gran número de herramientas y metodologías capaces de brindar todo tipo de soluciones. Esta situación es la que conduce la problemática a la cual se enfrentan los desarrolladores de hoy, ¿cuál de las herramientas y metodologías emplear?, ya que las mismas determinan la calidad que tendrá el producto, pues a la vez definen qué papel debe ejercer cada miembro dentro del equipo de desarrollo, las actividades que debe cumplir cada uno de ellos, los artefactos que serán generados de cada tarea, así como el registro de cada detalle de la información que se irá generando.

Las metodologías y técnicas existentes para el desarrollo de productos de software, son analizadas y definidas en la arquitectura de cada proyecto. En el caso de Reko, de las existentes, la metodología definida fue Open Up.

Open Up/Basic es un framework de procesos de desarrollo de software de código abierto, construido sobre una donación realizada por IBM del Basic Unified Process y liberado por el Eclipse Process Framework (EPF). Permite abordar ágilmente el desarrollo de proyectos pequeños y tiene un enfoque centrado en el cliente con iteraciones cortas.

Este proceso de desarrollo unificado está basado en las mejores prácticas del Rational Unified Process (RUP), permitiendo de esta manera que se puedan desarrollar artefactos ligeros apropiados, utilizando el lenguaje de modelado unificado (UML por sus siglas en ingles) e incrementando de esta forma las probabilidades de éxito en función de calidad, costo, tiempo y alcance.

Debido a que está basada en RUP, aplica un enfoque iterativo e incremental dentro de su estructura del ciclo de vida y puede adaptarse para desarrollar diversos tipos de proyectos. Además es un proceso iterativo del desarrollo del software que es mínimo, completo, y extensible [7].

El proceso es mínimo pues solamente el contenido fundamental es incluido; es completo porque puede ser manifestado como todo el proceso para construir un sistema y es extensible pues puede ser utilizado como fundamento sobre el cual el contenido de proceso se pueda agregar o adaptar según lo necesitado.

Open Up. Roles y Artefactos.

Siguiendo el ciclo de vida que propone Open Up, para un proyecto de desarrollo de software, que incluye las fases de Inicio, Elaboración, Construcción y Transición; se desempeñarán los siguientes roles, que obtendrán sus respectivos artefactos:

Analista: Durante la fase de Inicio, en el Modelamiento del Negocio, su misión será definir la visión del sistema; describirá el problema que se presenta y las características del sistema según los requerimientos de los interesados. Estas actividades propiciarán que obtenga los artefactos correspondientes al Glosario de Términos y el Documento Visión.

En el proceso de Administración de Requerimientos, será el encargado de identificarlos y luego refinarlos, así como comunicarlos al equipo de desarrollo; estas actividades se realizan a partir de los Casos de Uso del Negocio, y producen como salida los Casos de Uso del Sistema, Modelo de Casos de Uso y se actualiza el Glosario de Términos.

Arquitecto: Durante la Fase de Inicio, desarrollará la visión de la arquitectura a través del análisis de los requerimientos más significativos. Esta actividad será realizada partiendo del uso del Glosario de Términos, del Documento Visión y del Modelo de Casos de Uso.

La salida obtenida serán las Notas Arquitectónicas.

Durante la Fase de Elaboración, refinará la arquitectura; lo cual permitirá la obtención de una arquitectura robusta y la primera aproximación al diseño.

Probador: Durante la Fase de Inicio, en la Administración de Requerimientos, será el encargado de crear y desarrollar los Casos de Prueba, así como identificar los Datos de Prueba con los que se validarán los requerimientos que serán probados. Las salidas resultantes de estas actividades serán los Casos de Prueba.

Durante la Fase de Elaboración, implementará uno o varios artefactos de prueba para permitir la validación del sistema. Esto lo hará partiendo de los Casos de Prueba y generando el Script de Pruebas. Luego ejecutará las pruebas para determinar la calidad del producto, partiendo del Build y del Script de Pruebas y obteniendo el Log de Prueba.

Desarrollador: Durante la Fase de Elaboración será el encargado de diseñar la solución. Partiendo de la Arquitectura, los Requerimientos de Soporte y los Casos de Uso, obtendrá un diseño que dará paso para luego implementar las Pruebas de Desarrollador, siendo este el artefacto resultante de esta actividad.

Luego implementará la solución obteniendo como salidas el Build, la Implementación, los Requerimientos De Soporte y los Casos de Uso. Posteriormente ejecutará las pruebas de desarrollador para verificar los resultados y obtendrá los Resultados de Pruebas de Desarrollador.

1.3.2 Lenguaje de programación. Java.

Java es un lenguaje de programación orientado a objeto (OO) desarrollado por Sun Microsystems, a principio de los años 90´s, e independiente de la plataforma. Es un lenguaje de propósito general, lo cual

permite crear diversos tipos de aplicaciones con él; radicando su mayor éxito en Internet, con el uso de los Applets, las aplicaciones cliente – servidor o las Java Server Pages [8].

Se distingue por ser capaz de gestionar la memoria automáticamente, no permite el uso de técnicas de programación inadecuadas; posee mecanismos de seguridad incorporados, los cuales limitan el acceso a recursos de las máquinas donde se ejecuta; incorpora herramientas de documentación; es multihilos (multithreading). Todas estas características lo califican como un lenguaje de programación robusto y nos lleva a adoptarlo como lenguaje de programación.

1.3.3 Herramientas CASE. Visual Paradigm.

Herramientas CASE o Computer Aided Software Engineering, lo que se traduce como Ingeniería de Software Asistida por Computación, representan una forma que permite modelar los procesos de negocios de las empresas y desarrollar los sistemas de información gerenciales. Sin importar su arquitectura, tales herramientas deben abarcar propiedades como las siguientes:

- Una interfaz gráfica y textual, que le permita al usuario manejar los objetos de diseño.
- Contar con un diccionario de datos, a fin de rastrear y controlar los objetos diseñados.
- Disponer de un conjunto de herramientas que permitan chequear las reglas del diseño y analizar su lógica.

La herramienta escogida tras una amplia labor investigativa fue Visual Paradigm, ya que es una de las herramientas CASE más completas que existe en el mercado actualmente.

Visual Paradigm

Visual Paradigm es una herramienta UML profesional que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientado a objeto (OO), construcción, pruebas y despliegue. Permite la elaboración de todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación de diferentes extensiones (.Pdf, .Doc, etc.).

Facilita el modelado colaborativo con Sistema de Versiones Concurrentes (Concurrent Versions System - CVS) y Subversión (control de versiones), y la realización del proceso de ingeniería así como ingeniería inversa (proceso ingenieril en el que se obtienen modelos conceptuales a partir de los artefactos software como código fuente, ejecutables, binarios y ficheros intermedios). Brinda soporte para el mapeo de objeto relacional (Object-Relational Mapping – ORM) y facilidades para la generación de bases de datos. Es una herramienta colaborativa, pues soporta múltiples usuarios trabajando sobre el mismo proyecto.

Presenta licencia gratuita y comercial. Es fácil de instalar y actualizar y compatible entre ediciones. Debido a todo lo antes mencionado, se puede concluir que Visual Paradigm se escogió por su robustez, usabilidad y portabilidad.

1.3.4 Ambiente de desarrollo. Eclipse IDE.

Desarrollado inicialmente por IBM como el sucesor de su familia de herramientas para VisualAge. Siendo ahora desarrollado por la Fundación Eclipse, una organización independiente, sin ánimo de lucro que fomenta una comunidad de código abierto y un conjunto de productos complementarios, capacidades y servicios [9].

Eclipse es una plataforma universal o Entorno Integrado de Desarrollo (Integrated Development Environment – IDE) de código abierto, con una arquitectura abierta y basada en plug-ins. La arquitectura de plug-ins permite integrar diversos lenguajes sobre un mismo IDE e introducir otras aplicaciones accesorias.

Eclipse es soportado por los principales sistemas operativos:

- Linux
- Windows
- Solaris 8 (SPARC/GTK 2)
- Mac OSX –Mac/Carbon

Dispone de un Editor de Texto con resaltado de sintaxis. La compilación es en tiempo real. Contiene pruebas unitarias con JUnit, control de versiones con CVS, integración con Ant, asistentes (wizards) para creación de proyectos, clases, pruebas, etc., y refactorización.
Características principales de Eclipse como IDE de Java:
• Editor visual con sintaxis coloreada.
• Compilación incremental de código.
• Modifica e inspecciona valores de variables.
• Avisa de los errores cometidos mediante una ventana secundaria.
• Depura código que resida en una máquina remota.
Por estas razones este fue el entorno de desarrollo seleccionado.

1.3.5 Frameworks.

Un framework es una estructura de software compuesta de componentes personalizables e intercambiables para el desarrollo de una aplicación. En otras palabras, un framework se puede considerar como una aplicación genérica incompleta y configurable a la que podemos añadirle las últimas piezas para construir una aplicación concreta.

1.3.5.1 Spring.

Spring es un framework Open Source, desarrollado por la compañía Interface 2, para aplicaciones escritas en el lenguaje Java. Los desarrolladores de este framework, lograron combinar en él herramientas tales como: J2EE (Java 2 Enterprise Editions), incluyendo EJB (Enterprise JavaBeans), Servlets y JSP (Java Server Pages); esto facilitó brindar una estructura mucho más sólida y brindar un mejor soporte [14].

Dicho framework es considerado liviano, ligero, pues no necesita de muchos recursos para ser ejecutado. Está basado en inyección de dependencias (DI) y orientación a aspectos. Brinda abundante funcionalidad de infraestructura (integración con frameworks de persistencia, gestión de transacciones, etc.) y deja el desarrollo de la lógica de la aplicación al desarrollador. Contiene y gestiona el ciclo de vida y configuración de objetos de aplicación; por estos motivos es considerado un contenedor.

Su arquitectura está compuesta por siete capas o módulos, lo cual permite tomar y ocupar únicamente las partes que son necesarias para el proyecto. Es por esto que se decidió adoptar esta herramienta como framework de desarrollo.

1.3.5.2 Dojo.

Dojo es un Framework en JavaScript el cual te permite añadir características dinámicas fácilmente a las páginas Web. Puedes utilizar los componentes que incorpora Dojo para mejorar la usabilidad y funcionabilidad. Se puede construir interfaces con degradados de color, reproductores y transacciones animadas de forma rápida y sencilla. Contiene varias características que trabajan a través de la mayoría de los navegadores, tales como: menúes, pestañas, información sobre herramientas y tablas ordenables.

1.3.6 Lenguaje de Modelado. UML.

Utilizar un lenguaje de modelado nos permite verificar efectivamente el funcionamiento adecuado que el modelo debe poseer, pues existen menos medios de verificación, lo cual hace al modelo más sencillo que la propia programación.

El Unified Modeling Language (UML) es un lenguaje estándar que permite especificar, visualizar, construir y documentar todos los elementos que forman un sistema de software orientado a objetos. Por lo que su propósito consiste en elaborar los artefactos de un sistema a través de las distintas etapas de su ciclo de vida, principalmente durante el análisis y el diseño del mismo.

Modelar con UML trae consigo una serie de beneficios a los programadores, pues mediante él es posible establecer un conjunto de requerimientos y estructuras necesarias para plasmar un sistema de software previo a la escritura del código. Aunque UML es un lenguaje, éste posee más características visuales que programáticas, facilitando así el entendimiento común entre los miembros de un equipo de trabajo, dígase, analistas, diseñadores, programadores, etc.

El tiempo invertido en el desarrollo de la arquitectura se minimiza, la trazabilidad y documentación del proyecto se realiza de una forma ordenada y guiada por los casos de uso, pero lo más importante es la notable efectividad y productividad que se consigue en labores de diseño arquitectónico y mantenimiento, haciendo uso de UML frente a la realización de las mismas tareas, en ausencia de modelos.

1.4 Conclusiones.

En este capítulo se expusieron las diferentes herramientas y metodología que se usarán en el desarrollo de la aplicación, para lograr un resultado satisfactorio. Se caracterizó cada una de ellas para su mejor entendimiento y en breves palabras se explica el por qué de su adopción. Además se analizaron diferentes aplicaciones utilizadas a nivel mundial y algunas de sus características, así como se describieron algunos de los tipos de conflictos que surgen en la réplica de datos y su posible resolución.

Capítulo 2: Descripción del sistema.

2.1 Introducción.

En este capítulo ofreceremos la descripción del sistema, sus requisitos funcionales, no funcionales, sus actores, sus diagramas, casos de uso, así como la descripción de cada caso de uso crítico. Además del modelo de dominio propuesto.

2.2 Descripción Detallada del Proceso a Automatizar.

El módulo para la resolución de conflictos en Reko se encarga de solucionar los conflictos que surgen a la hora de la réplica de datos. Esta aplicación es la encargada de solucionar conflictos dándole la oportunidad al usuario de dar tratamiento manual a los errores o de utilizar un motor de reglas previamente implementado con el objetivo de ahorrarle trabajo al usuario.

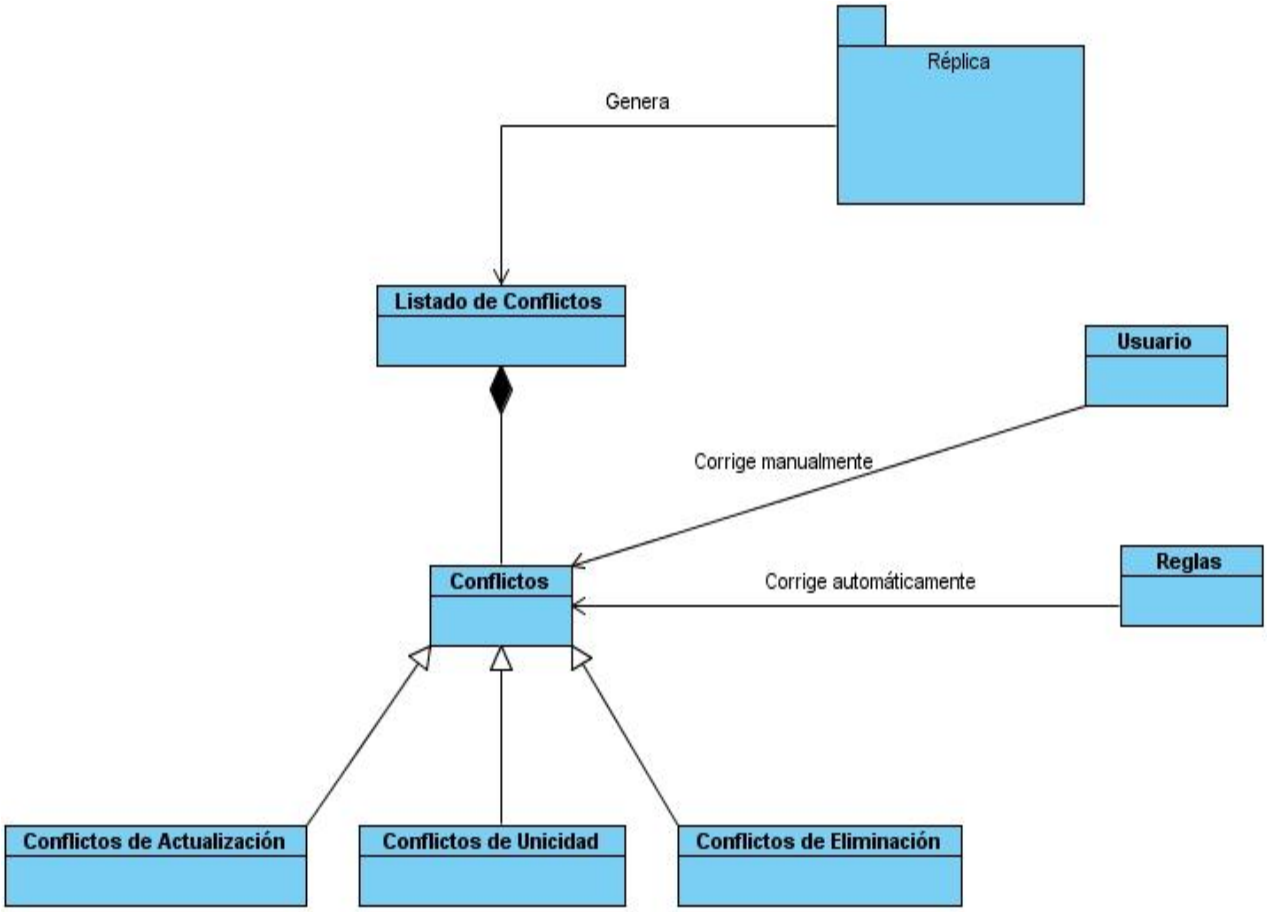
2.3 Modelo de Dominio.

El modelo del dominio es una representación de los conceptos u objetos del mundo real, significativos para un problema. Tiene como objetivo fundamental la descripción de las clases más importantes en el sistema y representa conceptos del mundo real, no de los componentes de software.

2.3.1 Descripción de las Clases del Modelo de Dominio.

- Réplica: contiene la información acerca del proceso de réplica que sucede antes de surgir los conflictos.
- Listado de Conflictos: contiene un listado de los conflictos que surgen luego de realizar la réplica.
- Conflictos: contiene la información acerca de todos los conflictos en general que pueden suceder en la réplica de datos.
- Conflictos de Actualización: contiene información acerca de este tipo de conflicto en particular.
- Conflictos de Unicidad: contiene información acerca de este tipo de conflicto en particular.
- Conflictos de eliminación: contiene información acerca de este tipo de conflicto en particular.
- Usuario: guarda información acerca de un usuario encargado de la resolución manual de los conflictos.
- Reglas: contiene reglas encargadas de dar solución automática a los conflictos detectados.

2.3.2 Diagrama de Clases del Dominio.



2.4 Modelado del Sistema.

Los requisitos son condiciones o capacidades que necesita el usuario para resolver un problema o conseguir un objetivo determinado. Esta definición se extiende y se aplica a las condiciones que debe cumplir o poseer un sistema -o uno de sus componentes- , para satisfacer un a un usuario determinado.

2.4.1 Requisitos Funcionales.

Los requisitos funcionales de un sistema describen su funcionalidad, los servicios que de él se esperan, o los que proveerá [10]. Para este modulo se identificaron los siguientes requisitos funcionales:

RF 1 Capturar los Conflictos.

RF 2 Listar los Conflictos.

RF 3 Resolver Conflictos

RF 3.1 Resolver automáticamente.
RF 3.2 Resolver manualmente.

2.4.2 Requisitos no Funcionales.

Los requerimientos no funcionales son propiedades o cualidades que el producto debe tener [10]. Para este modulo se tienen los siguientes:

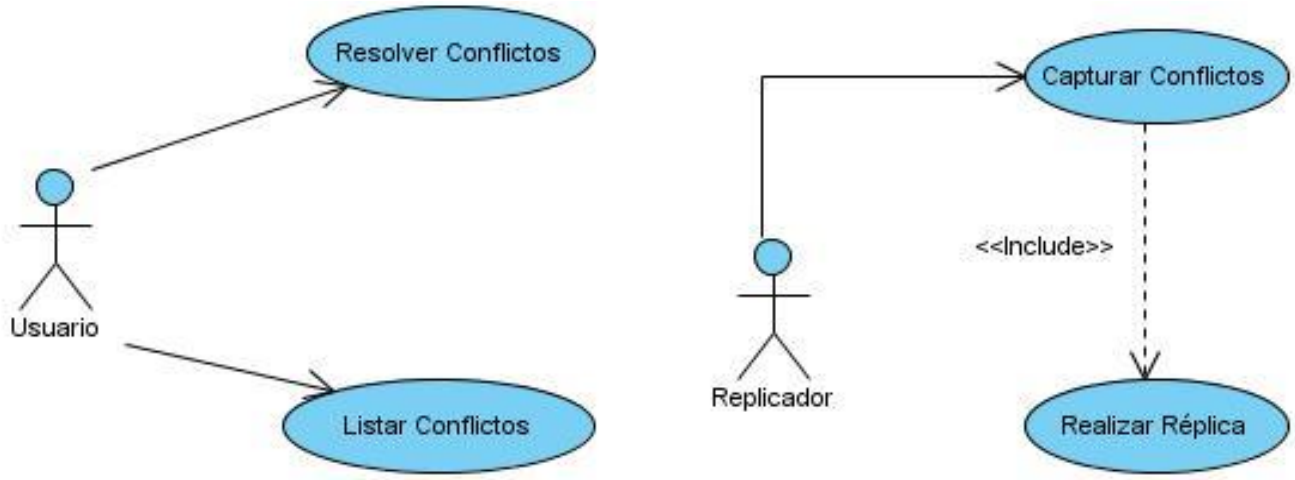
1. Apariencia o interfaz externa: La captura y tratamiento de conflictos en la réplica se realiza a través de una interfaz Web, por lo que es administrable vía remota usando un navegador. Además de ser una interfaz amigable, agradable a la vista del usuario con colores refrescantes.

2. Usabilidad: El módulo debe garantizar un acceso fácil y rápido, contando con un interfaz que satisfaga las necesidades de usuarios.

3. Rendimiento: Los tiempos de respuestas deben ser generalmente rápidos al igual que la velocidad de procesamiento de la información para así mantener a gusta el cliente.

4. Disponibilidad: Se debe garantizar el funcionamiento de la aplicación durante las 24 horas del día y los siete días de la semana, con el menor tiempo posible de recuperación de fallos. Se deben crear copias de respaldo periódicas que puedan restaurar el sistema en caso de fallo crítico o pérdida total de la información.

2.4.3 Diagrama de Casos de Uso del Sistema.



2.4.4 Descripción de los Diagramas de Casos de Uso del Sistema.

Luego de un profundo análisis se llego a la conclusión de que serian 3 los Casos de Uso derivados de los requisitos funcionales de la aplicación a desarrollar:

1. Capturar Conflicto.
2. Listar Conflictos.
3. Resolver Conflictos.

- Caso de Uso Capturar Conflicto.

Caso de Uso:	Capturar Conflicto
Actores:	Replicador
Resumen:	El caso de uso se inicia cuando surge un conflicto en la réplica de datos**.** El sistema lo captura.
Precondiciones:	
Referencias	RF1

Prioridad	Crítico
Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
	1. Surge un conflicto en la réplica de datos.
2. El replicador detecta el conflicto que surge, lo captura.	
3. Guarda el conflicto para darle tratamiento luego.	

- Caso de Uso Listar Conflictos.

Caso de Uso:	Listar Conflictos
Actores:	Usuario
Resumen:	El caso de uso se inicia cuando el usuario indica la opción de conflictos en la Réplica. El sistema muestra un listado con todos los conflictos capturados.
Precondiciones:	
Postcondiciones:	Los conflictos quedan listados y listos para ser solucionados.
Referencias	RF2
Prioridad	Crítico
Flujo Normal de Eventos	

Acción del Actor	Respuesta del Sistema
1. Indica la opción de Conflictos en Réplica	2. Muestra un listado con todos los conflictos capturados

- Caso de Uso Resolver Conflictos.

Caso de Uso:	Resolver Conflictos
Actores:	Usuario
Resumen:	El caso de uso se inicia cuando se captura un conflicto en la réplica de datos y se dispone a darle tratamiento. Se podrán realizar alguna de las siguientes operaciones: • Resolver automáticamente (con el motor de reglas del sistema). • Resolver manualmente (por el usuario).
Precondiciones:	El sistema inicialmente debe haber capturado y listado los conflictos generados en la réplica para luego darle tratamiento.
Postcondiciones:	El conflicto queda solucionado y se inserta en la base de datos nuevamente.
Referencias	RF4
Prioridad	Crítico

Flujo Normal de Eventos	
Acción del Actor	Respuesta del Sistema

1. El usuario en dependencia del tipo de error capturado en la replicación de los datos, selecciona el vínculo Manual o Automática • Si decide resolver automáticamente: ir a la sección "Resolución Automática" • Si decide resolver manualmente: ir a la sección "Resolución Manual".	
Sección "Resolver automáticamente"	
Acción del Actor	**Respuesta del Sistema**
1. El Usuario presiona el vínculo Automáticamente.	2. El sistema ejecuta el conjunto de reglas previamente definidas por el usuario hasta que coincida una.
	3. El sistema solucionará el conflicto finalizando así el Caso de Uso.
Flujos Alternos	
Acción del Actor	**Respuesta del Sistema**
	3.1. El sistema no encuentra ninguna regla que coincida con el tipo de conflicto y da la opción de ir a la sección "Solucionar manualmente" y finaliza el Caso de Uso.
Sección "Resolver manualmente"	
Acción del Actor	**Respuesta del Sistema**
1. El Usuario presiona el vínculo Manual.	2. El sistema muestra una interfaz donde mostrará un editor de acciones con el conflicto capturado.
3. El usuario edita los datos que generan conflicto y acepta para comprobar la réplica.	4. El sistema incluye los datos de la modificación dentro de los datos a replicar.
	5. El sistema le brinda la información al usuario, finalizando así el Caso de Uso

2.5 Conclusiones.

En este capítulo se definieron las características principales del sistema. Para dar cumplimiento a las funcionalidades y características del sistema, se detectaron los requisitos funcionales para de ahí obtener los casos de uso, y los requisitos no funcionales. Se identificaron los actores del sistema así como 3 casos de uso del sistema con sus descripciones. Todo lo anterior, provee de una visión general de lo que el sistema debe hacer.

Capítulo 3: Diseño del sistema.

3.1 Introducción

En este capítulo se describe la arquitectura del sistema, haciendo énfasis en el estilo arquitectónico y en los patrones de diseño utilizados. Además, se desarrolla el diseño del sistema dentro del cual entran todos los diagramas de clases del diseño además de los diagramas de secuencia para cada caso de uso.

3.2 Estilo de arquitectura.

Un estilo arquitectónico define una familia de sistemas de software en términos de su organización estructural. Un estilo arquitectónico representa los componentes y las relaciones entre ellos con las restricciones de su aplicación y las asociaciones y reglas de diseño para su construcción. Shaw y Garlan (Shaw y Garlan, 1996) precisan además, que un estilo arquitectónico define un vocabulario de componentes y tipos de conectores. Resultó seleccionado para el desarrollo del presente trabajo el Modelo Vista Controlador (MVC) como estilo y patrón, el cual entra en la familia de estilos de llamada y retorno, ya que abarca ambas definiciones independientemente del nivel de abstracción donde se aplique [11].

3.2.1 Patrón de arquitectura

Reconocido como estilo arquitectónico por Taylor y Medvidovic, aunque en ocasiones se le define más bien como un patrón de diseño. El patrón conocido como Modelo-Vista-Controlador (MVC) separa el modelado del dominio, la presentación y las acciones basadas en datos ingresados por el usuario en tres clases diferentes:

- ✓ Modelo: El modelo administra el comportamiento y los datos del dominio de aplicación, responde a requerimientos de información sobre su estado (usualmente formulados desde la vista) y responde a instrucciones de cambiar el estado (habitualmente desde el controlador).
- ✓ Vista: Maneja la visualización de la información.
- ✓ Controlador: Interpreta las acciones del ratón y el teclado, informando al modelo y/o a la vista para que cambien según resulte apropiado.

Tanto la vista como el controlador dependen del modelo, el cual no depende de las otras clases.

Ventajas:

- ✓ Soporte de vistas múltiples. Dado que la vista se halla separada del modelo y no hay dependencia directa del modelo con respecto a la vista, la interfaz de usuario puede mostrar múltiples vistas de los mismos datos simultáneamente.
- ✓ Menor acoplamiento, porque desacopla las vistas de los modelos y desacopla los modelos de la forma en que se muestran e ingresan los datos.
- ✓ Mayor cohesión debido a que cada elemento del patrón está altamente especializado en su tarea (la vista en mostrar datos al usuario, el controlador en las entradas y el modelo en su objetivo de negocio).

Desventajas:

- ✓ Costo de actualizaciones frecuentes. Desacoplar el modelo de la vista no significa que los desarrolladores del modelo puedan ignorar la naturaleza de las vistas.

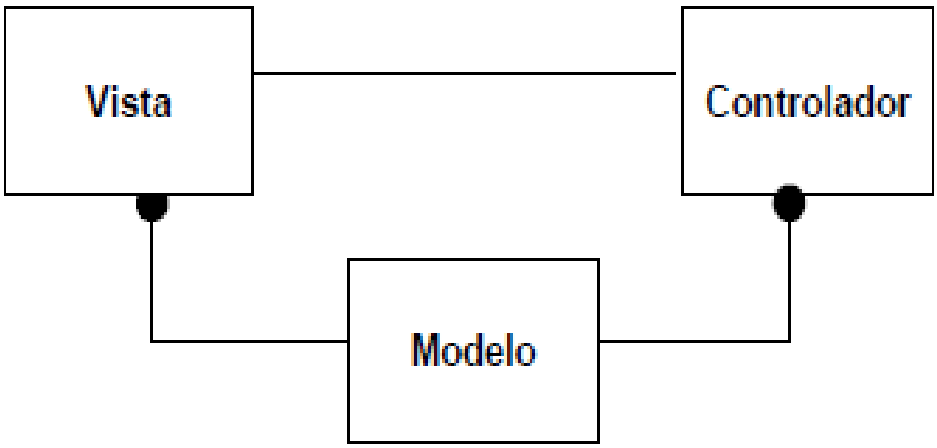


Figura 3.1. Representación del patrón MVC

Framework MVC de Spring: Es uno de los módulos del framework Spring que implementa una arquitectura Modelo-Vista-Controlador, el cual es utilizado como base para desarrollar la aplicación web del presente trabajo. Este módulo, define un conjunto de interfaces, que deben ser implementadas en la aplicación web, para dar respuesta a un evento determinado. En la figura a continuación se detalla cómo está implementado el patrón Modelo-Vista-Controlador sobre el framework Spring.

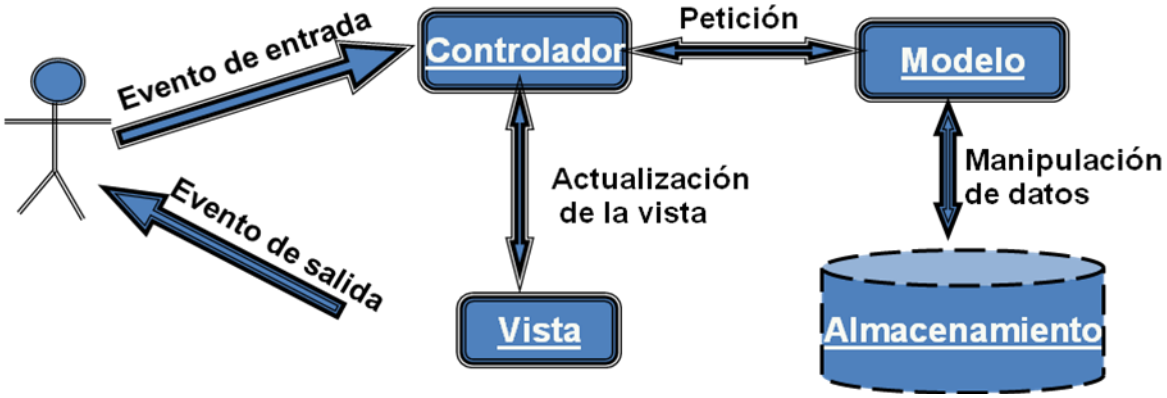


Figura 3.2 Representación del patrón MVC sobre el framework Spring

3.3 Patrones de Diseño

Un patrón de diseño, es una descripción de clases y objetos, adaptada para resolver un problema de diseño general en un contexto particular. Además, permite identificar clases, roles, colaboraciones, instancias y la distribución de responsabilidades [12].

Patrón de diseño DAO: Plantea, utilizar un *Data Access Object* (DAO) para abstraer y encapsular todos los accesos a la fuente de datos, siendo el encargado de implementar el mecanismo de acceso requerido para trabajar con la misma. Como la interfaz expuesta por el DAO no depende de la fuente de datos subyacente, este patrón le permite adaptarse a diferentes esquemas de almacenamiento, sin que se vean afectados los componentes del negocio. Esencialmente, el DAO actúa como un adaptador entre el componente y la fuente de datos.

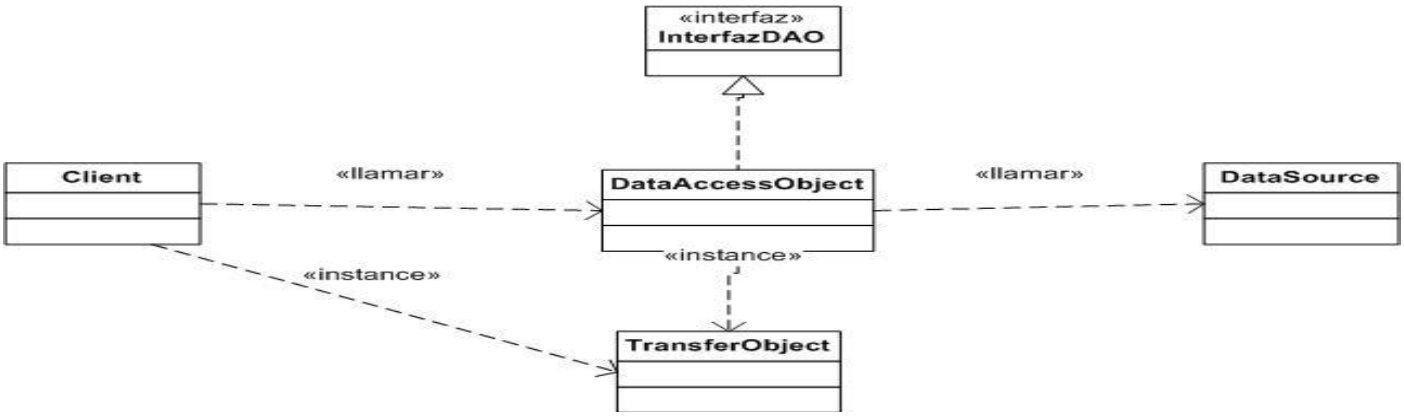


Figura 3.3 Estructura del DAO.

Patrón Creador: consiste en asignarle a una clase B la responsabilidad de crear una instancia de clase A en uno de los siguientes casos:

- B agrega los objetos A.
- B contiene los objetos A.
- B registra las instancias de los objetos A.
- B utiliza específicamente los objetos A.
- B tiene los datos de inicialización que serán transmitidos a A cuando este objeto sea creado (así que B es un Experto respecto a la creación de A). B es un creador de los objetos A.

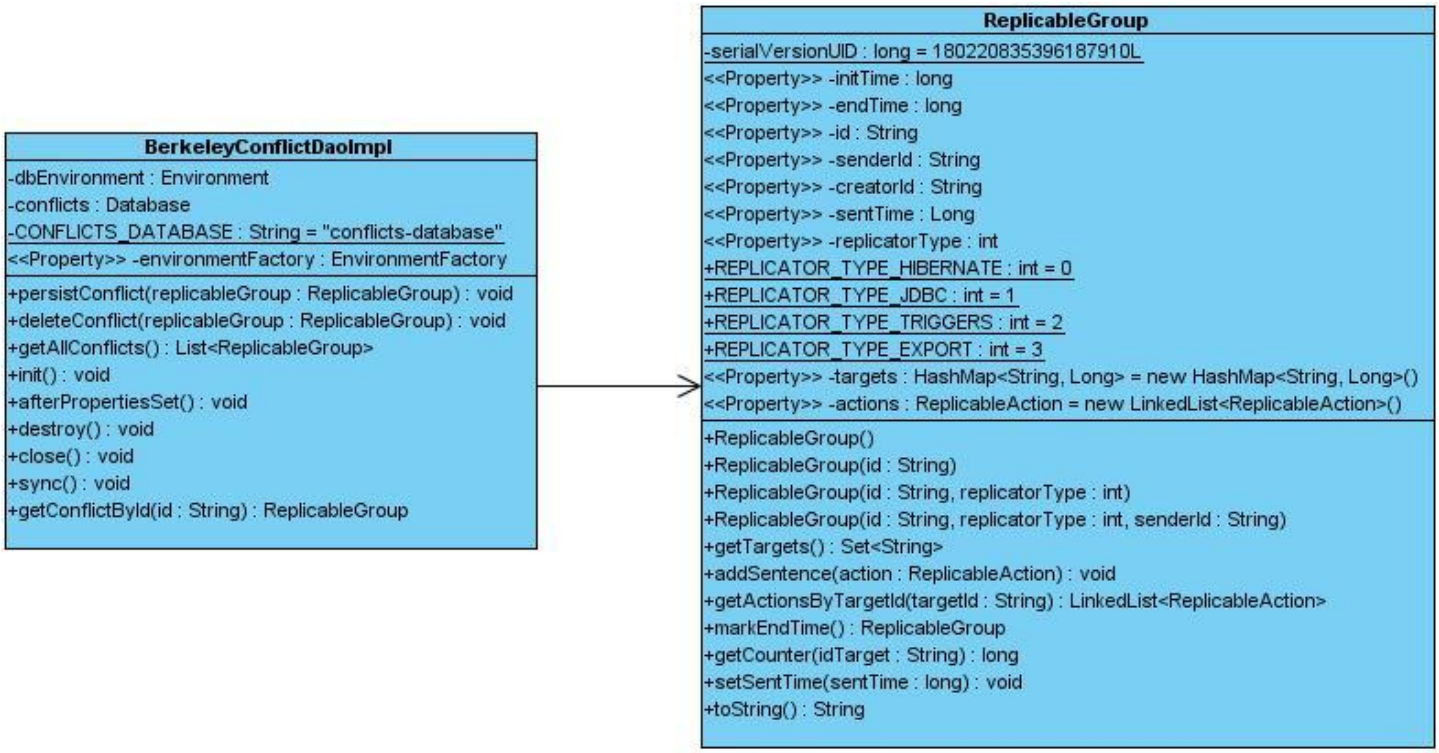


Figura 3.4 Patrón Creador

Patrón Controlador: Se encarga de asignar la responsabilidad de controlar el flujo de eventos del sistema a clases específicas, facilitando la centralización de actividades. Con la utilización del módulo Spring MVC se pone de manifiesto este patrón.

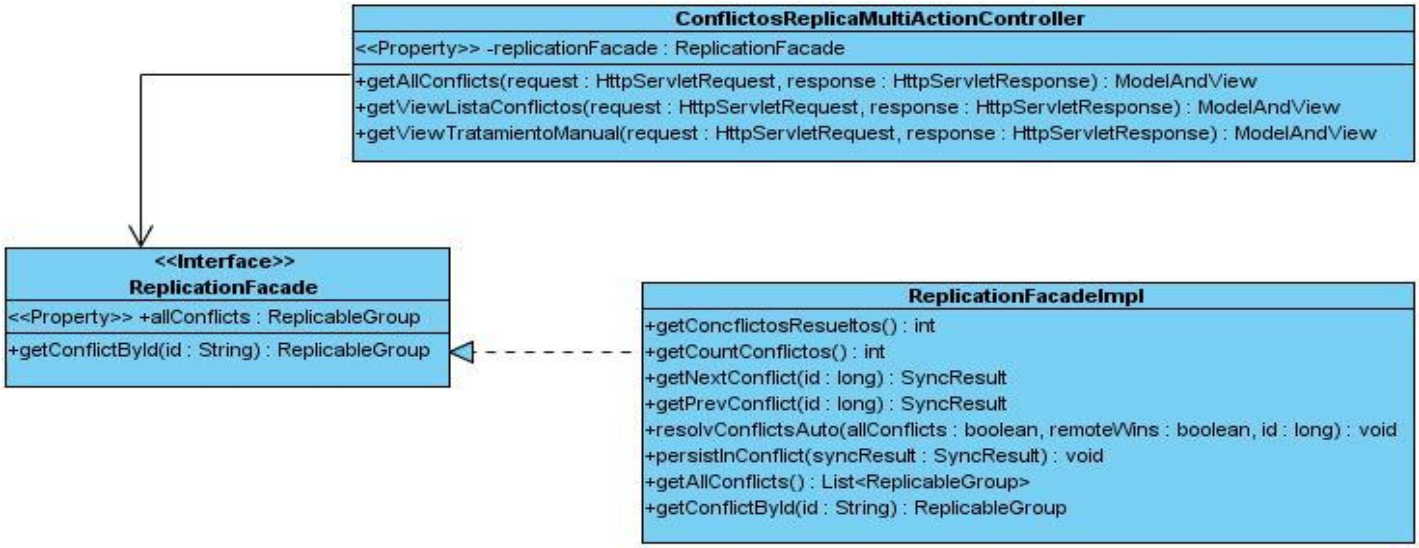


Figura 3.5 Patrón Controlador

Patrones GOF:

- **Estructurales:** Describen cómo las clases y los objetos pueden ser combinados para formar grandes estructuras y proporcionar nuevas funcionalidades.
  - ✓ **Fachada (*Facade*):** Provee de una interfaz unificada simple, para acceder a una interfaz o grupo de interfaces de un subsistema. Es utilizado para reducir la dependencia entre clases, ofreciendo un punto de acceso, de manera que si estas cambian o se sustituyen por otras, solo hay que actualizar la clase Fachada sin que el cambio afecte a las aplicaciones cliente. Fachada no oculta las clases, sino que ofrece una forma más sencilla de acceder a las mismas y en los casos que se requiera, permite acceder directamente a ellas.

Figura 3.6 Patrón Fachada

3.4 Diagrama de clases del diseño

Un diagrama de clases es un tipo de diagrama estático que describe la estructura de un sistema; también muestra sus clases, atributos y las relaciones entre ellos. Los diagramas de clases son utilizados durante el proceso de análisis y diseño de los sistemas, cuando se crea el diseño conceptual de la información que se manejará en el sistema, conjuntamente con los componentes que se encargarán del funcionamiento y la relaciones entre unos y otros. Debido al uso del framework Spring, se introduce para el diseño el patrón arquitectónico Modelo-Vista-Controlador. Es por esta razón, que los diagramas de clases del diseño representan las interacciones entre las clases del sistema, cumpliendo con el patrón arquitectónico definido. Este diseño está compuesto por tres paquetes (Modelo, Vista y Controlador), donde se encuentran las clases controladoras, las páginas clientes y las clases entidades. A continuación se muestran los diagramas referentes al diseño de la funcionalidad.

Diagrama de Clases del Diseño para el CU Capturar Conflicto.

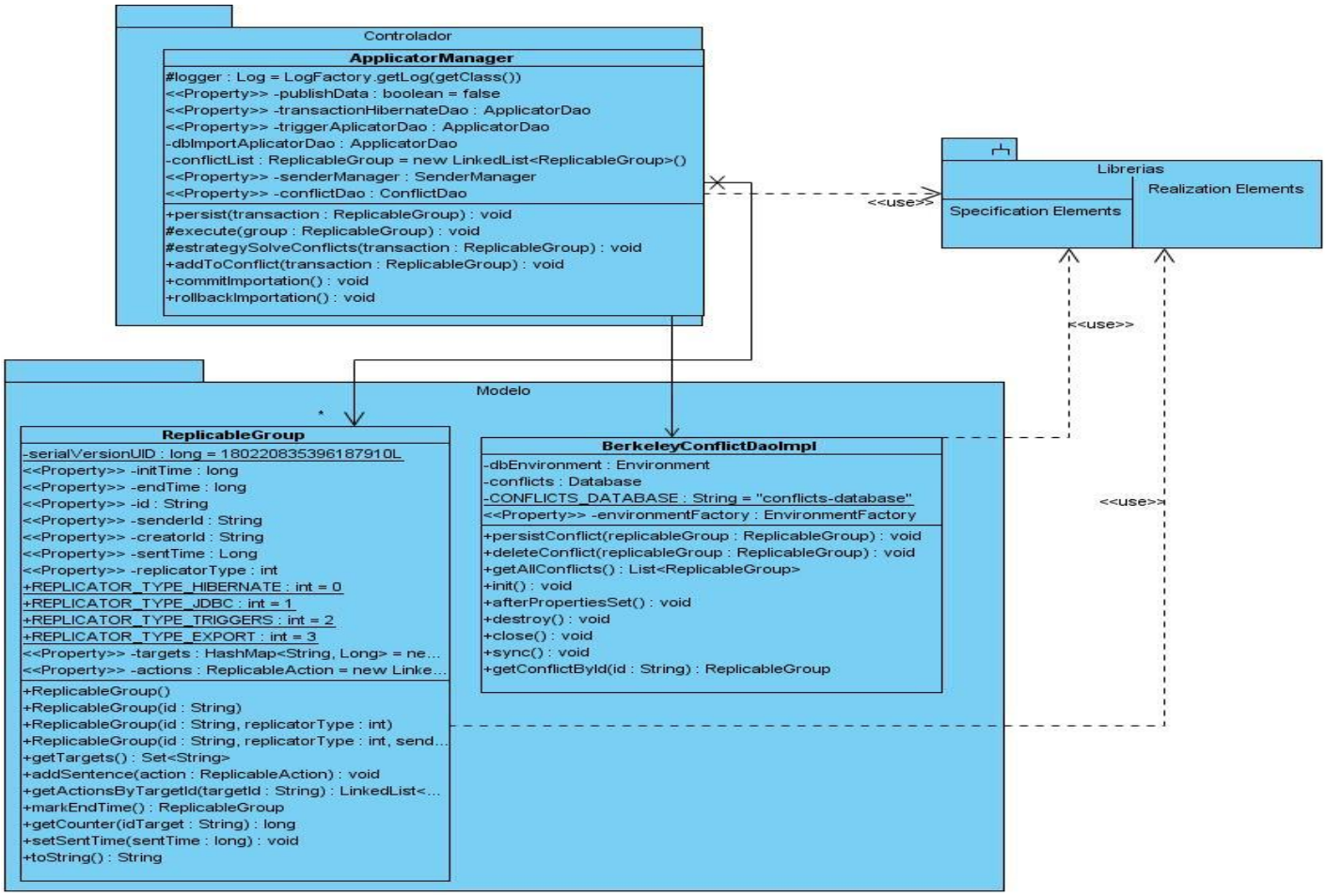


Diagrama de Clases del Diseño para el CU Listar Conflicto.

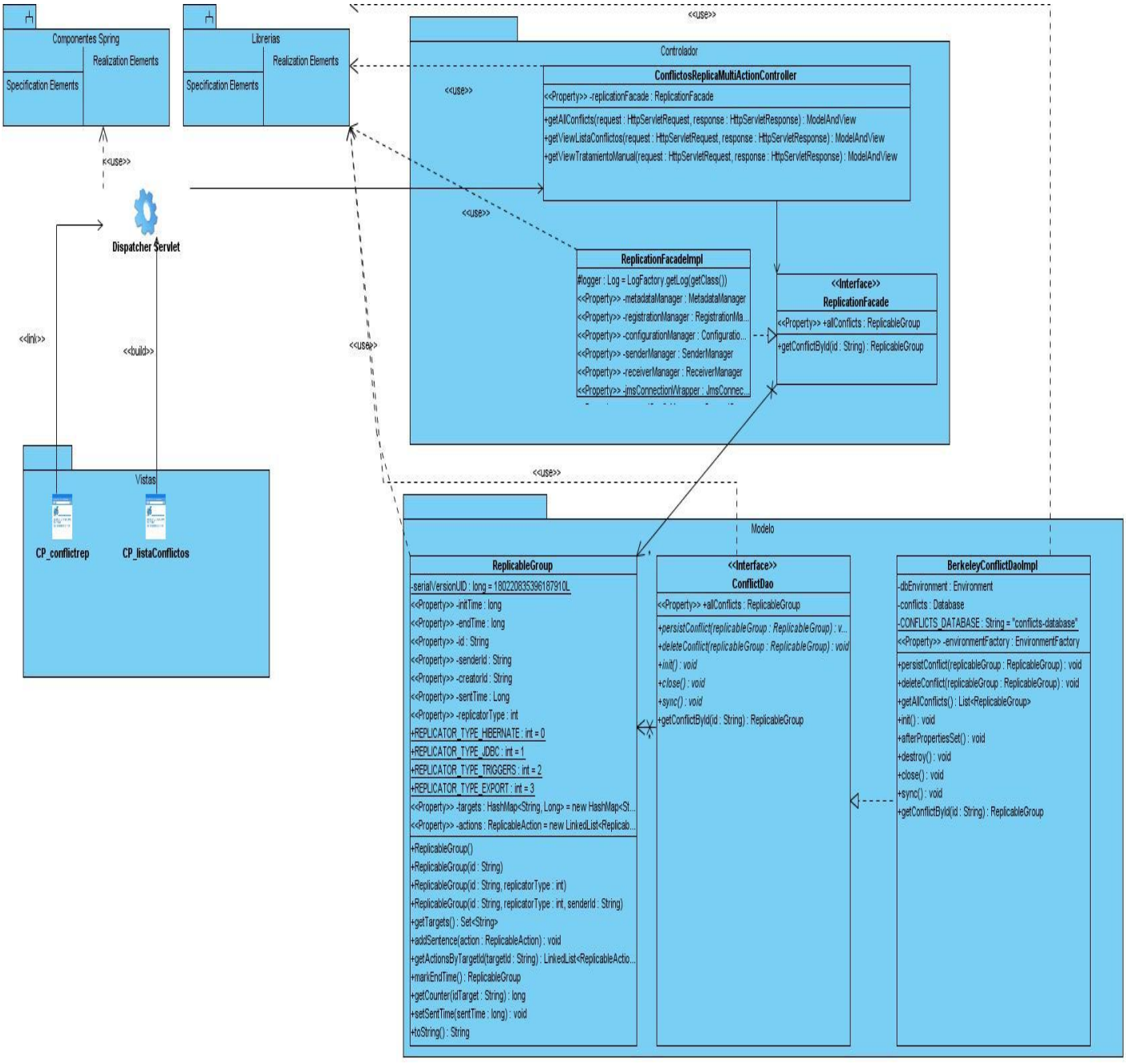


Diagrama de Clases del Diseño para el CU Resolver Conflicto Automáticamente.

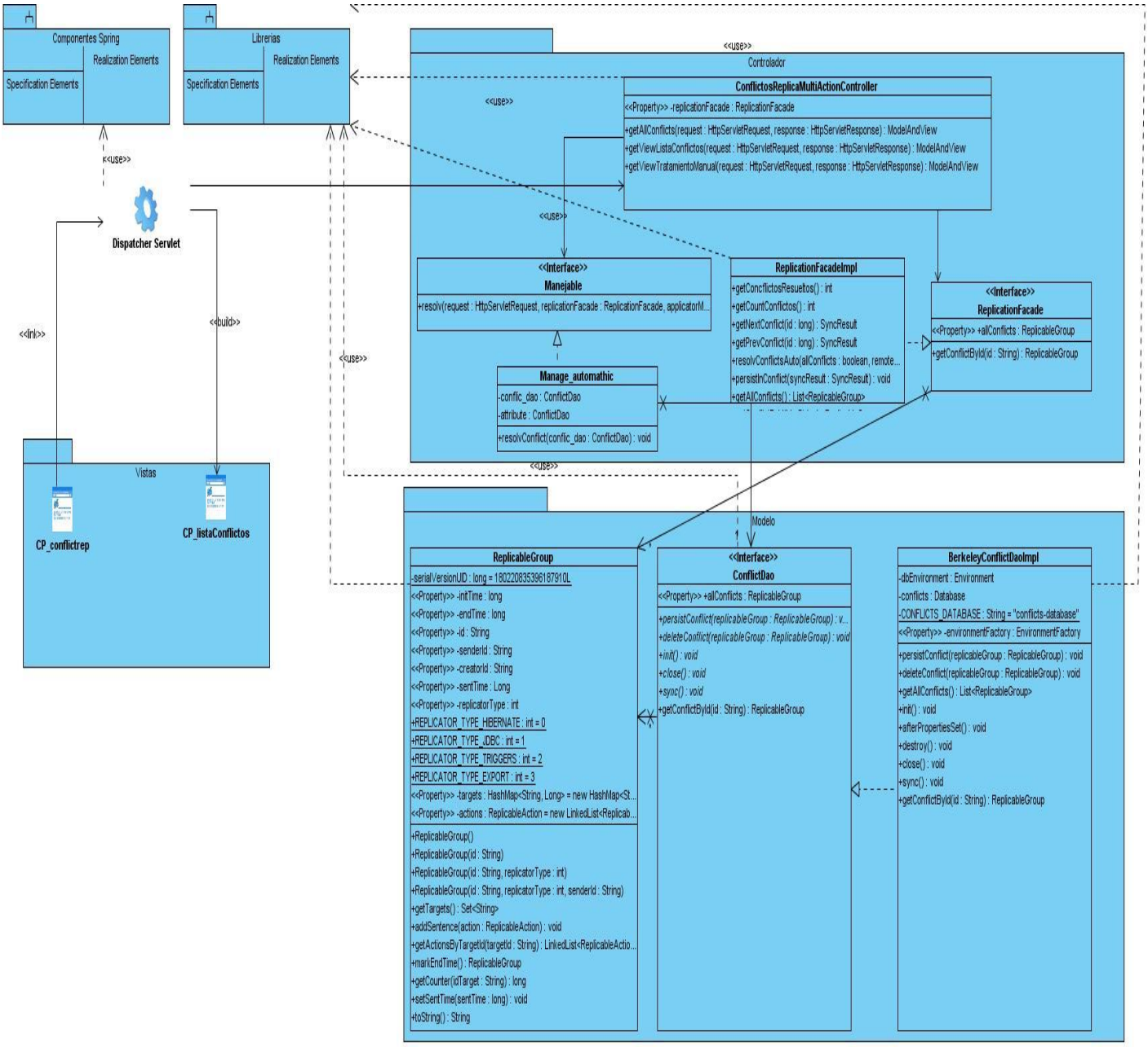
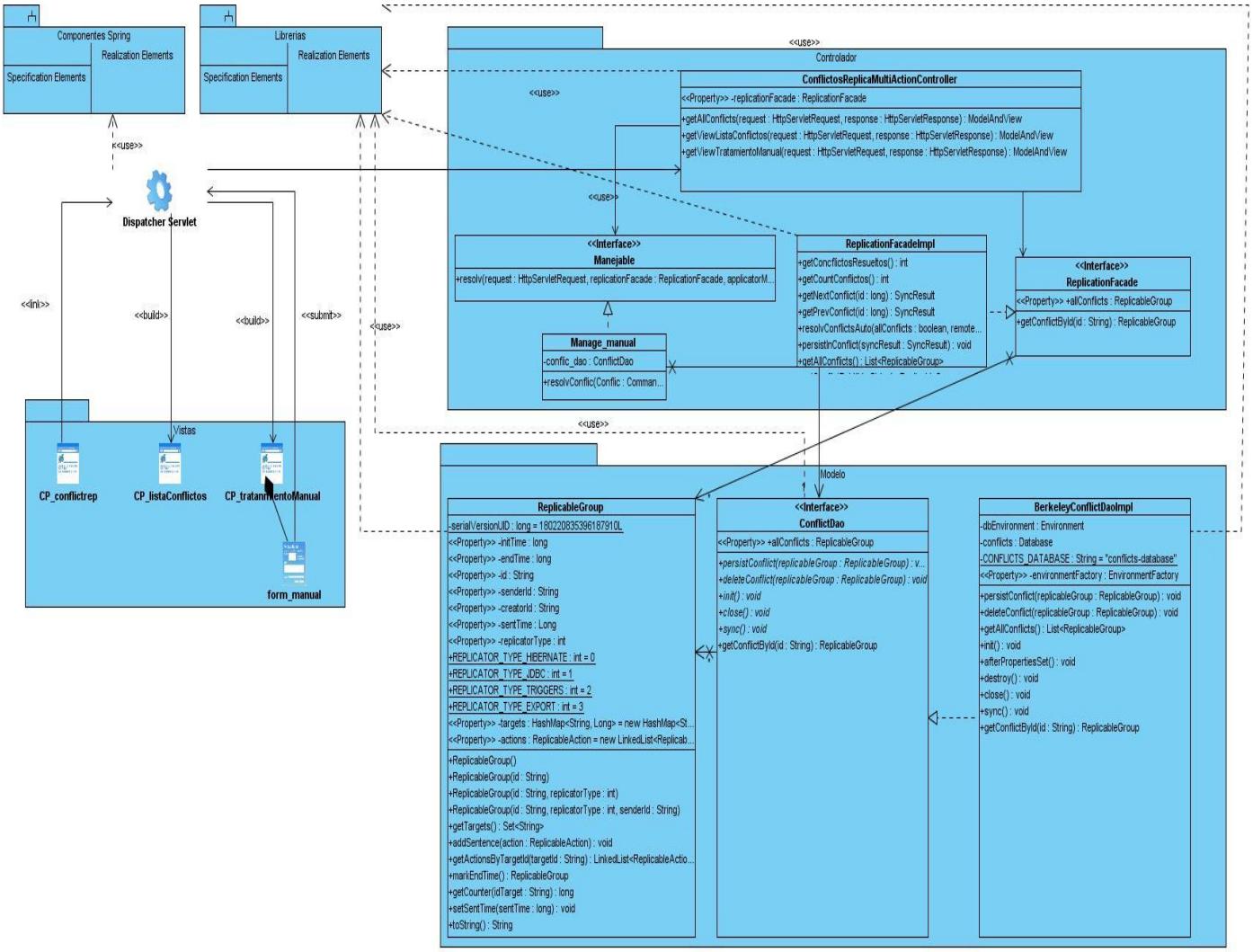


Diagrama de Clases del Diseño para el CU Resolver Conflicto Manualmente.



3.4.1 Diagramas de Secuencia.

Los diagramas de secuencia se utilizan para modelar los aspectos dinámicos de un sistema. La mayoría de las veces, esto implica modelar instancias concretas o prototípicas de clases, interfaces, componentes y nodos, junto con los mensajes enviados entre ellos, todo en el contexto de un escenario que ilustra un comportamiento.

Diagrama de Secuencia para el CU Capturar Conflicto.

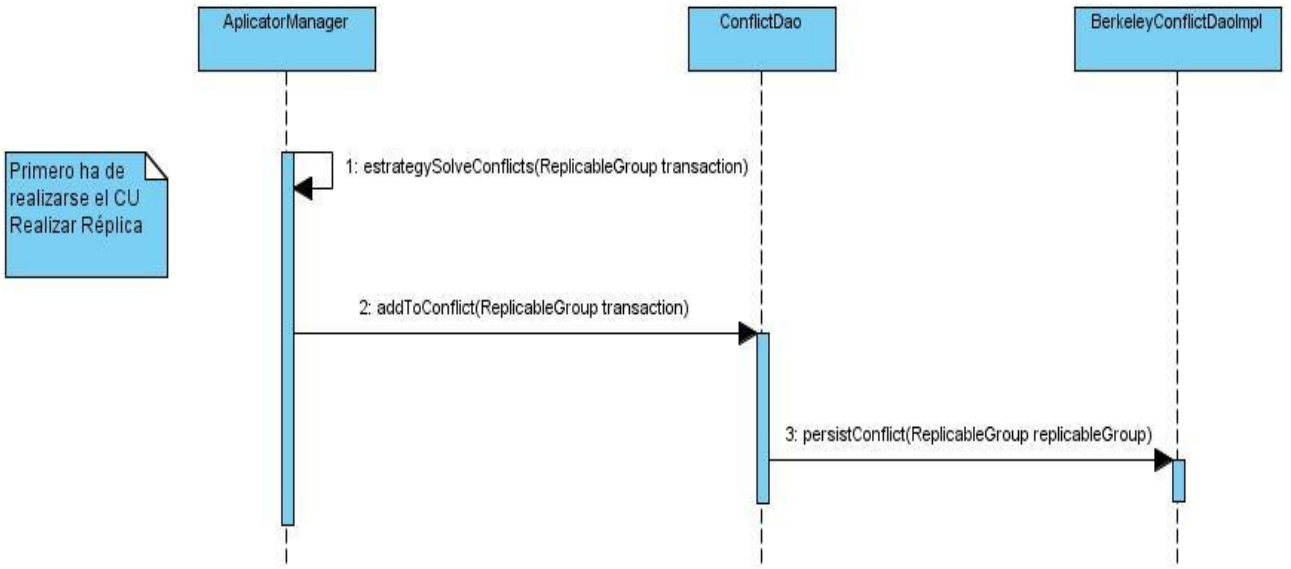


Diagrama de Secuencia para el CU Listar Conflictos.

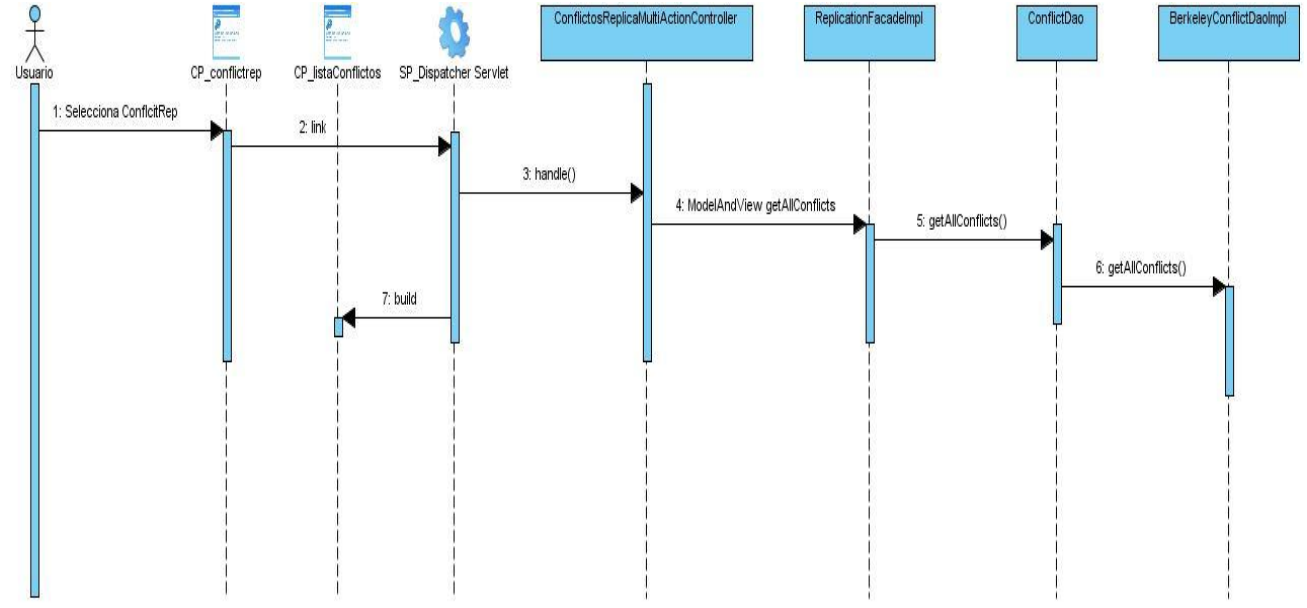


Diagrama de Secuencia para el CU Resolver Conflicto Manualmente.

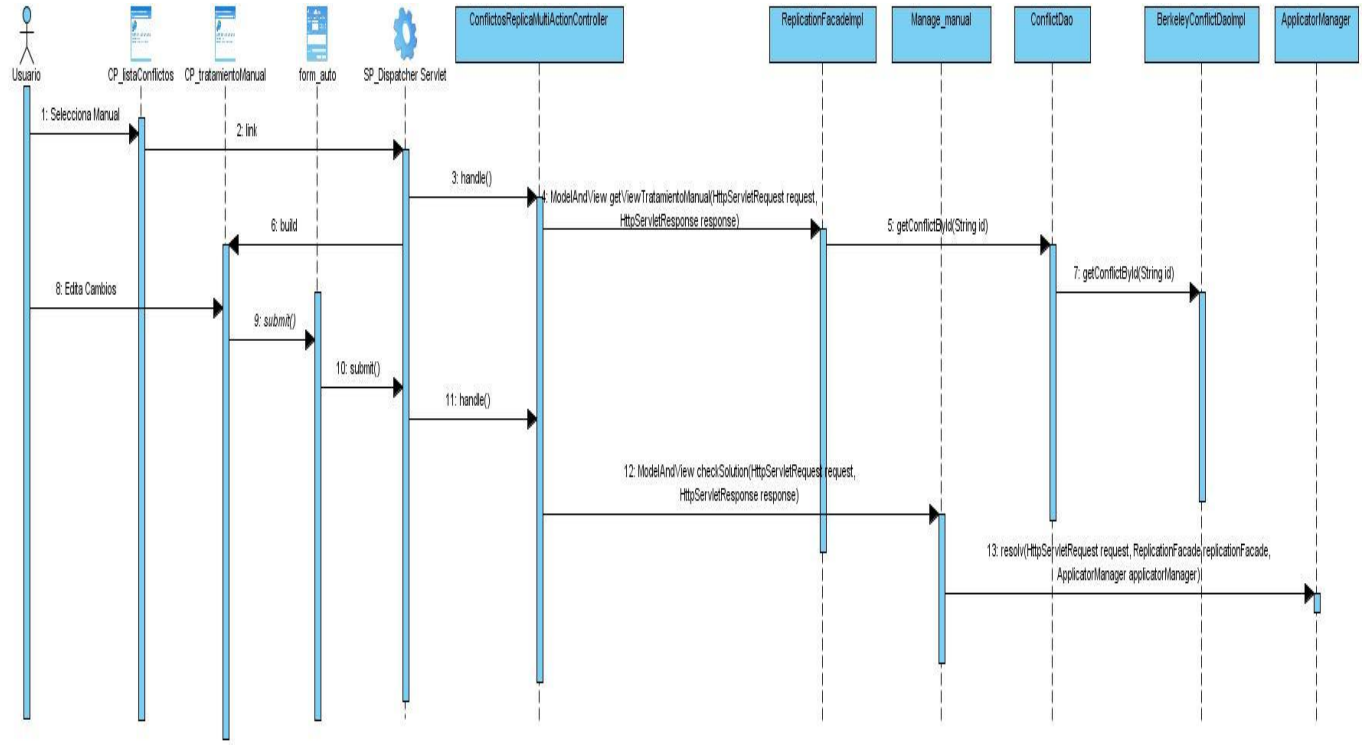
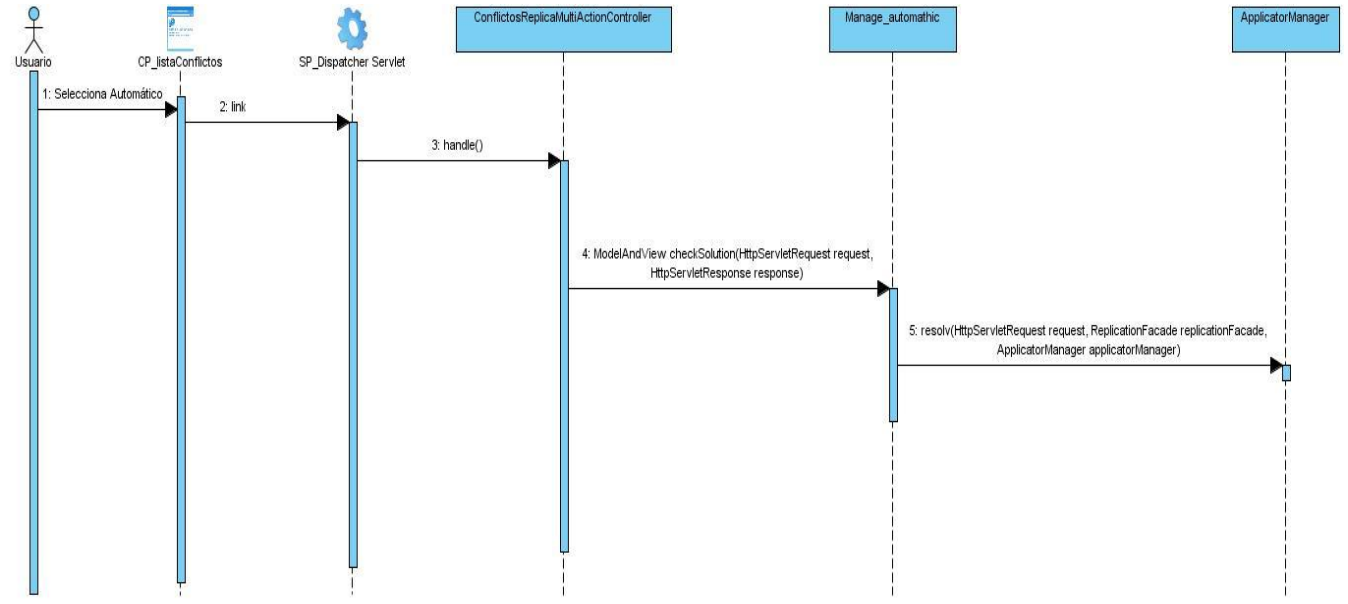
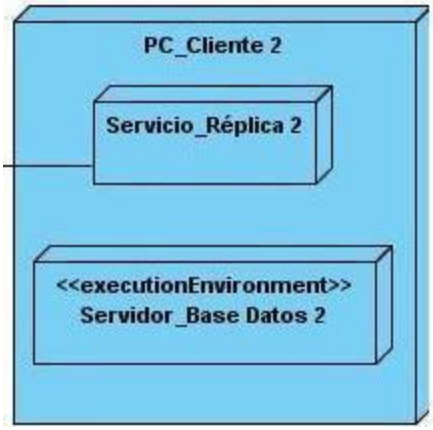


Diagrama de Secuencia para el CU Resolver Conflicto Automáticamente.



3.5 Modelo de Despliegue

El modelo de despliegue es utilizado para capturar los elementos de configuración del procesamiento y las conexiones entre esos elementos. También se utiliza para visualizar la distribución de los componentes de software en los nodos físicos.



3.6 Conclusiones

Fueron definidas las clases que se corresponden con el diseño del sistema, se realizaron los diagramas de clases para los casos de uso identificados, así como sus realizaciones del diseño. Además, se identificó el estilo arquitectónico y se fundamentaron varios de los patrones de diseño utilizados y además se muestra de forma breve el modelo de despliegue.

Capítulo 4: Implementación y pruebas.

4.1 Introducción

En este capítulo se describe la implementación del sistema, además de los diagramas de componentes, se mostrarán imágenes de la aplicación así como las pruebas realizadas al sistema.

4.2 Diagramas de Componentes

Los diagramas de componentes muestran las organizaciones y dependencias lógicas entre componentes de software, sean estos de código fuente, binarios, archivos, bibliotecas cargadas dinámicamente o ejecutables En el diagrama de componentes se tienen en consideración los requisitos relacionados con la facilidad de desarrollo, la gestión del software, la reutilización, y las restricciones impuestas por los lenguajes de programación y las herramientas utilizadas en el desarrollo. A continuación se muestran los diagramas de componentes de la aplicación desarrollada:

Diagrama de componentes: CU Capturar Conflicto

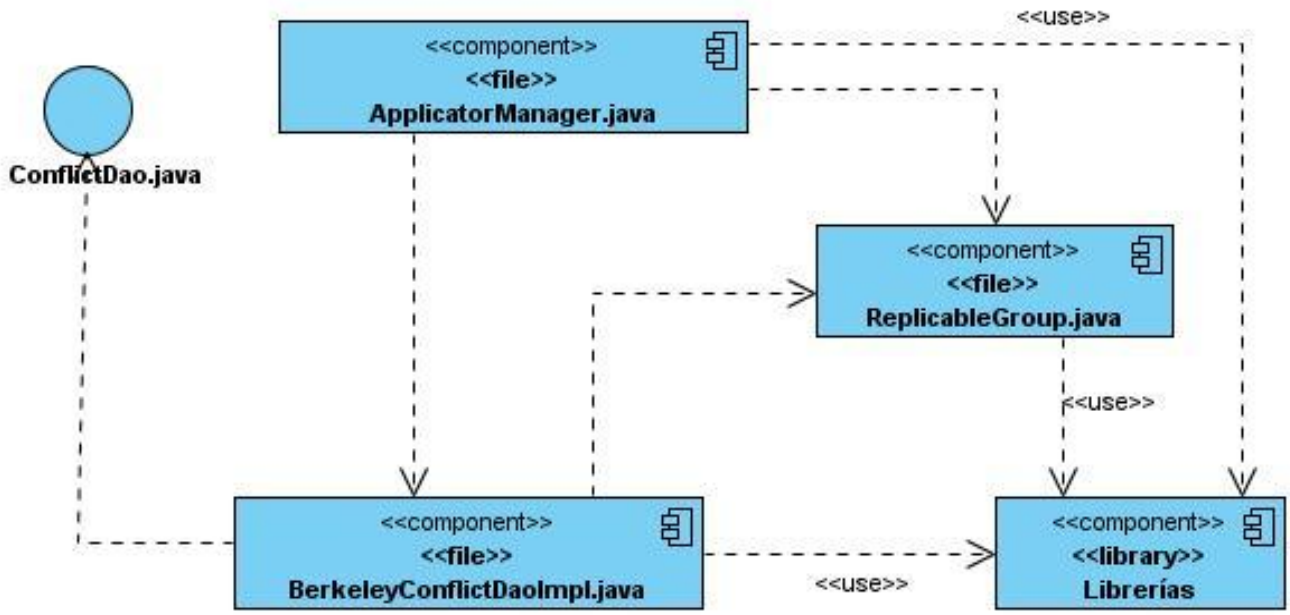


Diagrama de componentes: CU Listar Conflictos.

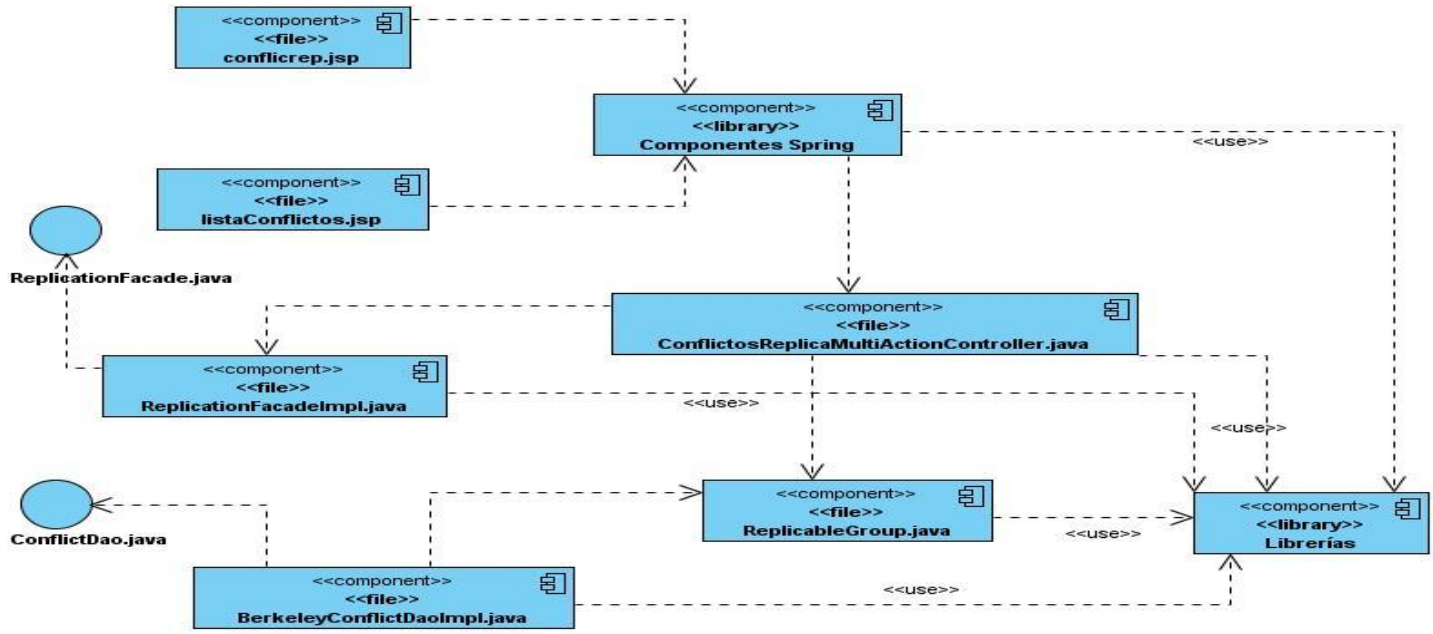


Diagrama de componentes: CU Resolver Conflictos Manualmente.

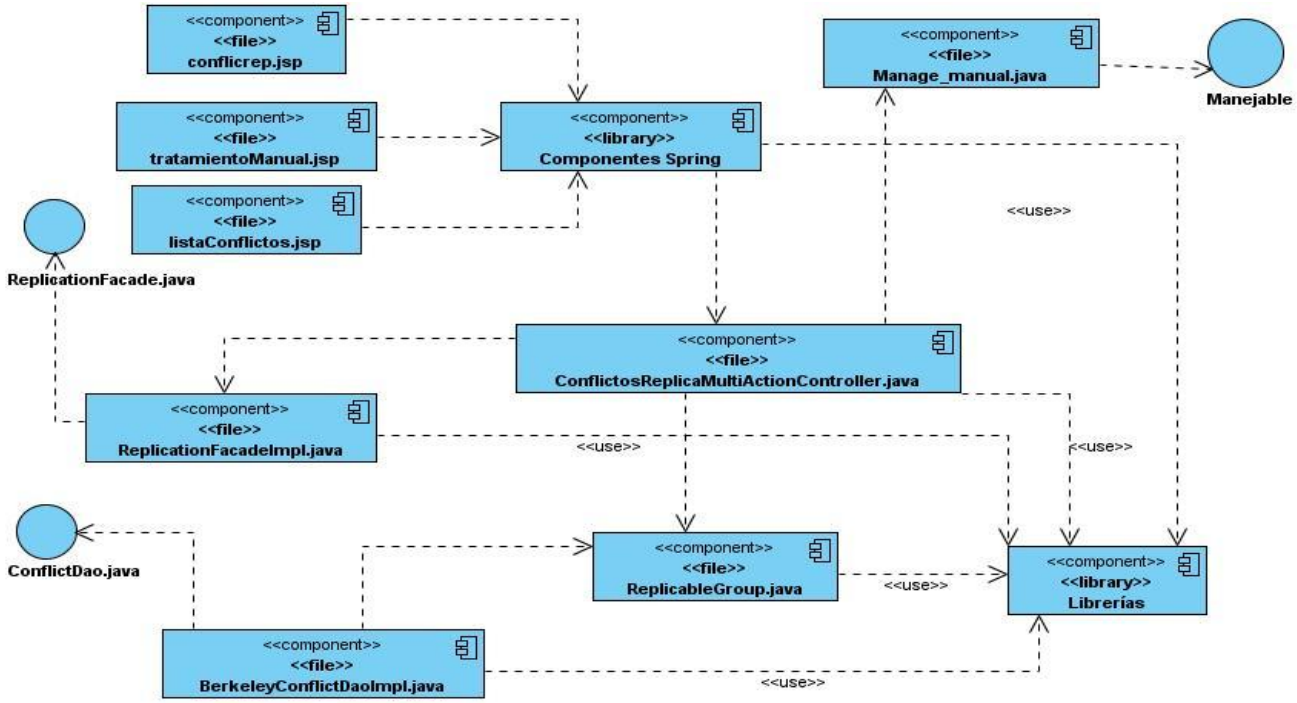
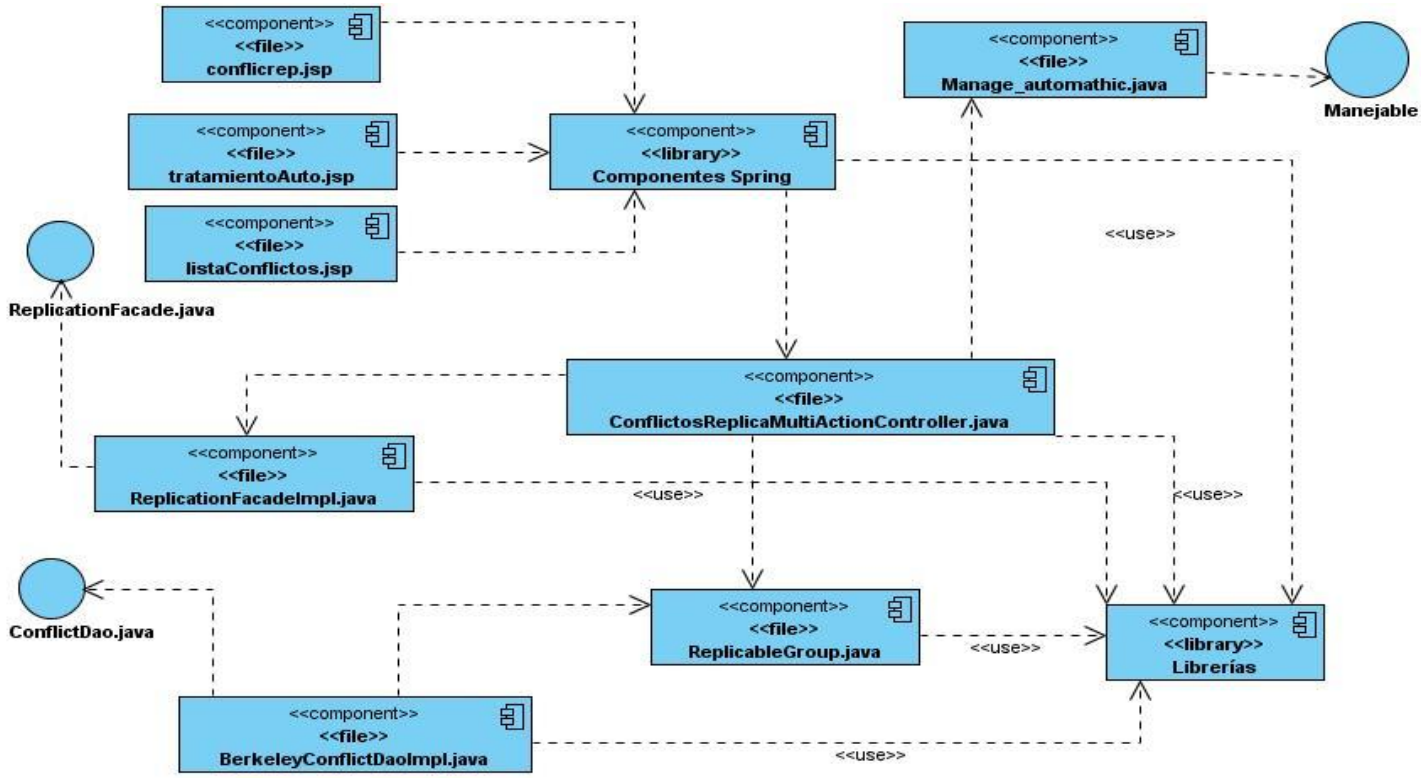


Diagrama de componentes: CU Resolver Conflictos Automáticamente.



4.3 Pruebas del sistema.

4.3.1 Plan de Prueba.

Un plan de pruebas está constituido por un conjunto de pruebas. Cada prueba debe dejar claro: qué tipo de propiedades se quieren probar (corrección, robustez, fiabilidad, amigabilidad); cómo se mide el resultado; especificar en qué consiste la prueba (hasta el último detalle de cómo se ejecuta) y definir cuál es el resultado que se espera. En lo adelante se abordara todo lo relacionado con las pruebas realizadas al sistema.

4.3.1.1. Configuración del entorno de Prueba

La configuración del entorno donde se vayan a ejecutar las diferentes pruebas que se realizan a un software es un aspecto muy importante dentro del proceso de pruebas, pues si no se analizan bien los recursos de software y hardware que necesita el producto que se está construyendo, a la hora de probarlo se prescindirá de los elementos necesarios para la ejecución de un proceso de pruebas exitoso.

Por ello se tuvo en cuenta algunos requerimientos de hardware y de software que hicieron posible un mejor desarrollo de las pruebas, en aras de que se lograran minimizar los errores de la aplicación en desarrollo. Los requerimientos que se consideraron necesarios para las pruebas se referencian a continuación:

Requerimientos de software:

· PC con Windows XP o superior.
· PC con Linux (Ubuntu).
· Máquina Virtual de Java 1.6.
- Servidor JMS.
- Servidor de base de datos PostgreSQL.

Requerimientos de hardware:

· PC con Microprocesador Pentium IV o superior.
· Se requiere para Windows XP Professional (SP1) una memoria de 64MB y 98MB.
· Linux una memoria de 64MB y espacio en el disco de 58 MB

4.3.2. Diseño de las Pruebas de Caja Negra

Para desarrollar las pruebas al sistema se decidió utilizar el método de caja negra, que se refiere a las pruebas llevadas a cabo sobre la interfaz del software. Es decir, a través de ellas se pretende demostrar que las funciones del software son operativas, que la entrada se acepta de forma adecuada y que se produce un resultado correcto, así como que la integridad de la información externa se mantiene. En general, este método se centra principalmente en los requisitos funcionales del software [13].

4.3.2.1 Diseño de las pruebas del Replicador Reko.

Caso de Prueba Resolución Manual

Descripción General: Solucionar los conflictos de manera manual.

CU: Resolver Conflicto.

1. **CPRF 3.2:** Resolver manualmente.

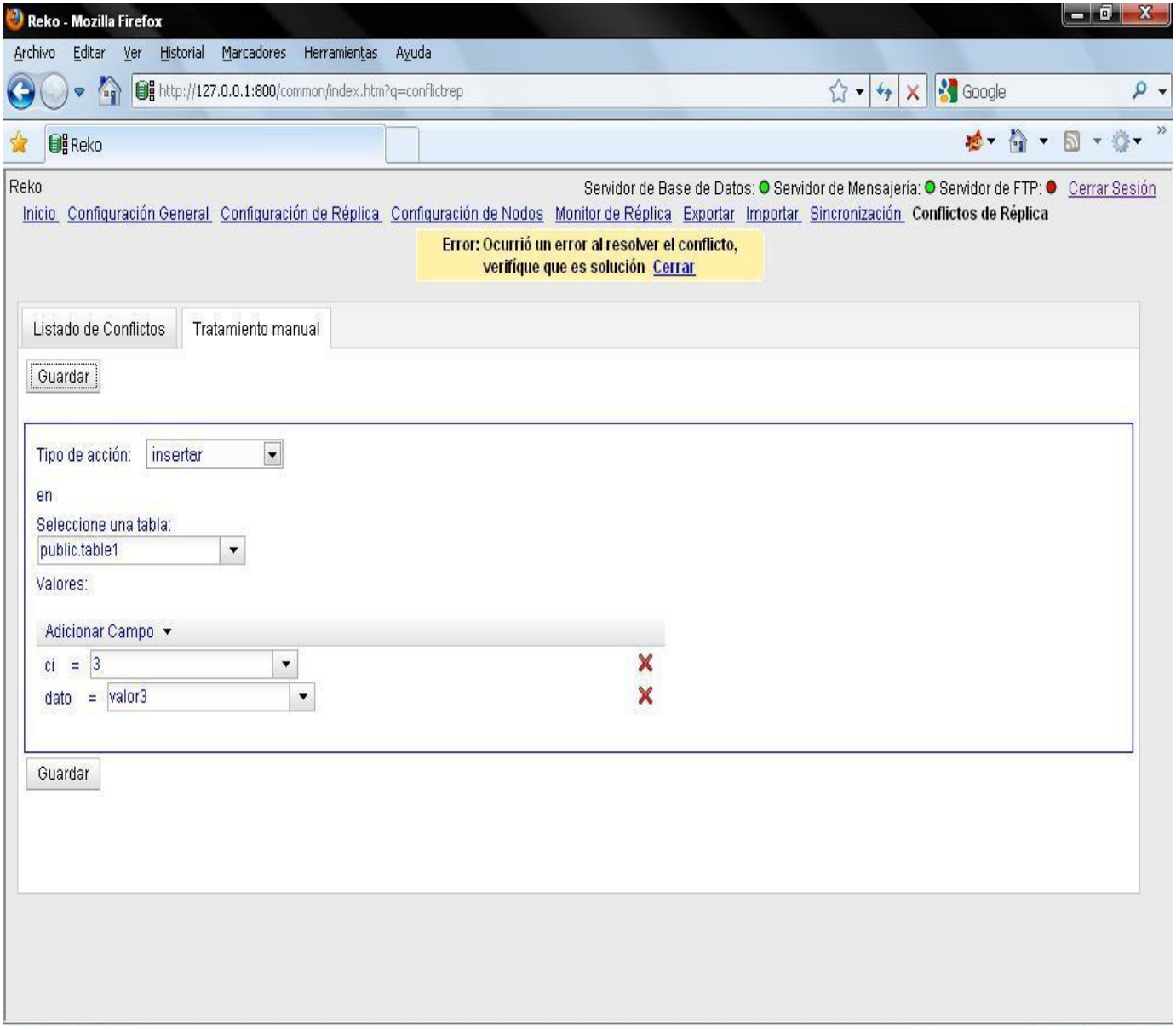
2. **Descripción:** El caso de prueba se inicia cuando se captura un conflicto en la réplica de datos y se dispone a darle tratamiento.

2.1 Flujo Central: El usuario selecciona la opción Manual, el sistema muestra una interfaz donde mostrará un editor de acciones con el conflicto capturado. El usuario edita los datos que generan conflicto y acepta para comprobar la réplica. El sistema incluye los datos de la modificación dentro de los datos a replicar.

2.2 Condiciones de Ejecución

Clases válidas	Clases inválidas	Resultado Esperado	Resultado de la prueba
Se realizan los cambios correspondientes manualmente en el SQL con conflictos.		El SQL modificado se debe replicar correctamente en la base de datos.	El SQL modificado se replica correctamente en la base de datos.
Se realizan los cambios correspondientes manualmente en el SQL con conflictos.	El usuario escoge un id que ya existe en la base de datos u otro dato incorrecto.	El SQL modificado no debe replicarse correctamente y se muestra un mensaje de error al resolver el conflicto.	El SQL modificado no se replica correctamente y se muestra un mensaje de error al resolver el conflicto.

Caso de Prueba Resolución Manual – Error al resolver el conflicto, campos requeridos incorrectos

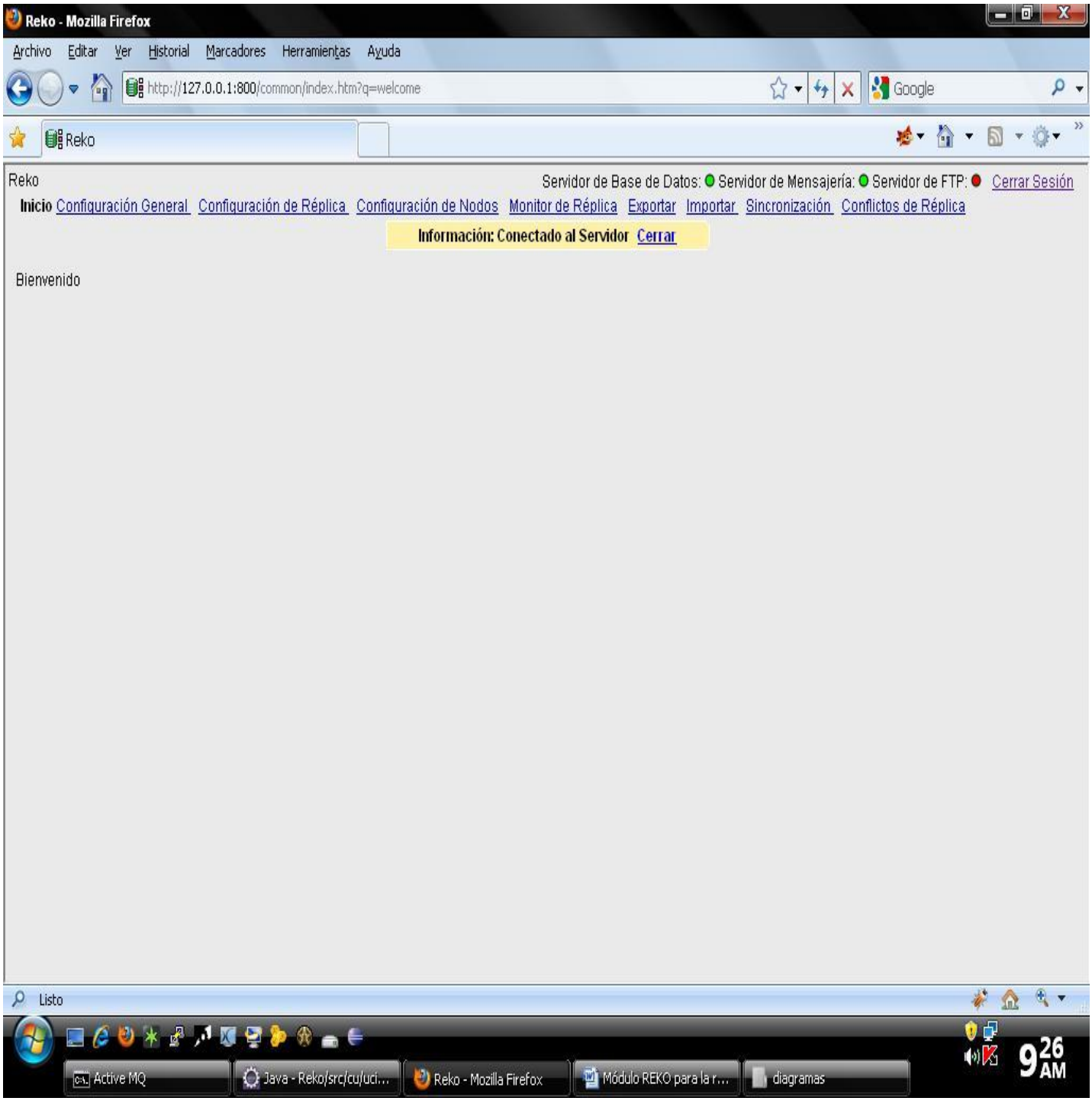


4.3.2.2 No conformidades Detectadas.

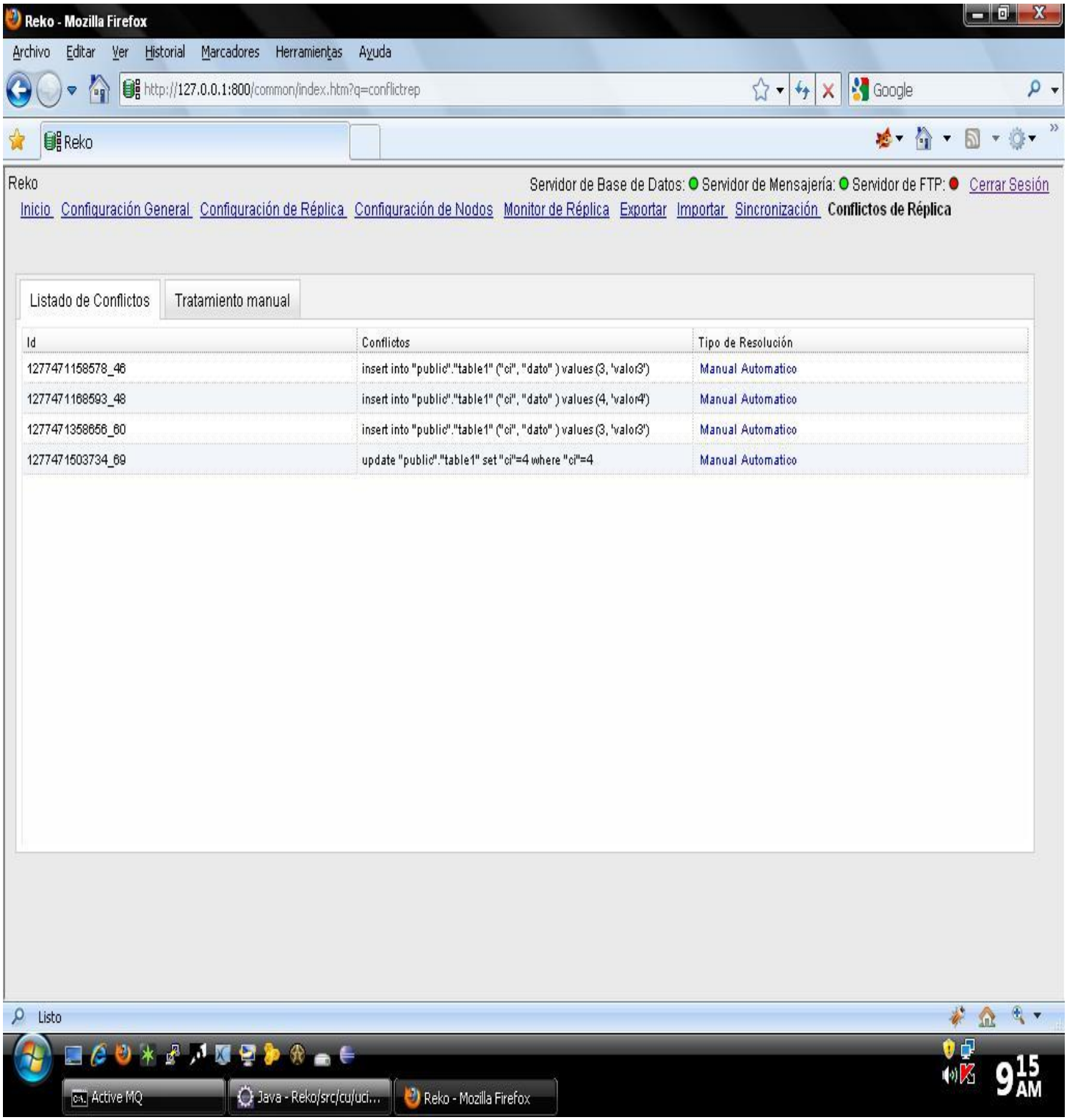
Elemento	Nº	No conformidad	Aspecto Correspondiente	Etapa de Detección	Significativa	No Significativa
Interfaz 1	1	No se mostraban correctamente los conflictos detectados en la réplica.	Cuando se mostraba el listado de los conflictos detectados.	Implementación	X	
Interfaz 1	2	Se podía escoger un mismo conflicto para las dos tipos de resolución.	Cuando se presionaba el vínculo con el tipo de resolución.	Implementación	X	
Interfaz 2	3	No se lograba guardar los cambios realizados a los SQL con conflictos.	Cuando se presionaba el botón guardar.	Implementación	X	

4.4 Pantallas de la aplicación.

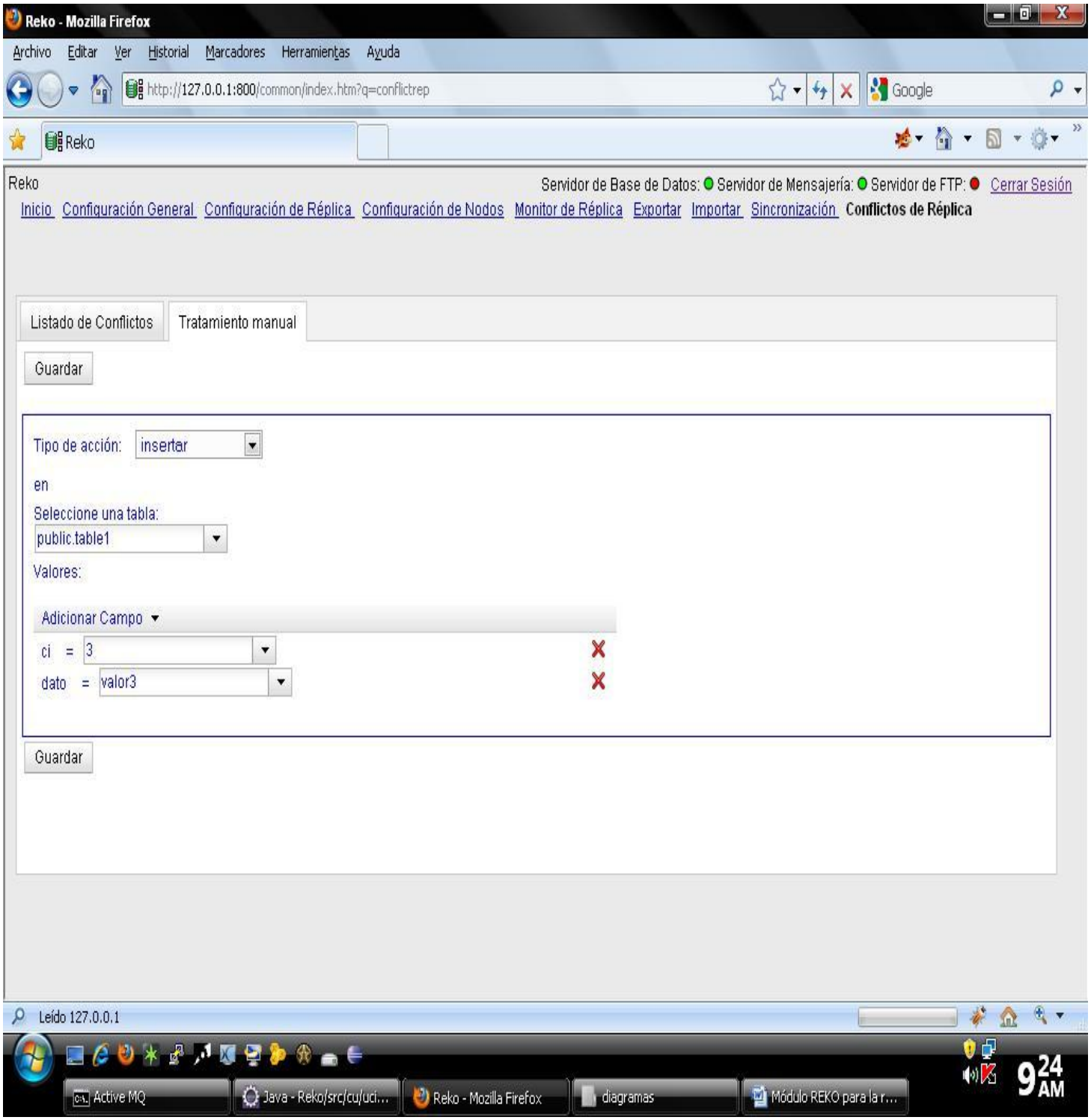
4.4.1 Interfaz principal.



4.4.2 Interfaz Listar Conflicto.



4.4.2 Interfaz Resolución Manual.



4.5 Conclusiones

Se logró implementar una parte de la funcionalidad para la resolución de conflictos, específicamente la resolución manual y se integró al software de replicación Reko. Se detectaron algunas no conformidades las cuales fueron registradas y solucionadas para un mejor funcionamiento del software, que verifican el cumplimento de algunos de los requerimientos funcionales identificados. Se le realizaron pruebas de caja negra al sistema especialmente a la funcionalidad de resolución manual arrojando un resultado satisfactorio.

Conclusiones.

Con la confección de este Trabajo de Diploma se obtuvo un módulo para la resolución de conflictos, donde se puede observar como resultado:

- Se desarrolló la funcionalidad que permite la resolución de conflictos en el Replicador Reko de forma manual.
- Con la implementación de la funcionalidad, se obtuvo un producto que cumple con los requerimientos establecidos.
- La funcionalidad obtenida contribuirá a un mejor funcionamiento del Replicador dándole la posibilidad al usuario de resolver manualmente conflictos que surgen al realizar la réplica.
- Se validó la funcionalidad realizándose pruebas de caja negra.

Por todo lo anteriormente expuesto se concluye que los objetivos trazados para el presente trabajo se han cumplido parcialmente.

Recomendaciones.

Se recomienda la implementación de la funcionalidad para la resolución automática de los conflictos en el Replicador Reko.

Referencias Bibliográficas.

1. Steffen, Hermann. (2003). Técnicas Avanzadas para Gestión de Sistemas de Información. [Citado el: 18 de Febrero de 2010.]
2. Daffodil Software. [Citado el: 15 de Enero de 2010.] Disponible en http://enterprise.replicator.daffodilsw.com/
3. DbReplicator. [Citado el: 21 de enero de 2010.] Disponible en http://dbreplicator.org/
4. PyReplica. [Citado el: 5 de Febrero de 2010.] Disponible en http://blog.unnoba.edu.ar/wp-content/uploads/2009/06/pgdaypyreplica.pdf.
5. Forge. [Citado el: 10 de Febrero de 2010.] Disponible en https://forge.uci.cu/gf/project/reko/docman..
6. Drools [Citado el: 4 marzo de 2010] Disponible en http://www.planetacodigo.com/planeta/1308/drools-i-introduccion-a-los-motores-de-reglas-de-negocios
7. Balduino, Ricardo. Introduction to OpenUP (Open Unified Process). Eclipse. [Citado el: 18 de Febrero de 2010.] Disponible en http://www.eclipse.org/epf/general/OpenUP.pdf
8. Cruzado Nuño, Ignacio. Guía de iniciación al lenguaje Java. Versión 2.0, Noviembre de 1999. [Citado el: 6 de Marzo de 2010.] Disponible en http://pisuerga.inf.ubu.es/lsi/Invest/Java/Tuto/java20.pdf
9. Eclipse. The Eclipse Foundation. [Citado el 6 de marzo de 2010] Disponible en http://www.eclipse.org/
10. Requerimientos.[Citado el 10 de marzo de 2010] Disponible en http://www.mitecnologico.com/Main/EspecificacionesDeRequerimientos
11. Modelo Vista Controlador (MVC). [Citado el 9 de abril de 2010] Disponible en http://prof.usb.ve/lmendoza/Documentos/PS-6116/Teor%EDa%20PS6116%20Arq.%20de%20Software.pdf
12. Patrones de diseño. [Citado el 11 de marzo de 2010] Disponible en http://www.slideshare.net/jmruizforem/patrones-de-diseo-en-e-learning
13. Asignatura: Ingeniería de Software II Curso: 2005-2006 Conferencia 4. Título: Flujo de trabajo de Prueba. [Citado el 4 de junio de 2010].

Bibliografía.

- Steffen, Hermann. (2003). Técnicas Avanzadas para Gestión de Sistemas de Información.
- Daffodil Software. Disponible en http://enterprise.replicator.daffodilsw.com/
- DbReplicator. Disponible en http://dbreplicator.org/
- PyReplica. Disponible en http://blog.unnoba.edu.ar/wp-content/uploads/2009/06/pgdaypyreplica.pdf.
- Forge. Disponible en https://forge.uci.cu/gf/project/reko/docman..
- Drools Disponible en http://www.planetacodigo.com/planeta/1308/drools-i-introduccion-a-los-motores-de-reglas-de-negocios
- Balduino, Ricardo. Introduction to OpenUP (Open Unified Process). Eclipse. Disponible en http://www.eclipse.org/epf/general/OpenUP.pdf
- Cruzado Nuño, Ignacio. Guía de iniciación al lenguaje Java. Versión 2.0, Noviembre de 1999. Disponible en http://pisuerga.inf.ubu.es/lsi/Invest/Java/Tuto/java20.pdf
- Eclipse. The Eclipse Foundation. Disponible en http://www.eclipse.org/
- Requerimientos Disponible en http://www.mitecnologico.com/Main/EspecificacionesDeRequerimientos
- Modelo Vista Controlador (MVC). Disponible en http://prof.usb.ve/lmendoza/Documentos/PS-6116/Teor%EDa%20PS6116%20Arq.%20de%20Software.pdf
- Patrones de diseño. Disponible en http://www.slideshare.net/jmruizforem/patrones-de-diseo-en-e-learning
- Asignatura: Ingeniería de Software II Curso: 2005-2006 Conferencia 4. Título: Flujo de trabajo de Prueba.
- Microsoft Development Network disponible en http://msdn.microsoft.com/es-es/library/cc761628.aspx

  http://technet.microsoft.com/es-es/library/ms152576%28SQL.90%29.aspx
- Microsoft Development Network disponible en http://msdn.microsoft.com/es-es/library/ms165742%28SQL.90%29.aspx
- Artículo titulado: Nueva Versión de JBoss rules disponible en http://www.xnoccio.com/159-nueva-version-de-jboss-rules/

Glosario de Términos.

1- Hibernate.

Es una herramienta de Mapeo objeto-relacional para la plataforma Java (y disponible también para .Net con el nombre de NHibernate) que facilita el mapeo de atributos entre una base de datos relacional tradicional y el modelo de objetos de una aplicación, mediante archivos declarativos (XML) que permiten establecer estas relaciones.

Hibernate es software libre, distribuido bajo los términos de la licencia GNU LGPL.

2- Triggers.

Un trigger (o disparador) en una Base de datos , es un procedimiento que se ejecuta cuando se cumple una condición establecida al realizar una operación de inserción (INSERT), actualización (UPDATE) o borrado (DELETE).

3- Gestores soportados: Se integra actualmente con los gestores de bases de datos *Oracle* y *PostgreSQL*.

4- Selección de datos: Provee la facilidad de seleccionar el subconjunto de tablas y datos a ser replicados de la base de datos mediante la definición de filtros aplicables a las tablas y la selección de usuarios propios del gestor.

5- Transmisión de datos: La transferencia de datos de replicación puede realizarse con soporte para replicación sobre TCP/IP, FTP, HTTP o por ficheros de forma manual. Los archivos de gran tamaño son enviados por FTP permitiendo resumir la trasmisión en caso de interrupciones en la red.

6- Configuración entre nodos: El mecanismo de registro entre nodos de replicación se basa únicamente en un identificador (id) del nodo lo que permite la abstracción de los datos físicos de cada nodo como son el IP y el protocolo de comunicación. Este mecanismo permite que los nodos puedan moverse por distintas subredes y mantener sus datos sincronizados. Por ejemplo, mantener sincronizada la base de datos de una laptop con la base de datos central a través de Internet.

7- Seguridad: Los nodos de replicación manejan credenciales entre ellos para verificar la autenticidad de los datos transferidos. El envío de datos se realiza utilizando protocolos de comunicación seguros como son hyper text transfer protocol secure(HTTPS) y secure socket layer (SSL).
8- Monitoreo: Permite conocer el estado de los datos transmitidos, puede realizarse en tiempo real a través de la Web así como dar un seguimiento al funcionamiento interno del mecanismo.
9- Interfaz visual Web: La administración y configuración de la réplica se realiza a través de una interfaz Web, por lo que es administrable vía remota usando solamente un navegador.
10- Independiente de la plataforma: El software de réplica puede ser instalado en cualquier sistema operativo y no necesita de un ambiente gráfico en el mismo para su funcionamiento.
11- Tolerancia a fallos: Detecta errores de conexión. Mantiene los datos de réplica en un estado estable en caso de desconexión. Al restablecerse la conexión, automáticamente sincroniza los datos entre las bases de datos y continúa con la réplica.