Universidad de las Ciencias Informáticas

Ciudad de La Habana, junio de 2010

"Año 52 de la Revolución"

Título: "Análisis, diseño e implementación de una herramienta que permita la centralización de la información gestionada por el módulo Resultados de la Colección Multisaber"

Autor: Rafael Jacobo Hidalgo Urbino

Tutor: Lic. Héctor Matías González

Co-tutores: Ing. Yonnys Pablo Martín Olivera

Ing. Geyser Zamora Sánchez

Declaración de autoría

Declaro que soy el único autor del trabajo *"Análisis, diseño e implementación de una herramienta que permita la centralización de la información gestionada por el módulo Resultados de la Colección Multisaber"* y autorizo a la Universidad de las Ciencias Informáticas (UCI) a que haga el uso que estime pertinente con el mismo.

Para que así conste firmo la presente a los ____ días del mes de junio del año 2010.

Autor:

Rafael Jacobo Hidalgo Urbino

_____

Tutor:

Lic. Héctor Matías González

_____

Co-tutores:

Ing. Yonnys Pablo Martín Olivera

_____

Ing. Geyser Zamora Sánchez

_____

“El arte nunca se termina, sólo se abandona.”
—Leonardo Da Vinci

Dedicatoria y agradecimientos

A mi mamá por darnos tanto amor a mis hermanos y a mí, por quitarse lo que tiene para dárselo a sus hijos, por todas las enseñanzas que me ha dado, por todo su amor.

A mi papá que me ha enseñado a ser un hombre de principios, a ser responsable, a compartir todo con los que me rodean, a ser trabajador, honesto, estudioso, sacrificado y por todo el amor que me ha dado.

A mis hermanos Oscar e Isaac que me han apoyado mucho durante mi carrera y hemos vivido momentos inolvidables de la vida juntos. Espero tenerlos cerca toda la vida.

A mis abuelos maternos Gracia y Jacobo. A Gracia no la conocí pero le agradezco con la vida haber dado a luz a la persona que ha estado más cerca de mí durante toda la vida: mi mamá. Al abuelo Jacobo que lo extrañamos mucho y del cual heredé no sólo el nombre sino muchas cualidades humanas.

A mis abuelos paternos Delia y Oscar por haberme brindado tanto cariño y apoyo durante toda mi vida. Y haberme demostrado además de que la vida está llena de sacrificios y retos a los que hay que enfrentar con coraje.

A mis padrinos Mero y Angelo que siempre me han dado mucho amor, los quiero mucho.

A mi tía Martha por siempre estar pendiente de mis estudios, por haberse convertido en mi segunda madre durante los cinco años de mi carrera, por haberme acogido en su casa como un hijo.

A todos mis tíos y tías que son muchos y no sé cómo organizarlos en una lista, pero a todos los quiero por igual porque todos me han enseñado mucho en la vida.

A Sofía, que aunque ya sea un hombre, seguiré llamándola Mamafía por todo el cariño, el amor y la dedicación que ha tenido con mis hermanos y conmigo.

A Pitin y a Laudis por siempre estar ahí, por haberme dado tanto afecto durante toda mi vida.

A todos mis primos: Tania, las dos Mercy, Katerine, Rafe, Lazarito, Frank, Henry, Michel, Yovi, Sandro, Kolia, Gabriel, Pipo, Iradis, Leydis, Vanessa y a los hijos de ellos. Y si se me quedó alguno por mencionar también le agradezco mucho.

A mis amigos inseparables: Pepe, José Enriques y Roberto.

A Roberto por haberme mostrado el camino de la informática y del software libre.

A Jacqueline por todos sus consejos, por ser un ejemplo de responsabilidad profesional.

A mis amigos del barrio: Tony, Julito, Rodolfo, Daniel, Rauly, Yosvanis, Eduar y a todos los que se me quedaron.

A los amigos de la vocacional: Yoandry, Pedro, Armando, Liana, Yailín, Lilian, Ana Julia y todos los que no he mencionado.

A todos mis profesores que desde la primaria me han guiado. Muchas gracias por inculcarme tantos valores.

A la profe Liana por haberme permitido entrar en Multisaber desde 2do. año de la carrera.

Al equipo de béisbol de Holguín que con tanto afecto me acogió durante los meses que estuve con ellos en el 4to. año de la carrera, para mí siempre serán campeones.

A mis tutores Héctor, Yonnys y Geyser.

A Yadima que sin su ayuda no hubiera terminado mi tesis.

A mis dos grandes amigas de la UCI: Lisy y Yoly.

A todos los integrantes de mi grupo y amigos inseparables de la UCI que voy a extrañar muchísimo.

A todos los que se han ido y no están con nosotros en estos momentos.

A todos los que se me quedaron sin mencionar allá arriba.

A todos ellos, muchas gracias.

Resumen

A partir de un acuerdo con la República Bolivariana de Venezuela para la migración de 14 productos de la Colección Multisaber a software libre, se comenzaron a desarrollar sobre la plataforma web, aprovechando las ventajas que ofrece este entorno en cuanto a factibilidad del uso exclusivo de software libre. Los productos de la colección están divididos en módulos, uno de ellos, llamado Resultados, se encarga de recopilar la traza de los estudiantes que utilizan el producto.

A pesar de que las aplicaciones web facilitan su distribución y uso en la red, no todas las escuelas venezolanas, donde será desplegado el software, cuentan con la infraestructura necesaria para mantener redes estables. Sin embargo, es necesario lograr el funcionamiento del software en todas las escuelas con independencia de su infraestructura. Para ello se hace necesario el desarrollo de una aplicación capaz de centralizar los datos referentes a las trazas de los estudiantes hacia la máquina del profesor, aún con la ausencia de conectividad en ciertos instantes de tiempo.

El presente trabajo de diploma contiene la investigación y el proceso de desarrollo realizado para obtener una aplicación que pueda sincronizar la información gestionada por el módulo Resultados hacia una única computadora. El desarrollo se realiza bajo la guía de la metodología XP, la aplicación es implementada haciéndose uso del lenguaje de programación C++ y el framework de desarrollo Qt. Se presentan los requisitos identificados y el cumplimiento de los mismos en la aplicación final, que es validada además por las pruebas realizadas propuestas por la metodología de desarrollo utilizada.

El impacto social de los resultados de esta investigación radica en que se pudo crear un sistema de software que posibilita un mejor despliegue de la Colección Multisaber por las escuelas de la República Bolivariana de Venezuela.

Índice

Declaración de autoría I
Dedicatoria y agradecimientos III
Resumen VI
Índice VII
Introducción 1
Capítulo 1 Análisis de las soluciones existentes 5
1.1 Soluciones existentes 6
1.1.1 Entornos de computación distribuidos 6
1.1.2 Configuración de servidores esclavos y maestros en MySQL 7
1.1.3 Utilización de un middleware orientado a objetos 7
1.1.4 Utilización de Internet Communications Engine (Ice) 8
1.2 Selección de la metodología de desarrollo más adecuada 9
1.2.1 RUP 9
1.2.2 XP 10
1.2.3 ¿Por qué XP y no RUP? 11
1.3 Selección del framework a utilizar 13
1.3.1 GTK+ 13
1.3.2 Qt 14
1.3.3 ¿Por qué Qt? 15
1.4 Selección del lenguaje de programación 15
1.4.1 Java 16
1.4.2 Python 17
1.4.3 C++ 17
1.5 Selección del IDE de programación 18

1.5.1 Eclipse 18
1.5.2 Qt Creator 19
1.6 Selección de la herramienta CASE para el modelado 20
1.6.1 Visual Paradigm para UML 20
1.7 Estilos arquitectónicos 21
1.7.1 Modelo Vista Controlador 21
1.7.2 Arquitectura en capas 22
1.7.3 Arquitectura orientada a objetos para sistemas distribuidos 22
1.7.4 Una arquitectura combinada 23
1.8 Conclusiones 24
Capítulo 2 Presentación de la solución propuesta 25
2.1 Características del sistema 26
2.2 Requerimientos funcionales y no funcionales 27
2.2.1 Requerimientos funcionales comunes para ambas aplicaciones 29
2.2.2 Requerimientos funcionales de la aplicación del lado del cliente 30
2.2.3 Requerimientos funcionales de la aplicación del lado del servidor 30
2.2.4 Requerimientos no funcionales 31
2.3 Conclusiones 34
Capítulo 3 Exploración y planificación 35
3.1 Exploración 36
3.1.1 Historias de usuario 36
3.1.2 Historias de usuario relacionadas con ambas aplicaciones 38
3.1.3 Historias de usuario relacionadas con la aplicación del lado del cliente. 40
3.1.4 Historias de usuario relacionadas con la aplicación del lado del servidor 42
3.2 Planificación 45
3.2.1 Iteraciones 45
3.2.2 Plan de iteraciones 46

3.2.3 Plan de entregas 48
3.3 Conclusiones 48
Capítulo 4 Implementación y pruebas 50
Los módulos de Qt 51
La programación multihilos en Qt 51
4.1 Diseño 52
4.1.1 Diagrama de clases del sistema 52
4.1.2 Modelo de despliegue 55
4.1.3 Diagrama de paquetes 56
4.1.4 Diagramas de componentes 57
4.2 Implementación 63
4.2.1 1ª iteración 64
4.2.2 2ª iteración 68
4.2.3 3ª iteración 71
4.3 Pruebas 74
4.3.1 Pruebas de aceptación 75
4.4 Conclusiones 92
Conclusiones 93
Recomendaciones 94
Glosario 95
Referencias bibliográficas 96
Bibliografía 97

Introducción

Multisaber es un proyecto especializado en la creación de software educativo, pertenece a la Facultad 8 de la Universidad de las Ciencias Informáticas, se encuentra enfrascado en la creación de un grupo de multimedia, las cuales conforman la Colección Multisaber. La primera versión de la colección tiene la peculiaridad de haber sido creada con herramientas propietarias, tales como Macromedia Flash, Macromedia Director, Toolbook, entre otras aplicaciones privativas.

Durante los últimos años el intercambio comercial entre Cuba y la República Bolivariana de Venezuela se ha incrementado exponencialmente, este factor ha propiciado que las estrategias de desarrollo de los proyectos productivos del país se hayan enfocado en buscar comercio en el mercado venezolano.

La República Bolivariana de Venezuela adoptó el Decreto Ley No. 3.390, mediante el cual se dispone que la Administración Pública Nacional empleará prioritariamente software libre. El software para ser libre debe cumplir con las siguientes libertades orientadas por la Free Software Foundation, en adelante FSF:

"Con *software libre nos referimos a la libertad de los usuarios para ejecutar, copiar, distribuir, estudiar, cambiar y mejorar el software. Nos referimos especialmente a cuatro clases de libertad para los usuarios de software:*

- ***Libertad 0**: la libertad para ejecutar el programa sea cual sea nuestro propósito.*
- ***Libertad 1**: la libertad para estudiar el funcionamiento del programa y adaptarlo a tus necesidades —el acceso al código fuente es condición indispensable para esto.*
- ***Libertad 2**: la libertad para redistribuir copias y ayudar así a tu vecino.*
- ***Libertad 3**: la libertad para mejorar el programa y luego publicarlo para el bien de toda la comunidad —el acceso al código fuente es condición indispensable para esto. "* (Stallman, 2004 pág. 45)

El Proyecto Multisaber con el propósito de adentrase en el mercado venezolano tuvo que desarrollar una nueva colección que cumpliera con la condición de ser software libre. Para lograr dicho propósito surge una novedosa variante: crear las multimedia educativas utilizando el CMS Joomla para la gestión de los contenidos. A la nueva versión obtenida se le denominó Colección Multisaber versión multiplataforma, la

cual estaría enfocada totalmente sobre la web.

Después de tres años de trabajo continuado, el Gobierno de Venezuela anuncia la compra de un millón de pequeñas laptops (*classmates*) para ser distribuidas en las escuelas primarias, sobre todo en aquellas ubicadas en los lugares más pobres. El Ministerio del Poder Popular para la Educación venezolano se ha interesado en la implementación libre que se realiza a los 14 productos que integran la Colección Multisaber para que puedan ser utilizados con dichas laptops. Estas pequeñas computadoras tienen instalado como sistema operativo una distribución de GNU\Linux llamada Canaima y soportan conexiones inalámbrica y cableada.

Uno de los módulos que poseen los productos de la Colección Multisaber se denomina Resultados, mediante él los maestros pueden ver las acciones que han realizado los estudiantes en los productos de la colección, pues los mismos llevan un seguimiento de las trazas que se generan. Tras varios intercambios entre los especialistas cubanos y venezolanos surge la necesidad de mantener la posibilidad de que el maestro pueda revisar las trazas del módulo Resultados aunque los productos sean instalados de manera local en las computadoras. Para facilitar esto las *classmates* PC deberían mantener conectividad permanente con una máquina encargada de centralizar el registro de los estudiantes, para que el maestro no tenga que ir por cada una de las máquinas revisándolo, sin embargo, surge el inconveniente de que la conectividad en muchos lugares no es posible por lo que se propone buscar variantes para lograr el trabajo independiente de los alumnos, y en los momentos en que sea posible enviar la información de las trazas de cada una de las laptops para la PC del maestro.

Actualmente no existe ningún sistema encargado de gestionar el envío de la información recogida en los ordenadores de los estudiantes a partir del trabajo sin conectividad hacia el de los maestros, así como poseer múltiples opciones de configuración para realizar estas sincronizaciones de la manera que el maestro desee, teniendo en cuenta que los mismos no tienen que poseer muchos conocimientos de computación. Si no se lleva a cabo este sistema la Colección Multisaber perdería en usabilidad, los futuros clientes no estarían satisfechos del todo, lo cual afectaría el prestigio de la Universidad de las Ciencias Informáticas como centro de desarrollo de software y los maestros no podrían tener en un mismo lugar la información gestionada por el módulo Resultados, todo esto afecta la calidad del software, de ahí la importancia vital que tiene esta investigación.

Para darle cumplimiento a la necesidad de desplegar la versión multiplataforma de la Colección Multisaber por la República Bolivariana de Venezuela se presenta como **problema a resolver**: ¿Cómo centralizar la información gestionada a través del módulo Resultados desde las *classmates* hacia una única PC, permitiéndose el trabajo sin conectividad? Para brindarle una solución a este problema se define como **objeto de estudio** a las técnicas y herramientas utilizadas en aplicaciones que se basan en arquitecturas cliente servidor y permiten el trabajo sin conexión, dejando el **campo de acción** enmarcado en la sincronización de la información gestionada por el módulo Resultados de la Colección Multisaber.

En correspondencia a lo antes expuesto se propone como **objetivo general** de la investigación desarrollar una herramienta que centralice en una PC la información gestionada por el módulo Resultados de la Colección Multisaber desde las *classmates*, permitiendo el trabajo sin conexión. Para cumplir con este objetivo se plantean los siguientes **objetivos específicos**:

1. Definir los requerimientos de la aplicación.
2. Realizar el análisis y diseño de la aplicación necesaria para lograr la sincronización.
3. Implementar la aplicación.
4. Probar la aplicación.

Para cumplir los objetivos específicos se plantean las siguientes **tareas**:

1. Realizar un estudio del estado del arte de herramientas y técnicas que se utilicen para tareas similares.
2. Realizar un estudio del módulo Resultados de la Colección Multisaber.
3. Realizar entrevistas con integrantes del proyecto Multisaber que ayuden a definir los requerimientos.
4. Seleccionar y fundamentar la metodología de desarrollo adecuada.
5. Seleccionar el IDE y framework de desarrollo.
6. Seleccionar las herramientas y el lenguaje de programación a utilizar.
7. Definir y describir los requerimientos funcionales y no funcionales de la aplicación.
8. Definir los Casos de Uso/Historias de Usuario.
9. Detallar los Casos de Uso/Historias de Usuario.
10. Realizar el análisis y diseño del software.

11. Implementar los Casos de Uso/Historias de Usuario más importantes.
12. Realizar pruebas que permitan detectar posibles errores.

Como parte de la investigación se plantea como **idea a defender**: si se logra desarrollar una aplicación que permita centralizar, de manera segura, toda la información generada por la interacción de los estudiantes con los productos de la versión multiplataforma de la Colección Multisaber, hacia un único servidor, entonces se facilitaría a los maestros la revisión de las trazas de los estudiantes y se lograría añadir una herramienta a la Colección Multisaber que permita un despliegue basado en dicho modelo de una manera sencilla. Como **posibles resultados** se propone obtener una aplicación escalable que resuelva la necesidad de sincronización de la base de datos del módulo Resultados de la Colección Multisaber.

Capítulo 1
Análisis de las soluciones existentes

En el presente capítulo se realizará la fundamentación teórica de la investigación. Inicialmente se expondrá el estado del arte de las soluciones existentes, se analizarán las mismas, dando paso a la selección de la metodología de desarrollo, el lenguaje de programación, framework, IDE de desarrollo y herramientas a utilizar.

1.1 Soluciones existentes

A continuación se muestran algunas soluciones existentes que se basan en arquitecturas cliente servidor y están estrechamente relacionadas con el objeto de estudio de la investigación. Este análisis permitirá llegar a importantes conclusiones relacionadas con las características que deberá poseer el sistema a implementar.

1.1.1 Entornos de computación distribuidos

"Consiste en una colección finita ε de entidades computacionales comunicándose por medio de mensajes. Las entidades se comunican con otras entidades para lograr una meta común, para realizar una tarea, para computar la solución a un problema, satisfacer una solicitud ya sea por parte del usuario […] o de otras entidades" (Santoro, 2007 pág. 1)

Los entornos de computación distribuidos o sistemas distribuidos no son más que un conjunto de *entidades* que se comunican mediante la transmisión y recepción de mensajes. Son sistemas de software que se comunican entre sí mediante *mensajes*[1].

En los sistemas de computación distribuidos una entidad se puede comunicar directamente con un grupo de entidades, las cuales se pueden denominar *entidades vecinas*. Una entidad puede distinguir entre sus vecinas, e incluso podría conocerlas de antemano, por eso es común encontrar entidades que conocen cuáles son las entidades desde las cuales recibirán mensajes y a cuáles se le mandarán.

El problema de la actual investigación se asemeja mucho a un ambiente de computación distribuido porque una entidad pudiera ser un sistema en el ordenador del profesor que recibirá la información proveniente de otras entidades vecinas, las cuales le mandarán una serie de información a través de

[1] Los mensajes constituyen la unidad básica de comunicación que emplean los sistemas distribuidos para comunicarse, que en otras palabras, son una secuencia finita de bits.

mensajes.

1.1.2 Configuración de servidores esclavos y maestros en MySQL

Una opción viable para lograr enviar la información desde una base de datos de MySQL a otra situada en otro servidor que no siempre podría estar disponible, es a través del uso de los servidores esclavos y maestros. Este método consiste en convertir un servidor en maestro, el cual recibiría cada cierto tiempo, la información nueva de uno o varios servidores esclavos. Este sistema es una solución rápida para lograr la replicación de la información recogida por el módulo Resultados, pero tiene un gran inconveniente, la configuración de estos sistemas es sumamente complicada y requeriría de un especialista informático con avanzados conocimientos de MySQL para configurar todas las computadoras que harán la función de esclavos, así como el servidor maestro.

Esta solución no es la más factible porque en primer lugar limita a que deben haber especialistas informáticos para mantener estos sistemas. Pero es bueno conocer esta solución para el caso de un rápido despliegue de la Colección Multisaber en un pequeño entorno de pruebas reales.

1.1.3 Utilización de un middleware orientado a objetos

La palabra middleware se traduce como *software intermediario (en lo adelante se seguirá usando middleware para evitar ambigüedad en la traducción)* que se comporta como una capa intermedia entre las aplicaciones y el sistema operativo, o un sistema de administración de bases de datos o entre clientes y servidores. Los *middleware* orientados a objetos surgieron a mediados de los 90s, enriqueciendo el uso de los sistemas distribuidos, este paso de avance significó que los desarrolladores no tenían que ser grandes conocedores de detalles muy técnicos sobre las redes para poder desarrollar aplicaciones distribuidas. Entre los más conocidos se encuentran DCOM y CORBA.

DCOM fue el resultado de una implementación de Microsoft, su principal deficiencia radicaba en que no funcionaba en sistemas heterogéneos donde estuvieran computadoras con distintos sistemas operativos, tampoco funcionaba correctamente en sistemas distribuidos donde existieran cientos o miles de ordenadores operando, mientras que CORBA presentaba problemas con la compatibilidad absoluta con lenguajes como C o C++.

La clave de los *middleware* orientados a objetos es que permiten la implementación de funciones que pueden ser llamadas por objetos desde otras computadoras, permitiendo que el despliegue de sistemas distribuidos sea una tarea más sencilla.

1.1.4 Utilización de Internet Communications Engine (Ice)

Ice es un middleware orientado a objetos de avanzadas funcionalidades, fue creado por la compañía ZeroC. Su plataforma soporta la implementación de sistemas distribuidos con una gama de lenguajes de programación, tales como: C++, Java, C#, Visual Basic, Objective-C, Python y Ruby.

Permite además el despliegue de sistemas distribuidos en ambientes heterogéneos, lo mismo se puede crear un sistema que funcione como servidor, el cual puede correr sobre Windows, GNU\Linux, Mac, Unix, etc., y sus clientes, de igual manera, pueden estar corriendo sobre cualquier sistema operativo. En GNU\Linux es posible programar aplicaciones utilizando sus librerías, las cuales están disponibles en los repositorios de varias distribuciones.

"Es contra las insuficientes opciones existentes que ZeroC, Inc. decidió desarrollar el Internet Communications Engine, o Ice abreviando. Las principales metas del diseño de Ice son:

- *Ofrecer una plataforma middleware orientada a objetos adecuada para su uso en entornos heterogéneos.*
- *Proporcionar un conjunto completo de funcionalidades que apoyen el desarrollo de aplicaciones distribuidas reales para una amplia variedad de dominios.*
- *Evitar complejidad innecesaria, haciendo la plataforma fácil de aprender y usar.*
- *Proporcionar una implementación que sea eficiente en el ancho de banda, uso de memoria y en la carga del CPU.*
- *Ofrecer una implementación que tenga seguridad incorporada, haciéndola adecuada para usar sobre redes públicas inseguras."* (Henning, y otros, 2009 pág. 4)

Pero Ice va más allá, también es posible implementar un servidor en un lenguaje de programación y los clientes tener otro lenguaje de programación. El código fuente de estas aplicaciones es funcional en cualquier entorno de despliegue.

1.2 Selección de la metodología de desarrollo más adecuada

Uno de los temas más polémicos y complicados con los que tienen que lidiar los equipos de desarrollo es el referente a la selección de la metodología de desarrollo. La ingeniería de software es una ciencia de relativamente corta edad, por eso en la actualidad no existe una metodología estándar que sea considerada como "*exacta*", más bien han surgido decenas de éstas en los últimos años.

Dentro de las metodologías de desarrollo de software existen varias familias, existen las llamadas metodologías *tradicionales*, entre las que se encuentra RUP como su principal exponente, otra familia nacida en los últimos años son las conocidas como metodologías *ágiles*, entre las que sobresale XP.

1.2.1 RUP

El Proceso Unificado del Rational o Proceso Unificado de Desarrollo es conocido internacionalmente por las siglas RUP y es una de las metodologías de desarrollo más utilizadas, *"...los verdaderos aspectos definitorios del Proceso Unificado se resumen en tres frases clave—dirigido por casos de uso, centrado en la arquitectura, e iterativo e incremental."* (Jacobson, y otros, 2000 pág. 4)

En RUP los casos de uso (en adelante CU) no sólo especifican los requerimientos del sistema, sino que guían a los diseñadores y desarrolladores a confeccionar posteriormente los modelos de diseño e implementación, además de servir como guía para que los ingenieros de prueba puedan confeccionar los casos de prueba.

La arquitectura es el eje central para el desarrollo de un software, todo gira en torno a ella, RUP plantea que un software tiene una función y una forma, la función la describen los CU y la forma es la arquitectura, *"...los arquitectos modelan el sistema para darle una forma. Es esta forma, la arquitectura."* (Jacobson, y otros, 2000 pág. 6). Por eso la arquitectura debe permitir que los CU encajen en la misma, así como la arquitectura permitir que se implementen todos los CU.

Se dice que RUP es iterativo e incremental porque el proceso se puede dividir en varias iteraciones, obteniéndose de cada iteración un miniproyecto, y los incrementos del software se refieren al crecimiento del producto.

Otra característica distintiva de RUP es que utiliza los diagramas del *Lenguaje Unificado de Modelado*

(*Unified Modeling Language*, UML) para representar los esquemas de software.

RUP es muy común en proyectos de grandes dimensiones con gran cantidad de trabajadores y con planes de desarrollo de larga duración, iteraciones largas. Durante todo el ciclo se van obteniendo un gran número de artefactos que conforman al producto.

1.2.2 XP

Programación Extrema (*Xtreme Programming*, XP) es una de las metodologías ágiles más conocidas. Por eso antes de hablar de XP hay que remitirse a las características de las metodologías ágiles: se basan en satisfacer al cliente mediante tempranas y continuas entregas de software, que funcione, desde un par de semanas a un par de meses, con el menor intervalo de tiempo posible entre una entrega y la siguiente, orientadas a pequeños proyectos, de corta duración, equipos pequeños.

XP defiende algunas prácticas, tales como:

1. Programación en parejas: Dos personas frente una misma pantalla.

"Algunas *conclusiones hasta ahora respecto a la programación por parejas:*

- *La programación por parejas realmente mejora la calidad del código.*

[...]

- *La programación por parejas es agotadora y no debería hacerse durante todo el día.*
- *Es bueno cambiar de parejas frecuentemente.*
- *La programación por parejas realmente mejora la distribución de conocimiento entre el equipo.*

[...]

- *La revisión de código es una alternativa aceptable a la programación por parejas.*

[…]

- *No fuerces a la gente a hacer programación por parejas. Anima a las personas y proporcionales las herramientas adecuadas, pero permíteles experimentar con ella a su propio ritmo.*" (Kniberg, 2007 págs. 81-82).

2. Desarrollo guiado por pruebas (del inglés *Test Driven Development*, TDD): Se basa en escribir una prueba, a continuación, se escribe el código suficiente para pasarla y después se refactoriza el

código, principalmente para mejorar la legibilidad y eliminar duplicaciones. XP propone que antes de implementar una nueva funcionalidad se deben escribir la o las pruebas que debe pasar dicho código para considerar que esa funcionalidad está implementada, inicialmente el código debe fallar la prueba, luego se escribe nadamás el mínimo código necesario para darle cumplimiento a la tarea, después se pasa a la refactorización, proceso mediante el cual se eliminan duplicaciones y se mejora la legibilidad del código fuente.

3. Diseño incremental: Significa tener un diseño simple desde el principio y mejorarlo continuamente. Es un efecto secundario de aplicar TDD. Los proyectos son propensos a los cambios, y muchas veces, un cambio significa cambiar el diseño, en el mundo real casi nunca se puede tener un diseño completo al inicio y que ese mismo se adapte a todos los cambios.
4. Integración continua.
5. Estandarización de código: Se debe tener un estándar de código por el cual se regirá todo el equipo de desarrollo. Esto mejora muchísimo el entendimiento del código.
6. Ritmo sostenible / trabajo enérgico: Las jornadas de trabajo no se pueden sobrecargar con horas extras, las jornadas maratónicas tienden a que baje la calidad del trabajo.
7. Propiedad colectiva del código: La propiedad colectiva del código es un efecto secundario de aplicar la programación en parejas con rotación frecuente, gracias a esto los equipos logran tener un alto nivel de propiedad colectiva, lo que propicia una alta robustez en los equipos de desarrollo. Así no se afectan tanto cuando falta uno de los programadores, debido a que todos tienen una visión global de código fuente.

Para aplicar XP hay que saber escuchar para saber qué quiere el cliente y satisfacerlo, hay que diseñar para poder crear un software escalable que se adapte al cambio, los diseños son mejores mientras más sencillos son, también hay que codificar porque sin la codificación(*"implementación"*) no se obtiene el software y por último hay que saber probar porque si no se prueba el software no se sabe si la codificación ha terminado y si el software cumple con la calidad esperada y con las expectativas de los clientes.

1.2.3 ¿Por qué XP y no RUP?

El punto más discordante de por qué utilizar XP y no RUP radica en que la interacción con los verdaderos

clientes va a ser casi nula debido a que los mismos se encuentran en Venezuela, sin embargo, XP propone que para su éxito debe existir una *alta comunicación* con los clientes, los mismos priorizarán cuáles son las Historias de Usuario (en lo adelante HU) que se deben incorporar tempranamente a las iteraciones, si es posible, el cliente debe estar muy cerca del equipo de desarrollo. Para esta investigación, los requerimientos de la aplicación se definirán a través de entrevistas con los líderes del Proyecto Multisaber, quienes, por suerte, están muy cerca del equipo de desarrollo de la aplicación. Ellos serán además quienes prioricen las tareas y las HU más importantes, ellos podrán probar la aplicación continuamente –gracias a la integración continua y las iteraciones cortas–, en otras palabras, cumplirán casi todas las funciones de los clientes.

Por lo antes expuesto se puede arribar a la conclusión que la metodología de desarrollo que más se acerca a las necesidades de la presente investigación es XP. Las *integraciones continuas*, las *entregas tempranas*, el *desarrollo guiado por pruebas*, la *estandarización del código*, las *iteraciones cortas* son los puntos de XP que más satisfacen a las necesidades de la investigación. En este caso el equipo de desarrollo está integrado sólo por una persona y se cuenta con muy poco tiempo para la entrega final del proyecto, cuando RUP es más aplicable en equipos de desarrollo grandes, iteraciones largas, también se documenta con gran cantidad de artefactos, a veces abrumadora, algo que en XP no es así. El hecho de que el equipo de desarrollo está integrado por una sola persona limita mucho el tiempo para la programación de la aplicación, ya que esa misma persona es la que tiene que documentar todo, construir los diagramas necesarios para representar la aplicación y a la vez programarlo todo.

RUP utiliza UML para modelar los esquemas representativos, algo que XP también posee, lo que es más flexible, no hay que realizar todos los diagramas de UML que propone RUP, sino los necesarios que el equipo considere. Los equipos que han aplicado RUP, a menudo se ven abrumados por la cantidad de diagramas que generan, que muchas veces, casi ni los utilizan, lo cual no significa que no sean importantes, mientras más representaciones se tengan, mayor entendimiento del sistema se tendrá por el equipo de desarrollo. Si se tiene en cuenta que el equipo de desarrollo está conformado por una sola persona para realizar todo el proceso, XP es lo más adecuado.

RUP y XP tienen puntos de convergencia, como los casos de uso y las historias de usuario. Los primeros se especifican en RUP en un documento para ofrecer un entendimiento global de las funcionalidades del

mismo, por otro lado, las HU de XP se especifican también con el mismo propósito. Otro punto en común es que ambas metodologías se centran en la arquitectura y son iterativas. La gran ventaja que le ofrece XP al actual equipo de desarrollo son las entregas al final de cada iteración, propiciando un espacio para que los líderes de Multisaber prueben la aplicación.

Las integraciones continuas van a permitir descubrir errores tempranamente e incorporar sus correcciones como tareas en las iteraciones siguientes. Las iteraciones de corta duración permitirán obtener software funcional tempranamente. Además, aplicar TDD permitirá incorporar las pruebas desde un inicio, así se evitan sorpresas indeseadas –errores difíciles de corregir– al final de las iteraciones.

1.3 Selección del framework a utilizar

Para GNU\Linux existen varios framework que cumplen con la condición de ser software libre, debido a que los mismos son liberados bajo la licencia GPL del proyecto GNU. Dentro de estos framework los que más sobresalen son GTK+ y Qt.

1.3.1 GTK+

GTK+ es un conjunto de bibliotecas multiplataforma que ofrece un conjunto de herramientas para crear interfaces de usuario. Está programada en C y ofrece soporte para programar en muchos lenguajes de programación, tales como C++, Python, C#, Java, Ruby, PHP, entre otros. GTK+ está licenciado bajo la licencia GNU GPL versión 2.1, que permite desarrollar software propietario y software libre. GTK significa *Gimp Toolkit*, fue creada por Spencer Kimball y Peter Mattis[2] para crear componentes visuales para el popular programa de manipulación de imágenes Gimp en 1997. Pero con el paso de los años se fue expandiendo gracias a la comunidad de programadores alrededor del mundo que la utilizan. Gracias a las librerías de GTK+ es que existen populares entornos de escritorio para plataformas Unix, como GNOME y XFCE, y un sinnúmero de aplicaciones muy usadas, tales como: el editor de imágenes, Gimp; el popular programa de mensajería instantánea, Pidgin; el cliente de correo, Evolution; el procesador de texto, AviWord; el editor de gráficos vectoriales, Inkscape; entre otras aplicaciones.

[2] En 1997 Spencer Kimball y Peter Mattis eran miembros del eXperimental Computing Facility (XCF) de la Universidad de California.

1.3.2 Qt

Qt es un framework multiplataforma usado fundamentalmente en programas que requieren interfaz gráfica de usuarios. Su origen se remonta a 1991, fue creado por la empresa Trolltech. Está programado en C++ y ofrece soporte para una variedad de lenguajes de programación, tales como: C++; Python, gracias a PyQt, PySide y PythonQt; Ruby, gracias a QtRuby; PHP, gracias a PHP-Qt; Java, gracias a QtJambi y D, gracias a Dqt; entre otros lenguajes de programación. En la actualidad Qt puede ser utilizado lo mismo en aplicaciones web como en aplicaciones de escritorio, así como en dispositivos móviles.

Qt es propiedad de Nokia, empresa que lo utiliza fundamentalmente en sus plataformas móviles. Sin embargo, a pesar de ser propiedad de Nokia se sigue distribuyendo como software libre, utiliza tres licencias: una comercial, llamada Qt Commercial Developer License, la cual es para los usuarios que no quieran compartir su código fuente, esta licencia hay que pagarla; GNU LGPL versión 2.1, que es gratis y se puede utilizar para desarrollar aplicaciones propietarias y aplicaciones libres; GNU GPL versión 3.0, su principal diferencia respecto a la LGP versión 2.1 es que estrictamente no permite crear aplicaciones propietarias.

Para la actual investigación, se puede utilizar Qt, porque la licencia LGPL versión 2.1 permite lo mismo distribuir las aplicaciones como propietarias o como software libre, de igual manera se puede utilizar la licencia GNU GPL versión 3 porque lo que se desea es una aplicación que cumpla con los requisitos de software libre.

Lo mejor de Qt es que no es solamente una simple librería para crear aplicaciones con interfaz gráfica, sino que es todo un framework que permite desarrollar todo tipo de aplicaciones, sobresalen entre otros su framework para aplicaciones multimedia, llamado Phonon; su módulo para aplicaciones 3D con OpenGL; su módulo WebKit; su módulo SVG para manipular imágenes vectoriales; su módulo para interactuar con archivos XML; el soporte para la programación multihilos; el soporte para la comunicación entre procesos en tiempo de ejecución; el soporte para la comunicación por sockets, muy útil en sistemas distribuidos; entre otras funcionalidades.

Entre las aplicaciones más importantes que utilizan Qt se pueden mencionar: *Google Earth*, *Opera*, *Skype*, *VLC Media Player*, *Virtualbox*, *Psi*, *Doxygen*, *MythTV* y el entorno de escritorio *KDE*.

Qt tiene a su favor la gran comunidad de usuarios y su abundante documentación, junto con el SDK de Qt

Nokia también distribuye un novedoso IDE de programación llamado QtCreator, el cual será abordado próximamente.

1.3.3 ¿Por qué Qt?

Para implementar la solución de esta investigación se propone la utilización de las librerías Qt. Esto se debe a varios factores. El primero es la facilidad para programar, con Qt en sólo unas pocas líneas de código te ahorras muchísimas que utilizarías usando por ejemplo GTK, realmente es muy rápido crear aplicaciones con estas librerías, como parte de la investigación realizada se pudo constatar lo fácil que es implementar en Qt una aplicación que se comunique con otra por la red a través de los sockets, algo que el sistema a implementar podría necesitar. La otra gran ventaja es la abundante documentación que se puede encontrar, así como la abundante comunidad de desarrolladores que existen. Para poner un ejemplo, en la Universidad de las Ciencias Informáticas existe una gran comunidad de desarrollo de Qt, sobresalen: el compilador libre para Action Script, Picassus; el editor de videos, QEVEN; el diccionario bilingüe español-inglés, Shakespeare y la aplicación para medir el consumo de Internet, Kuenta. GTK+ es mucho más complicado para programar, además, sus componentes visuales ofrecen mucho menos funcionalidades que los de Qt, el proceso de implementación con GTK+ se torna un poco engorroso y cuesta trabajo adaptarse a la sintaxis que GTK necesita. Qt además ofrece una alta portabilidad, la aplicación final podría adaptarse muy fácilmente para que funcione no sólo en Linux, sino en otras plataformas como Windows o Mac. Debido a todo lo planteado anteriormente es que se ha decidido utilizar Qt en vez de las librerías GTK+.

1.4 Selección del lenguaje de programación

Seleccionar el lenguaje de programación es uno de los aspectos de más peso en los proyectos, debido a que se deben tener una gran cantidad de factores antes de elegir, por ejemplo: grado de familiaridad con el lenguaje del equipo de desarrollo, IDE que lo soporta, eficiencia y consumo de memoria, soporte del lenguaje para poder programar la aplicación deseada, y podría seguir con una innumerable lista de factores a tener en cuenta.

Existen dos familias de lenguajes de programación, los lenguajes de bajo nivel y los lenguajes de alto

nivel. Los lenguajes de bajo nivel poseen una sintaxis muy cercana al Lenguaje de Máquina[3], el cual es muy difícil de interpretar por los seres humanos, mientras que los lenguajes de alto nivel ofrecen una sintaxis muy cercana al lenguaje de los seres humanos.
Entre los lenguajes de programación de alto nivel más populares se encuentran el C, C++, Java, C#, Python, D, entre otros.

1.4.1 Java

El lenguaje Java surgió en 1995, fue creado por Sun Microsystems. Java es un lenguaje que permite la programación orientado a objetos y marcó el inicio de una nueva generación de lenguajes de programación. Ofrece una sintaxis muy elegante y se puede decir que heredó muchas cosas de C.
Cuando salió a la luz, tenía de todo para remplazar a otros lenguajes como C y C++ pero eso no ocurrió por una serie de factores.
El primer error que cometió Sun Microsystems fue haber bloqueado el acceso al código fuente de Java, y sólo le permitió a unos pocos mejorarlo, tampoco ofreció libremente compiladores mediante el cual los programadores pudieran experimentar. En el sitio web de Sun Microsystems sólo se pueden descargar el soporte de Java para 4 arquitecturas de hardware: x86, PowerPC, AMD64 y Sparc, es por eso que muchos se cuestionan para qué adoptar un lenguaje que no ofrece alta compatibilidad con las arquitecturas de hardware existentes.
"Durante sus primeros cinco años, cuando todo el mundo estaba considerando seriamente la posibilidad de utilizarlo, Java corría diez veces más lento que C, porque era interpretado, en lugar de compilado, [...]. Esto se habría arreglado mucho más rápido si Sun hubiera participado y alentado a la comunidad de desarrollo de un compilador libre existente en aquel momento." (Curtis, 2009 pág. 155).
En estos momentos Java depende mucho de su máquina virtual, su máquina virtual ofrece la portabilidad entre plataformas, un buen punto a su favor, pero por otro lado, las aplicaciones en Java son muy lentas y el consumo de memoria es alto, a pesar de que Java posee un sistema de recogida de residuos, conocido internacionalmente como *garbage collector*[4], en lo adelante GC, dicho sistema ofrece una mejoría en el

[3] Lenguaje formado por 0 y 1 (Sistema Numérico Binario) que es el que las computadoras interpretan.
[4] Sistema que busca los objetos innecesarios cargados en memoria y los elimina.

consumo de memoria.

En la actualidad para correr una aplicación de Java se debe instalar el entorno en tiempo de ejecución de Java (del inglés Java Runtime Environment, JRE), el cual contiene a su máquina virtual y un conjunto de utilidades necesarias. En el año 2007 Sun liberó completamente la biblioteca de clases de Java bajo la licencia GNU GPL, surgiendo así el OpenJDK, la implementación libre de la plataforma de desarrollo Java.

La principal limitante para desarrollar en Java es que se obligaría a los clientes a instalar su máquina virtual, esto representa un consumo adicional de memoria para esos pequeños dispositivos donde se planea desplegar el software.

1.4.2 Python

Python es uno de los lenguajes más populares para el desarrollo de aplicaciones de escritorio para GNU\Linux, aunque también es usado para la programación web por importantes empresas como Google. Su principal arma es que es mejorado por una gran comunidad alrededor del mundo, quienes le han añadido soporte para hacer de todo, desde la computación del entretenimiento hasta la computación científica. La principal desventaja de Python es el rendimiento de las aplicaciones, Python es lento en tiempo de ejecución porque es interpretado en vez de compilado. Es por eso que muchos programadores siempre se han cuestionado la viabilidad de programar en Python cuando se esfuerzan en construir aplicaciones de alto rendimiento. Entre los principales IDEs con soporte para Python se pueden mencionar el Eclipse y Geany, entre otros, –que realmente son muy pocos si se compara con otros lenguajes.

1.4.3 C++

C++ es un lenguaje de programación diseñado a mediados de los años 80 por Bjarne Stroustrup[5]. En varias bibliografías se puede leer que C++ es un lenguaje de programación multiparadigmas porque soporta la programación orientada a objetos (en adelante POO) y la programación estructurada.

C++ es uno de los lenguajes más potentes de la actualidad, realmente la principal ventaja de C++ es su

[5] Bjarne Stroustrup es un científico de la computación y Catedrático de ciencias de la computación en la Universidad A&M de Texas.

eficiencia, debido a que es un lenguaje compilado, por estos motivos es que C++ es el lenguaje de programación seleccionado para implementar la solución, pues las *classmates* PC son computadoras de bajas prestaciones de hardware, lo cual impedimenta usar un lenguaje que no haga el mejor uso de los recursos.

Para programar en C++ en GNU\Linux el compilador más utilizado es el g++. Este compilador pertenece a la familia de compiladores del proyecto GNU llamada GCC, del inglés *GNU Compiler Collection,* dicha colección está compuesta enteramente por software libre y hoy por hoy se considera un estándar para las plataformas basadas en Unix.

Sin la creación del GCC del Proyecto GNU en la actualidad no se pudiera contar con las distribuciones de GNU\Linux que hoy están disponibles ni miles de aplicaciones de software libre.

Para programar en C++ sobre GNU\Linux existen varios IDEs de renombre, entre ellos el Eclipse y el QtCreator.

1.5 Selección del IDE de programación

Para seleccionar el IDE con el que se llevará a cabo la implementación se tuvieron en cuenta los siguientes criterios:

- Que sean compatibles con GNU\Linux.
- Que tengan soporte para C++ y hagan uso del compilador g++ del Proyecto GNU.
- Soporte para Subversion.
- Que tengan un buen completamiento de código.
- Que se integren con las librerías Qt.
- Que cuenten con abundante documentación.
- Que su consumo de memoria no sea excesivo.

Si se parte de los criterios anteriores, la selección se reduce a dos IDEs: el Eclipse y el QtCreator.

1.5.1 Eclipse

El Eclipse es un IDE multiplataforma con soporte para varios lenguajes de programación, aunque

originalmente es para Java, el soporte para otros lenguajes como C++ y PHP se ha logrado gracias a que el Eclipse presenta una arquitectura que permite la incorporación de plugins. Actualmente se puede descargar directamente una versión preparada con soporte para C++, dicho soporte llegó gracias a la incorporación del plugin CDT, del inglés *C and C++ Development Tools.* Para un correcto funcionamiento sobre GNU\Linux se debe instalar el compilador g++ y para debuguear el GDB, del inglés *GNU Debugger*[6], el cual es el depurador del Proyecto GNU.

El Eclipse también brinda soporte para Subversion, gracias a varios plugins, los más usados son el Subclipse y el Subversive.

Quizás el mayor defecto que se le puede señalar al Eclipse es su alto consumo de memoria, porque para que corra depende de la máquina virtual de Java. El Eclipse es probablemente el IDE más utilizado por usuarios de GNU\Linux debido a su versatilidad como IDE.

1.5.2 Qt Creator

Qt Creator al igual que el Eclipse es un IDE multiplataforma. Es distribuido por Nokia como parte del Qt SDK, que es un grupo de herramientas para desarrollar utilizando el framework Qt, el SDK de Qt es multiplataforma y a la vez es software libre, actualmente soporta como lenguaje el C++. Entre las herramientas que conforman el Qt SDK, aparte del Qt Creator, se encuentran Qt Assistant, Qt Designer y Qt Linguist. Todas estas herramientas se integran dentro del QtCreator para convertirlo en un completo IDE.

- Qt Assistant es una herramienta que permite navegar fácilmente por la documentación de Qt, realmente es una herramienta sorprendente que constituye, sin dudas, una de las mejores ayudas que se pueda encontrar acerca de Qt. Los usuarios noveles sin duda alguna deberían comenzar a estudiar por Qt Assistant.
- Qt Designer es la herramienta que permite crear interfaces gráficas de usuarios, muy intuitiva y fácil de usar, y el hecho de que se integre dentro del IDE hace que sea más fácil de usar el Qt Creator.

[6] El GDB es el depurador creado por el proyecto GNU en el año 1988, su programador fue el padre del software libre Richard Stallman.

- Qt Linguist hace fácil y rápida la traducción e internacionalización de las aplicaciones que usen Qt, su función es permitir crear aplicaciones con soporte para varios idiomas.

La versión actual del Qt Creator es la 1.3, la misma tiene un rápido completamiento de código, su consumo de memoria es moderado, también corrige la sintaxis en tiempo real y se integra automáticamente con varios sistemas de control de versiones, tales como CVS, Perforce, Git y Subversion.

Todo el SDK de Qt, incluyendo el Qt Creator se distribuye bajo 3 licencias, entre las cuales se encuentran la GNU GPL versión 3.0 y la GNU LGPL versión 2.1, ambas licencias permiten el desarrollo de software libre, pero una de ellas, la GNU GPL versión 3.0 sólo permite que el software sea libre y nunca propietario. Para esta investigación cualquiera de las dos se podría utilizar, lo más importante es que se desarrollará sobre una plataforma que cumple con la condición de ser software libre.

1.6 Selección de la herramienta CASE para el modelado

Las herramientas CASE influyen directamente en el aumento de la velocidad de los equipos de desarrollo, ahorrando horas de trabajo. La más ideal debe cumplir con varios requisitos: el primero es que debe permitir modelar partes del software a través de los diagramas de UML, el segundo es que permite la generación de código en el lenguaje en el que se va a programar, en este caso C++. A estos requisitos se les pueden añadir otros como el consumo moderado de memoria, generación de sentencias SQL, integración con los IDEs, entre otros.

1.6.1 Visual Paradigm para UML

Se ha decidido utilizar Visual Paradigm para UML como herramienta CASE, esta selección se basa en las funcionalidades que ofrece, la aplicación de las mismas en los proyectos de desarrollo de software permiten un ahorro considerable de tiempo e influyen directamente en la calidad del proceso de desarrollo, entre ellas:

- Soporte para UML versión 2.1: Tal y como se explicó anteriormente, la metodología XP expresa en que se deben modelar los esquemas representativos del software a través de los diagramas de UML. La versión 2.1 de UML es la más utilizada a nivel internacional.

- Generación de código para varios lenguajes de programación, entre ellos se encuentra el C++.
- Generación de bases de datos: Visual Paradigm permite la generación automática de bases de dato a partir de un Modelo entidad-relación.
- Interoperabilidad entre diagramas: Visual Paradigm permite a partir de un diagrama obtener otro que guarde relación con el mismo.

Sin duda alguna, el uso del Visual Paradigm agilizará el proceso de desarrollo.

1.7 Estilos arquitectónicos

Antes de definir la arquitectura de una aplicación lo primero que se debe hacer es definir la infraestructura con la que se cuenta para desplegar el software, para ello se tienen en cuenta las necesidades de hardware, cuántas computadoras, servidores, desde qué tipo de dispositivos se accederá, etc. Luego se deben seleccionar y combinar los estilos y patrones arquitectónicos que se emplearán, para terminar seleccionando los subsistemas e integrarlos en la arquitectura.

A continuación los estilos arquitectónicos que más se acercan a las necesidades de la investigación.

1.7.1 Modelo Vista Controlador

El Modelo Vista Controlador (MVC) se caracteriza por separar el componente que manipula los datos de la aplicación, la interfaz gráfica de usuario y la lógica de control de la aplicación en tres componentes. El modelo es la parte del sistema que administra el comportamiento y manipula los datos persistentes, la vista se refiere a la interfaz de usuario y el controlador es la parte encargada de recibir los eventos provenientes de la vista, es quien manipula estos eventos, cada vista posee su controlador, cuando se recibe la notificación de una acción desde la vista, el controlador puede responder directamente a la vista o comunicarse con el modelo para mandar a realizar una acción, que puede incluir hacer alguna modificación. El modelo no tiene que conocer a las vistas, sin embargo, utilizando el patrón de diseño Observer[7] el modelo puede avisar a la vista interesada del cambio de estado en alguna información manejada.

[7] Es un patrón de diseño que permite que varios objetos observadores se suscriban en un objeto denominado sujeto, y cuando un evento ocurra el sujeto lo notificará a todos sus observadores.

En ocasiones se le define como un patrón de diseño o práctica recurrente.
El MVC permite que una misma aplicación tenga diferentes vistas, además, permite que cada componente se especialice en su función, ofrece claridad en el diseño y facilita el mantenimiento. El patrón MVC tiene su principal aplicación en las aplicaciones web y ha tomado auge en aplicaciones de escritorio que manejan recursos multimedia.

1.7.2 Arquitectura en capas

Las arquitecturas en capas son una de las más usadas en todo tipo de aplicación. Consiste en separar la aplicación en una jerarquía de capas, donde una capa sólo se comunica con su capa inmediata superior e inferior. Una de las formas en que más aparece este tipo de arquitectura es dividida en 3 capas:

- Presentación: es con la que el usuario interactúa, la que le presenta la información y a la vez valida los datos entrados por el usuario, esta capa sólo se comunica con la capa inmediatamente inferior, la capa de negocio.
- Negocio: es donde se maneja la lógica del funcionamiento de la aplicación, es quien recibe la información de la capa presentación y actúa de acuerdo a lo pedido, es donde se define todo el comportamiento de la aplicación, se comunica con la capa de datos para solicitar alguna información, finalmente, puede comunicarse con la capa presentación para brindarle datos de interés para el usuario.
- Datos: es la capa encargada de acceder a los datos persistentes y a la vez de almacenarlos, puede incluir varios gestores de base de datos. Se comunica con la capa de negocio para recibir las peticiones de la misma y en consecuencia modifica los datos que maneja o devuelve la información solicitada.

La jerarquía en capas permite un fácil mantenimiento del código fuente, permite que un cambio en una capa no necesariamente incluya cambios en las otras capas.

1.7.3 Arquitectura orientada a objetos para sistemas distribuidos

La arquitectura orientada a objetos para sistemas distribuidos permite que un objeto local se comunique con un objeto remoto en un lugar distante, es decir, en otra computadora. Este estilo arquitectónico está

vinculado con los conceptos de la computación distribuida y como es obvio a la programación orientada a objetos.

Para que los programas se comuniquen a través de la red es muy común que se haga uso de un middleware que exponga una interfaz remota, este middleware haría la función de un servidor y mediante él se puede brindar acceso a bases de datos u otro servicio que se pueda necesitar. Los objetos remotos y los objetos locales se comunicarán mediante el envío de mensajes. También se puede obviar el uso de un middleware y se puede implementar un servicio que simule ser un servidor y se comunicará con objetos de otras entidades mediante el envío y recepción de mensajes.

1.7.4 Una arquitectura combinada

Por un lado existe la necesidad de que se comuniquen entidades situadas en distintas computadoras mediante la transmisión de mensajes, este mecanismo es un ejemplo de entorno de computación distribuida, que ya se explicó, entonces, se puede decir que se está en presencia de una arquitectura orientada a objetos para sistemas distribuidos. El otro punto de vista es que es necesario que las entidades accedan a los datos almacenados en la base de datos de Multisaber, que estará situada en la misma computadora donde se instalará la aplicación, en este caso, debe de existir una interfaz visual con la que los clientes interactúen, también un mecanismo que medie entre la interfaz visual y los datos, entonces el problema se puede dividir en tres capas, por lo que se pudiera utilizar el estilo MVC o la arquitectura en tres capas.

Se ha decidido utilizar el patrón arquitectónico tres capas por la claridad que ofrece este último, simplifica mucho la comprensión del código fuente, reduce la dependencia entre clases y permite una mejor evolución del sistema ya que al modificar alguna de sus capas los cambios no afectan las otras capas, además, es muy fácil de mantener.

La arquitectura que permitirá brindar una solución a la presente investigación está constituida por la convergencia de dos tipos de arquitecturas, que en esta ocasión se combinan entre sí: la arquitectura en capas, específicamente el patrón arquitectura en tres capas y la arquitectura orientada a objetos para sistemas distribuidos.

1.8 Conclusiones

En este capítulo se ha analizado el estado del arte de las herramientas existentes que se relacionan con el campo de acción de la investigación, una de las conclusiones más importantes a las que se puede arribar es que el problema al que se le está buscando solución se asemeja mucho a un entorno de computación distribuido, donde varias entidades se comunican a través del envío y recepción de mensajes.

También se aprecia la existencia de los middlewares para permitir el desarrollo de aplicaciones distribuidas sin tener que ser un gurú en los conocimientos acerca de redes, de ellos, el que más puede ayudar es el Ice.

Otra conclusión relevante a la que se puede llegar es acerca de la necesidad inobjetable de utilizar una metodología de desarrollo ágil, en este caso, XP, pues está probada su eficiencia en equipos de desarrollo pequeños. Las entregas frecuentes, las iteraciones cortas y aplicar TDD son premisas que ayudarán a entregar software funcional y probado tempranamente.

También se puede concluir que el ambiente de desarrollo integrado por el SDK de Qt, el Visual Paradigm y el lenguaje de programación C++, y la integración del Qt Creator con un sistema de control de versiones es lo más conveniente. El análisis exhaustivo de los estilos arquitectónicos existentes ha permitido deducir que el futuro sistema será soportado por la combinación de dos estilos arquitectónicos. Sin duda alguna este capítulo ha constituido un buen punto de partida para el desarrollo final.

Capítulo 2
Presentación de la solución propuesta

En el presente capítulo se presentará la forma que deberá poseer el sistema. Este análisis permitirá arribar a las principales características que debe poseer el sistema a implementar, logrando alcanzar la identificación de los *requerimientos funcionales y no funcionales*[8] con los que debe cumplir la solución propuesta para dar cumplimiento a la situación problémica de la presente investigación.

2.1 Características del sistema

El sistema propuesto estará compuesto de dos aplicaciones independientes que se comunicarán entre sí mediante el envío y la recepción de mensajes. Una de ellas hará la función de servidor, estaría instalada en la computadora utilizada por el profesor y la otra aplicación cumplirá el rol de cliente, que es la que utilizarán los alumnos para poder interactuar con el servidor.

En las computadoras de los estudiantes, así como en la del profesor estará instalado el servidor de base de datos MySQL Server. La aplicación que se debe implementar para funcionar como cliente se conectará con su base de datos local en su servidor MySQL, buscará las trazas y las enviará a través de un puerto, que se debe definir, hacia la aplicación que estará recepcionando esta información desde la PC del profesor, ésta sería la aplicación servidora, que debe atender cada petición por un hilo independiente, esto significa atender varias peticiones en paralelo o concurrentemente, tal y como funcionan los servidores reales, pues en un aula existen varias computadoras que podrían estar enviando la información hacia la computadora del profesor al mismo tiempo.

El envío de la información hacia el servidor no se debe hacer ejecutando sentencias SQL desde el cliente ni mandando por la red directamente las sentencias SQL, en ambos casos se crea un gran margen para que agentes externos con malas intensiones dañen el sistema. La primera opción mencionada tiene como inconveniente que requeriría que todos los clientes conocieran un usuario y una contraseña con privilegios de modificar la base de datos que gestiona las trazas del módulo Resultados en la computadora del profesor, y la segunda brinda la posibilidad de que se manden a ejecutar sentencias SQL directamente en

[8] Algunos autores mencionan que en XP a las capacidades y cualidades que debe cumplir el software se les denomina requerimientos funcionales y no funcionales, mientras otros expresan que se les denomina lista de reservas. En esta investigación se utilizarán los términos requerimientos funcionales y no funcionales.

el servidor que podrían estar cargadas de inyecciones SQL.

La solución radica en establecer un sistema de envío y recepción de datos, que posea una estructura que permita interpretar los campos a insertar en la base de datos del servidor y luego preparar las sentencias SQL utilizando mecanismos ya probados que evitan los ataques por inyecciones SQL.

El sistema debe permitir que los clientes envíen una petición de suscripción a su servidor, esta funcionalidad permite a los clientes notificarle a su respectivo servidor el deseo de inscribirse. Una suscripción significa ser autorizado por el profesor para enviar periódicamente la información al servidor. Luego el profesor podrá autorizar o denegar estas computadoras en espera de aprobación.

También se podrán especificar las opciones de sincronización que deben usar los estudiantes, o sea, si va a ser manual o automáticamente, entre otras opciones, creando para ello un perfil de sincronización. Las opciones de acceso a la base de datos de la colección se podrán especificar en el panel de configuraciones de ambas aplicaciones en la sección *Conexión local* o creando un perfil de base de datos para ser utilizado. Dicho perfil se puede cargar desde un archivo existente, esto permite que el profesor pueda crear un perfil de base de datos y exportarlo hacia un archivo, luego en cada una de las máquinas de los estudiantes se puede importar este perfil y ser utilizado.

El proceso de sincronización se podrá realizar manual –para ello se brindará una opción en la aplicación cliente para que se pueda iniciar– o se puede llevar a cabo automáticamente a una hora específica o cada cierto intervalo de tiempo en minutos o en horas. Las opciones de sincronización estarán presentes en los perfiles de sincronización.

Se ha analizado la posibilidad de incorporar a la aplicación servidor un sistema que permita guardar en un archivo de *logs*[9] los sucesos ocurridos en el sistema, por ejemplo, una sincronización incompleta porque el cliente se desconectó antes de tiempo, el inicio y fin de una sincronización realizada desde una PC, etc.

2.2 Requerimientos funcionales y no funcionales

[9] Los logs son registros que se guardan en ficheros para llevar un seguimiento de los sucesos ocurridos que son significativos en un sistema de software. Su aplicación está presente en casi todas las aplicaciones con función de servidores –desde base de datos hasta servidores web– y en los sistemas operativos.

En el desarrollo de sistemas informáticos lo primero que se debe tener en cuenta es qué es lo que el usuario final realmente quiere y necesita, no se puede comenzar a implementar sin tener una visión completa de los requisitos y capacidades con los que el software debe cumplir.

Es muy común encontrar personas que su especialidad es la de entrevistarse con los clientes para identificar las funcionalidades que los mismos esperan. Un software con un mal o incompleto levantamiento de requerimientos está destinado al fracaso. Los equipos de trabajo se deben enfocar en satisfacer a los clientes. Los requerimientos expresan qué debe hacer el sistema a implementar, si los clientes están insatisfechos la culpa recae sobre todo el equipo de desarrollo y el trabajo realizado durante meses y en ocasiones años es en vano.

Para identificar los requerimientos se deben llevar a cabo una serie de entrevistas con los clientes e identificar sus necesidades. Una única entrevista no es suficiente y siempre que se tenga una duda hay que regresar y preguntar al cliente una vez más. En este proceso se deben preguntar cuáles son sus expectativas con el software, cuáles son sus grandes ideas, qué es lo que realmente esperan del software.

Entre las prácticas más utilizadas cuando se están definiendo los requerimientos de una aplicación se encuentran:

- Entrevistas con los clientes: grupo de preguntas con la intención de recabar información sobre un tema. Se entrevistan a personas individuales o en grupos.
- Tormenta de ideas con el cliente: técnica que ayuda a juntar ideas cuando no se está muy seguro de qué es lo más importante a tratar sobre el tema de interés, se recogen muchas ideas inicialmente en un papel y al final se releen con el fin de ordenarlas, la tormenta de ideas debe ser corta de cerca de diez minutos.
- Cuestionarios: son una buena alternativa a las entrevistas, pues pueden ser la única manera de relacionarse con gran cantidad de personas implicadas con el futuro sistema.
- Observación: no hay una forma mejor de saber cómo funcionan los procesos a automatizar que estar dentro del ambiente donde se desplegará el futuro sistema, mediante la observación se

exploran mejor los procesos, se revisa que los procesos identificados se llevan a cabo y permite un mejor entendimiento del entorno donde se empleará el software.

Para la presente investigación los requerimientos se definieron mediante tormentas de ideas entre el equipo de desarrollo y los líderes del Proyecto Multisaber. Como resultado de estas tormentas de ideas se ha podido obtener una lista con los requerimientos que el sistema debería cumplir, los cuales se han dividido en tres secciones: una dedicada a los requerimientos que están presentes tanto en la aplicación que funcionará del lado del cliente ‘’la cual será utilizada por los estudiantes, como en la aplicación que funcionará del lado del servidor, la cual utilizarán los profesores. La segunda sección es para los requerimientos que son propios para la aplicación del lado del cliente y la tercera a las funcionalidades de la aplicación del lado del servidor

2.2.1 Requerimientos funcionales comunes para ambas aplicaciones

R1. Gestionar perfiles de conexión a base de datos.

R1.1 Crear manualmente un perfil de conexión a base de datos.

R1.2 Exportar a un archivo un perfil de conexión a base de datos.

R1.3 Cargar desde un archivo existente un perfil de conexión a base de datos.

R1.4 Cargar automáticamente el perfil de conexión a base de datos usado.

R1.5 Permitir regresar al perfil de conexión a base de datos por defecto.

R2. Gestionar perfiles de sincronización.

R2.1 Crear perfil de sincronización manual.

R2.2 Crear perfiles de sincronización automática.

R2.3 Eliminar perfil de sincronización.

R2.4 Cargar automáticamente perfil de sincronización en uso al inicio de la aplicación.

R2.5 Cambiar perfil de sincronización en uso.

R3. Mostrar mensajes en el área de notificación del sistema.

2.2.2 Requerimientos funcionales de la aplicación del lado del cliente

R4. Realizar petición de inscripción.

R5. Realizar sincronización.

R5.1 Realizar sincronización diaria a una hora específica.

R5.2 Realizar sincronización cada un intervalo de tiempo.

R5.3 Realizar sincronización manual.

2.2.3 Requerimientos funcionales de la aplicación del lado del servidor

R6. Reportar sucesos del sistema en un fichero de logs.

R6.1 Reportar inicio de sincronización.

R6.2 Reportar fin de sincronización.

R6.3 Reportar sincronización incompleta.

R7. Gestionar los equipos autorizados a sincronizar.

R7.1 Autorizar equipo.

R7.2 Denegar equipo.

R7.3 Listar equipos en espera de aprobación.

R7.4 Listar equipos autorizados.

R7.5 Eliminar equipo autorizado.

R8. Gestionar estado del servidor.

R8.1 Iniciar servidor manualmente.

R8.2 Detener servidor manualmente.

R8.3 Mostrar cantidad de conexiones del servidor.

R8.4 Mostrar puerto de escucha.

R8.5 Cambiar puerto de escucha.

R8.6 Mostrar un aviso en la bandeja del sistema al iniciar el servidor.

R8.7 Mostrar un aviso en la bandeja del sistema al detener el servidor.

R9. Atender peticiones de sincronización.

R9.1 Procesar la información recibida.

R9.2 Notificar a un cliente fin de la sincronización.

R9.3 Mostrar un aviso en la bandeja del sistema al terminarse la sincronización desde una PC.

2.2.4 Requerimientos no funcionales

Los requerimientos no funcionalidades son cualidades con las que debe cumplir el software, las cuales determinan en muchos casos el nivel de calidad y de acabado del software. Existen muchas formas de

clasificar a estos requisitos por categorías. A continuación se muestra la lista de requisitos no funcionales identificados para la solución propuesta.

Usabilidad:

El sistema debe ser lo más fácil posible de manipular, pues las personas que utilizarán el software no serán expertos en asuntos informáticos.

Hardware:

Se requiere de una red interna, donde cada computadora tenga una dirección ip asignada en dicha red. La red puede ser a través de un cable RJ45 o a través de una red inalámbrica, en ambos casos el sistema debe funcionar.

Software:

- El sistema podrá ejecutarse sobre la distribución venezolana GNU\Linux Canaima.
- Se requieren las librerías Qt a partir de la versión 4.5 en adelante.
- El gestor de base de datos MySQL Server 5.1 en adelante.
- Se requiere de un puerto abierto destinado para la aplicación, se ha seleccionado por defecto el 5623, pero se puede cambiar por otro que se estime conveniente.

Restricciones de diseño y de implementación.

- Se utilizará la programación orientada a objetos.
- Se utilizará la programación multihilos[10]. La misma permitirá que el servidor atienda peticiones concurrentemente.

[10] La programación multihilos se le conoce internacionalmente como multithread programming. Consiste en tener varios hilos de ejecución dentro de la misma aplicación en vez de tener solamente uno. Esto se ve mucho en servidores y en aplicaciones que necesitan que ciertos subprocesos de larga duración se ejecuten sin que la interfaz visual deje de responder en ese período.

- Para el desarrollo se emplearán sólo herramientas que cumplan con la condición de ser software libre, como el IDE Qt Creator versión 1.3 y las librerías Qt versión 4.5.
- Se utilizará un estándar para el código fuente.
- El diseño del sistema debe permitir una fácil adaptabilidad y extensibilidad por si en el futuro la colección de Multisaber empleara otro gestor de base de datos.

Apariencia o interfaz externa:

- Se utilizarán los componentes visuales de Qt para crear las interfaces gráficas de usuario.

Requisitos de seguridad:

- **Integridad:** El sistema contará con un mecanismo que permite verificar que la información transmitida por la red llegó íntegramente.
- **Confidencialidad:** A pesar de que la información manejada no es de alta confidencialidad como por ejemplo información secreta, la información manejada por el sistema está protegida de acceso no autorizado mediante el empleo de usuarios y contraseñas.
- **Evitar inyecciones SQL[11]:** Las sentencias SQL que se ejecuten se deben implementar de tal manera que eviten las inyecciones SQL.

Requisitos de soporte:

Se debe brindar un manual de usuario para facilitar el trabajo a los futuros usuarios.

Portabilidad:

El sistema será fácil de migrar hacia otros sistemas operativos como Windows y Mac OS. Lo cual se logra gracias a la utilización de las librerías Qt que son multiplataforma.

[11] La inyección SQL es una técnica utilizada por los hackers para atacar sistemas informáticos, se aprovechan de vulnerabilidades existentes en códigos fuentes inseguros de las aplicaciones, mediante la misma inyectan código malicioso para ejecutar sentencias SQL que pueden dañar las base de datos o leer datos confidenciales.

2.3 Conclusiones

En el presente capítulo se han determinado las principales funcionalidades y cualidades que el futuro sistema debe poseer. Dichas funcionalidades han sido agrupadas y ayudarán a determinar las HU necesarias para implementar la solución.

Uno de los aspectos clave del sistema es la seguridad, es por ello que se abordaron varios aspectos relacionados con el mismo, como la necesidad de programar de tal forma que se eviten posibles sentencias SQL, agregado a esto se ha decidido el empleo del estándar XML para la transferencia de la información gestionada por las trazas, pues el envío de sentencias SQL por la red constituye un enorme hueco de seguridad. La aplicación cliente no necesita conocer la contraseña para acceder a la base de datos del profesor, para eso es que se hizo necesario definir una especie de protocolo de intercambio de información entre los clientes y el servidor.

También se determinaron otros aspectos como la necesidad de lograr un diseño que permita un fácil cambio de gestor de base de datos, pues es probable que en un futuro se necesite brindar soporte para el empleo de otro sistema gestor de base de datos.

Capítulo 3
Exploración y planificación

El presente capítulo está dedicado a las dos primeras etapas que propone XP: la exploración y la planificación. La exploración es la etapa inicial en que se escriben las historias de usuario, se realizan estimaciones y se priorizan las historias de usuario. La planificación permite elaborar un plan de iteraciones y un plan de entregas, por el cual se deberá regir el equipo de trabajo. Estas dos etapas marcan el camino que debe seguir el software. El éxito del software dependerá en gran medida de las decisiones tomadas durante la exploración y la planificación.

3.1 Exploración

El propósito de la exploración es llegar a un entendimiento global entre desarrolladores y el cliente acerca de qué es lo que el futuro sistema debe hacer. Para ello se toman los requerimientos funcionales que se han definido y comienza el proceso de descripciones de las historias de usuario, los análisis en grupo y las tormentas de ideas. Se esclarecen las dudas sobre procesos que deben ser automatizados y que no han sido bien entendidos.

Básicamente se realizan tres operaciones fundamentales durante la exploración: escribir las historias de usuario, estimarlas y dividir historias en otras más pequeñas si se considera necesario.

También los desarrolladores aprovechan para buscar algunas alternativas que pudieran utilizar para implementar la solución, desde distintas maneras intentan implementar un sistema que se asemeje al que deberán implementar luego, prueban varias tecnologías, miden su nivel de conocimiento sobre el tema, se preparan para una eventual batalla con el tiempo.

3.1.1 Historias de usuario

“La historia de usuario es la unidad de funcionalidad en un proyecto de XP. Nosotros demostramos el progreso liberando código probado e integrado que implementa una historia. Una historia debe ser entendible para los clientes y desarrolladores, probable y de valor para el cliente, y lo suficientemente pequeña para que los programadores puedan implementar media docena en una iteración.” (Beck, y otros, 2000 pág. 43).

Las historias de usuario son los equivalentes a los casos de uso en la metodología de desarrollo de

software RUP. A diferencia de RUP, en XP las historias de usuario se describen con un texto bien corto que sintetice la funcionalidad que se busca con la historia de usuario. Las mismas pueden agrupar uno o varios requerimientos funcionales identificados. XP recomienda que las HU sean escritas por el propio cliente. Es por ello que debe existir una buena relación entre los clientes y el grupo de desarrollo, la falta de comunicación entre ambos factores influye directamente en el fracaso del proyecto. Las HU son priorizadas y colocadas de acuerdo a su prioridad en las iteraciones, las cuales se verán más adelante.

A cada HU se le debe asignar una duración, cuya duración no la determina el cliente, sino el grupo de desarrollo, ¿y cómo se hace esto si es el cliente quien prioriza las historias de usuario?, pues bien, una técnica muy conveniente que utilizan los desarrolladores es desglosar las HU en tareas, las cuales son descritas y poseen una duración en días. La duración de una HU está dada por la suma de la duración de sus tareas.

Priorización

En la mayoría de los casos los clientes no poseen grandes conocimientos de informática, pero son capaces de especificar qué funcionalidades poseen mayor prioridad para ellos, o sea, el nivel de importancia, que también se le conoce como *valor del negocio* o *prioridad del negocio*, este valor lo determinan sólo los clientes, los desarrolladores sólo pueden aportar ideas y opiniones, pero el valor para el negocio de una historia de usuario sólo lo puede determinar el cliente. A menudo, terminan por asignarle un nivel de prioridad alto a casi todas las historias de usuario. La segunda parte sería preguntarle al cliente algo así como: ¿qué te gustaría que se implementara primero?, ahora sí se está hablando el mismo idioma del cliente, esta es la gran pregunta que el cliente estaba esperando, pues pueden existir gran cantidad de historias con prioridad alta, pero en las primeras iteraciones se deben poner de éstas las más importantes.

Lo más recomendado es crear unas pequeñas tarjetas donde se coloque una historia de usuario por tarjeta, en la misma aparecerá el título de la historia y una descripción corta:

Título: _____

Descripción: _____

Estas tarjetas se muestran al cliente y entonces es que se procede a priorizar las historias de usuario.

Trazabilidad

El cliente debe ser capaz de asignarle una prioridad a cada una de las historias de usuario, mientras éstas deben ser lo suficientemente capaces de ser probadas por los mismos clientes. Los desarrolladores y clientes llegan a un acuerdo sobre las *pruebas de aceptación* que debería pasar una historia de usuario para ser considerada implementada correctamente. Si la prueba de aceptación funciona se puede asumir que la implementación es bastante buena.

Cada cierto tiempo se realizan cambios en el código fuente que en algún momento pudieran provocar que algunas pruebas de aceptación fallen, es por eso que se deben guardar las pruebas de aceptación para probarlas cada cierto tiempo y así se mantiene una trazabilidad bastante buena que asegura que todas las HU estén resueltas.

Para mostrar las HU presentes en la solución de esta investigación se ha decidido dividirlas en tres grupos para evitar confusión, primeramente se muestran las relacionadas con las funcionalidades que son comunes en ambas aplicaciones, luego se muestran las HU que se relacionan con la aplicación del lado del cliente y luego las relacionadas con la aplicación del lado del servidor.

3.1.2 Historias de usuario relacionadas con ambas aplicaciones

3.1.2.1 Historia de usuario "Gestionar perfiles de conexión a base de datos"

RF relacionados:

R1.1 Crear manualmente un perfil de conexión a base de datos.

R1.2 Exportar a un archivo un perfil de conexión a base de datos.

R1.3 Cargar desde un archivo existente un perfil de conexión a base de datos.

R1.4 Cargar automáticamente el perfil de conexión a base de datos usado.

R1.5 Permitir regresar al perfil de conexión a base de datos por defecto.

Historia de usuario	
No.: 1 **Nombre:** Gestionar perfiles de conexión a base de datos	
Usuario: Todos	
Prioridad en el negocio: Alta	**Nivel de complejidad:** Medio
Estimación: 3	**Iteración asignada:** 1
Descripción: El usuario puede especificar las opciones requeridas para poder conectarse a su base de datos local, puede además cargar las opciones a partir de un archivo existente o exportar su configuración hacia un archivo. Además, puede usar unas opciones de conexión que automáticamente pudiera obtener desde su servidor.	
Información adicional (observaciones): Da cumplimiento al requisito R1 Gestionar perfiles de conexión a base de datos.	

3.1.2.2 Historia de usuario "Gestionar perfiles de sincronización"

RF relacionados:

R2.1 Crear perfil de sincronización manual.

R2.2 Crear perfiles de sincronización automática.

R2.3 Eliminar perfil de sincronización.

R2.4 Cargar automáticamente el perfil de sincronización usado al inicio de la aplicación.

R2.5 Cambiar perfil de sincronización en uso.

Historia de usuario	
No.: 2 **Nombre:** Gestionar perfiles de sincronización	
Usuario: Todos	
Prioridad en el negocio: Alta	**Nivel de complejidad:** Medio
Estimación: 5	**Iteración asignada:** 2
Descripción: El usuario es capaz de realizar múltiples acciones con los perfiles de	

sincronización, puede crear uno para que se sincronice manualmente o automáticamente, eliminar un perfil de sincronización y cambiar de perfil de sincronización en uso. Además, el sistema debe ser capaz de cargar automáticamente el perfil de sincronización usado al iniciarse la aplicación.
Información adicional (observaciones): Da cumplimiento al requisito R2 Gestionar perfiles de sincronización.

3.1.2.3 Historia de usuario "Mostrar mensajes en el área de notificación del sistema"

RF relacionados:

R3 Mostrar mensajes en el área de notificación del sistema.

Historia de usuario	
No.: 3 **Nombre:** Mostrar mensajes en el área de notificación del sistema	
Usuario: Todos	
Prioridad en el negocio: Media	**Nivel de complejidad:** Bajo
Estimación: 2	**Iteración asignada:** 2
Descripción: Se muestran al usuario mensajes informativos a través del área de notificación del sistema.	
Información adicional (observaciones): Da cumplimiento al requisito R3 Mostrar mensajes en el área de notificación del sistema.	

3.1.3 Historias de usuario relacionadas con la aplicación del lado del cliente.

3.1.3.1 Historia de usuario "Realizar petición de inscripción"

RF relacionados:

R4. Realizar petición de inscripción.

Historia de usuario	
No.: 4 **Nombre:** Realizar petición de inscripción	
Usuario: Estudiante	
Prioridad en el negocio: Media	**Nivel de complejidad:** Medio
Estimación: 1	**Iteración asignada:** 1
Descripción: El usuario puede realizar una petición de inscripción a un servidor para ser autorizado a enviar información hacia el mismo.	
Información adicional (observaciones): Da cumplimiento al requerimiento R4 Realizar petición de inscripción.	

3.1.3.2 Historia de usuario "Realizar sincronización"

RF relacionados:

R5.1 Realizar sincronización diaria a una hora específica.

R5.2 Realizar sincronización cada un intervalo de tiempo.

R5.3 Realizar sincronización manual.

Historia de usuario	
No.: 5 **Nombre:** Realizar sincronización	
Usuario: Estudiante	
Prioridad en el negocio: Alta	Nivel de complejidad: Alto
Estimación: 12	**Iteración asignada:** 1
Descripción: El usuario puede llevar a cabo una sincronización manual o automática de acuerdo a las opciones de sincronización que tiene configurado.	
Información adicional (observaciones): Se da cumplimiento al requerimiento R5 Realizar sincronización.	

3.1.4 Historias de usuario relacionadas con la aplicación del lado del servidor

3.1.4.1 Historia de usuario "Reportar sucesos del sistema en un fichero de logs"

RF relacionados:

R6.1 Reportar inicio de sincronización.

R6.2 Reportar fin de sincronización.

R6.3 Reportar sincronización incompleta.

Historia de usuario	
No.: 6 **Nombre:** Reportar sucesos del sistema en un fichero de logs	
Usuario: Todos	
Prioridad en el negocio: Baja	**Nivel de complejidad:** Medio
Estimación: 1	**Iteración asignada:** 2
Descripción: El sistema guardará en un fichero los logs generados al ocurrir sucesos de interés.	
Información adicional (observaciones): Se da cumplimiento al requerimiento R6 Reportar sucesos del sistema en un fichero de logs.	

3.1.4.2 Historia de usuario "Gestionar los equipos autorizados a sincronizar"

RF relacionados:

R7.1 Autorizar equipo.

R7.2 Denegar equipo.

R7.3 Listar equipos en espera de aprobación.

R7.4 Listar equipos autorizados.

R7.5 Eliminar equipo autorizado.

Historia de usuario	
No.: 7 **Nombre:** Gestionar los equipos autorizados a sincronizar	
Usuario: Profesor	
Prioridad en el negocio: Alta	**Nivel de complejidad:** Media
Estimación: 2	**Iteración asignada:** 2
Descripción: El usuario realiza operaciones para gestionar los equipos autorizados a sincronizar, tales como: autorizar y denegar equipos.	
Información adicional (observaciones): Se da cumplimiento al requerimiento R7 Gestionar los equipos autorizados a sincronizar.	

3.1.4.3 Historia de usuario "Gestionar estado del servidor"

RF relacionados:

R8.1 Iniciar servidor manualmente.

R8.2 Detener servidor manualmente.

R8.3 Mostrar cantidad de conexiones del servidor.

R8.4 Mostrar puerto de escucha.

R8.5 Cambiar puerto de escucha.

R8.6 Mostrar un aviso en la bandeja del sistema al iniciar el servidor.

R8.7 Mostrar un aviso en la bandeja del sistema al detener el servidor.

Historia de usuario
No.: 8 **Nombre:** Gestionar estado del servidor
Usuario: Profesor

Prioridad en el negocio: Alta	**Nivel de complejidad:** Alto
Estimación: 5	**Iteración asignada:** 1
Descripción: El usuario podrá realizar las tareas para administrar el servidor de sincronización, tales como: iniciar el servidor, detenerlo, cambiar puerto, etc.	
Información adicional (observaciones): Se da cumplimiento al requerimiento R8 Gestionar estado del servidor.	

3.1.4.4 Historia de usuario "Atender peticiones de sincronización"

RF relacionados:

R9.1 Procesar la información recibida.

R9.2 Notificar a un cliente fin de la sincronización.

R9.3 Mostrar un aviso en la bandeja del sistema al terminarse la sincronización desde una PC.

Historia de usuario	
No.: 9 **Nombre:** Atender peticiones de sincronización	
Usuario: Profesor	
Prioridad en el negocio: Alta	**Nivel de complejidad:** Alto
Estimación: 12	**Iteración asignada:** 1
Descripción: La aplicación recibe la información enviada por sus clientes, la procesa, actualiza su información local con la nueva información recibida y se le muestra al profesor un mensaje indicándole que la sincronización desde una PC determinada se realizó.	
Información adicional (observaciones): Se da cumplimiento al requerimiento R9 Atender peticiones de sincronización.	

3.2 Planificación

Durante esta etapa se generan una serie de artefactos y se toman importantes decisiones en el proyecto. La interacción con los clientes es crucial, los mismos influyen en el proceso de priorizar los requerimientos funcionales que son más importantes, luego se pasa a definir las iteraciones, sus duraciones, el esfuerzo, para al final obtener un Plan de entregas que guarde correspondencia con las necesidades del proyecto, que su alcance sea posible.

"Toda técnica de planificación de software debe tratar de crear visibilidad, de tal manera que todo individuo involucrado en el proyecto pueda realmente ver cuan largo es el proyecto. Esto significa que se necesitan metas claras, metas que no sean dudosas y que claramente representen progreso. Las metas deben además ser cosas que toda persona involucrada con el proyecto, incluyendo el cliente, pueda entender y aprender a confiar en éstas. " (Beck, y otros, 2000 pág. 13).

La planificación es necesaria porque hay que asegurar que se está siempre trabajando en la parte más importante del software que se pueda en ese momento, también porque hay que coordinar el trabajo con otras personas involucradas en el proyecto y para que si un evento inesperado ocurre se necesita entender cuáles son las consecuencias reales que este cambio le traerá al proyecto.

3.2.1 Iteraciones

Antes de comenzar a implementar XP propone que una vez que se han definido cuáles son las HU y las mismas se han priorizado de acuerdo al valor que tiene cada una en el negocio, se pasa a definir las iteraciones. Una iteración es un mini-proyecto, donde se implementan una serie de HU, lo más recomendado es colocar las HU de mayor prioridad en las primeras iteraciones.

Pero no es colocarlas deliberadamente, en este proceso participa el cliente quien es el encargado de especificar cuáles deben implementarse primero, pero si existe alguna HU de menos prioridad de la cual depende alguna de las de mayor prioridad, se recomienda que se coloque en la misma iteración donde se decidió que se debía implementar la HU de mayor prioridad. Es por eso que es común que se

implementen HU de menor prioridad en las primeras iteraciones y que otras de mayor prioridad pasen para iteraciones siguientes.

Los equipos de desarrollo deben decidir la longitud de las iteraciones y se recomienda que todas tengan la misma duración, pues permite que el equipo se acostumbre a seguir un ritmo. XP recomienda que deben ser cortas, de entre 2 y 3 semanas, pues los ciclos cortos permiten que el mismo cliente pruebe constantemente el software a partir de los casos de prueba. Alargar la longitud de las iteraciones en ciclos mayores de 1 mes permite que errores indeseados sean detectados de forma tardía. Para la presente investigación se ha decidido utilizar iteraciones de 21 días laborables.

3.2.2 Plan de iteraciones

Para la confección del Plan de iteraciones se colocaron primero las HU con mayor prioridad del negocio y se fueron colocando en las iteraciones de forma tal que la implementación de cada una de ellas se relacione con la implementación de las otras que están en su misma iteración.

En la primera iteración las dos primeras HU que se colocaron son las 5 y la 9, debido a que se considera que son las más importantes, las mismas son las encargadas de llevar a cabo el proceso de sincronización de la información, pero ocurre que en ambas historias de usuario existen tareas que no se podían llevar a cabo al momento de comenzar su implementación porque su implementación dependía de que se definan otros aspectos relacionados con la Colección Multisaber que se relacionan con las tablas de la base de datos que se debe sincronizar. Esos aspectos aún no se habían especificado bien y dependían de decisiones que se deberían tomar por otros equipos de desarrollo del proyecto que trabajan a la par de esta investigación. Pero varias de las funcionalidades de ambas historias de usuario eran necesarias para poder implementar y probar otras que se implementarían más adelante. Para estos casos XP propone que se deben dividir las HU en otras más pequeñas y esas funcionalidades que quedan pendientes se implementan en futuras iteraciones, es por ello que se ha decidido que las historias de usuario 5 y 9 se dividan en dos historias cada una. Una parte de cada una se va a implementar en la primera iteración y el resto de sus funcionalidades quedan para iteraciones siguientes.

La segunda iteración posee una duración de 21 días al igual que la primera y se organizaron las HU que

en ella se van a implementar a partir de sus respectivas prioridades. Se decidió además pasar para la tercera iteración la implementación de las tareas que habían quedado pendiente de las historias de usuario 5 y 9.

A continuación se muestran las historias de usuario organizadas por iteraciones.

Iteraciones	Orden de las historias de usuario a implementar	Prioridad	Días	Total de días de la iteración
Iteración 1	5 a). Realizar sincronización	Alta	7	21 días
	9 a). Atender peticiones de sincronización	Alta	6	
	1. Gestionar perfiles de base de datos	Alta	3	
	8. Gestionar estado del servidor	Alta	5	
Iteración 2	2. Gestionar perfil de sincronización	Alta	8	21días
	7. Gestionar equipos autorizados a sincronizar	Alta	5	
	3. Mostrar mensajes en el área de notificación del sistema.	Media	3	
	4. Realizar petición de inscripción	Media	3	
	6. Reportar sucesos del sistema en un fichero de logs.	Baja	2	
Iteración 3	5 b). Realizar sincronización.	Alta	6	12 días
	9 b) Atender peticiones de sincronización	Alta	6	

3.2.3 Plan de entregas

El Plan de entregas constituye un esfuerzo para unir a los clientes con el equipo de desarrollo, donde cada cual juega su rol. Metafóricamente hablando, el cliente conduce el Plan de entregas mientras que los desarrolladores ayudan a navegar sobre él. El cliente decide qué historias de usuario colocar en la entrega y cuáles se implementan luego, mientras que los programadores se encargan de las estimaciones correspondientes a cada entrega.

El Plan de entregas debe ser lo más honesto posible porque constituye un documento oficial por el cual los clientes le exigen a los desarrolladores la entrega de las distintas versiones del producto. Si un proyecto no es capaz de cumplir con el Plan de entregas pactado podría perder muchísimo dinero, baja la moral del equipo de desarrollo y los clientes dejan de confiar, aumentando la incertidumbre y la desesperación de los clientes que no se ven respaldados.

Siempre se debe mantener una buena comunicación con los clientes, explicarles cuando una tarea se retrasa o cuando no se puede cumplir con el plan pactado. Es mejor avisarle a tiempo al cliente cuando algo no marcha a buen ritmo que decepcionarlo al final cuando ellos han malgastado su tiempo y su dinero por gusto, por eso el consejo es: siempre habla con el cliente.

A continuación se muestra el Plan de entregas:

Entregable	Fin de la primera iteración (13 de marzo)	Fin de la segunda iteración (10 de abril)	Fin de la tercera iteración (28 de abril)
Cliente:	versión 0.2	versión 0.5	versión 1.0
Servidor:	versión 0.2	versión 0.5	versión 1.0

3.3 Conclusiones

En el presente capítulo se ha dado cumplimiento a las dos primeras fases de la metodología de desarrollo XP. Los principales objetivos del capítulo han sido delimitar el alcance del proyecto y realizar las

estimaciones de tiempo necesarias para entregar software probado y en tiempo. Para cumplir con dichos objetivos se crearon y documentaron las HU necesarias, se priorizaron y se colocaron en las iteraciones donde debían ser implementadas, al final se pudo crear el Plan de entregas por el cual el equipo de desarrollo se debe regir para cumplir con las entregas en tiempo.

Sin duda alguna la culminación del presente capítulo marca una etapa decisiva en el desarrollo de la solución de la presente investigación.

Capítulo 4
Implementación y pruebas

En el presente capítulo se abordan dos fases sumamente importantes en el ciclo de desarrollo que propone XP: la fase de *implementación* y la fase de *pruebas*. Se muestran algunos artefactos de importancia para el diseño de la aplicación y las pruebas realizadas.

Los módulos de Qt

Las librerías Qt ofrecen una gran cantidad de clases que se pueden utilizar en los programas. Todas las clases de la biblioteca de Qt pertenecen a un módulo en específico, así es como se dividen arquitectónicamente sus clases. En la presente investigación se hace uso de varios de sus módulos, estos son:

- QtCore, ofrece las clases principales del núcleo de Qt.
- QtSql, brinda acceso a las clases para el trabajo con base de datos.
- QtGui, posee las clases para crear aplicaciones con interfaz gráfica de usuario.
- QtNetwork, abarca las clases necesarias para acceder a las interfaces de red de las computadoras.

La programación multihilos en Qt

La programación multihilos se puede lograr en Qt gracias a la clase QThread que pertenece al módulo QtCore. La clase QThread se puede heredar desde cualquier clase de la que se quiera crear uno o varios hilos de ejecución.

Entre las principales funciones a tener en cuenta de la clase QThread se encuentran los métodos run() y start(). El método run() se debe redefinir en la clase que herede de QThread, pues es el método que se ejecuta cuando se inicia la ejecución del hilo, una vez que el programa se sale de este método el hilo es cerrado automáticamente. El método start() es el método que se debe llamar cuando se desea que el hilo comience su ejecución.

Parecería que toda la lógica del hilo se debe encerrar en el método run(). Pues no es así, un truco es utilizar en la clase que correrá varios hilos un atributo booleano[12] para saber si el hilo está detenido, este

[12] Es un tipo de datos que sólo toma dos valores lógicos "*true*" *y "false"* que significan verdadero o falso.

atributo se inicializa en *false* y en el método run() se coloca un ciclo *while* que esté preguntando constantemente por el estado de este atributo, cuando esté en *true* entonces se sale del ciclo *while* y el hilo se cierra, mientras sea *false* no se saldrá del ciclo *while* y el hilo se mantiene en ejecución realizando otras tareas.

4.1 Diseño

XP propone que para representar los distintos componentes del sistema se pueden utilizar los diagramas de UML, pero no especifica cuáles. Se pueden utilizar todos los que el equipo de desarrollo considere necesarios para alcanzar un entendimiento lo más detallado posible de la aplicación.
Otro aspecto a tener en cuenta en XP es que el diseño debe ser limpio, sencillo e incremental. Un diseño incremental es un diseño que se va actualizando a medida que se van agregando nuevas HU a las iteraciones, y ese mismo ímpetu fue el utilizado en la presente investigación.

4.1.1 Diagrama de clases del sistema

El diagrama de clases del sistema es una descripción del sistema que se basa en mostrar las clases que componen al mismo. Es una de las descripciones más detalladas de un software porque en el mismo se muestran las clases, con sus funcionalidades y sus relaciones. La figura 4.1 muestra el diagrama de clases de la aplicación servidor, mientras que la figura 4.2 muestra el diagrama de clases de la aplicación cliente.

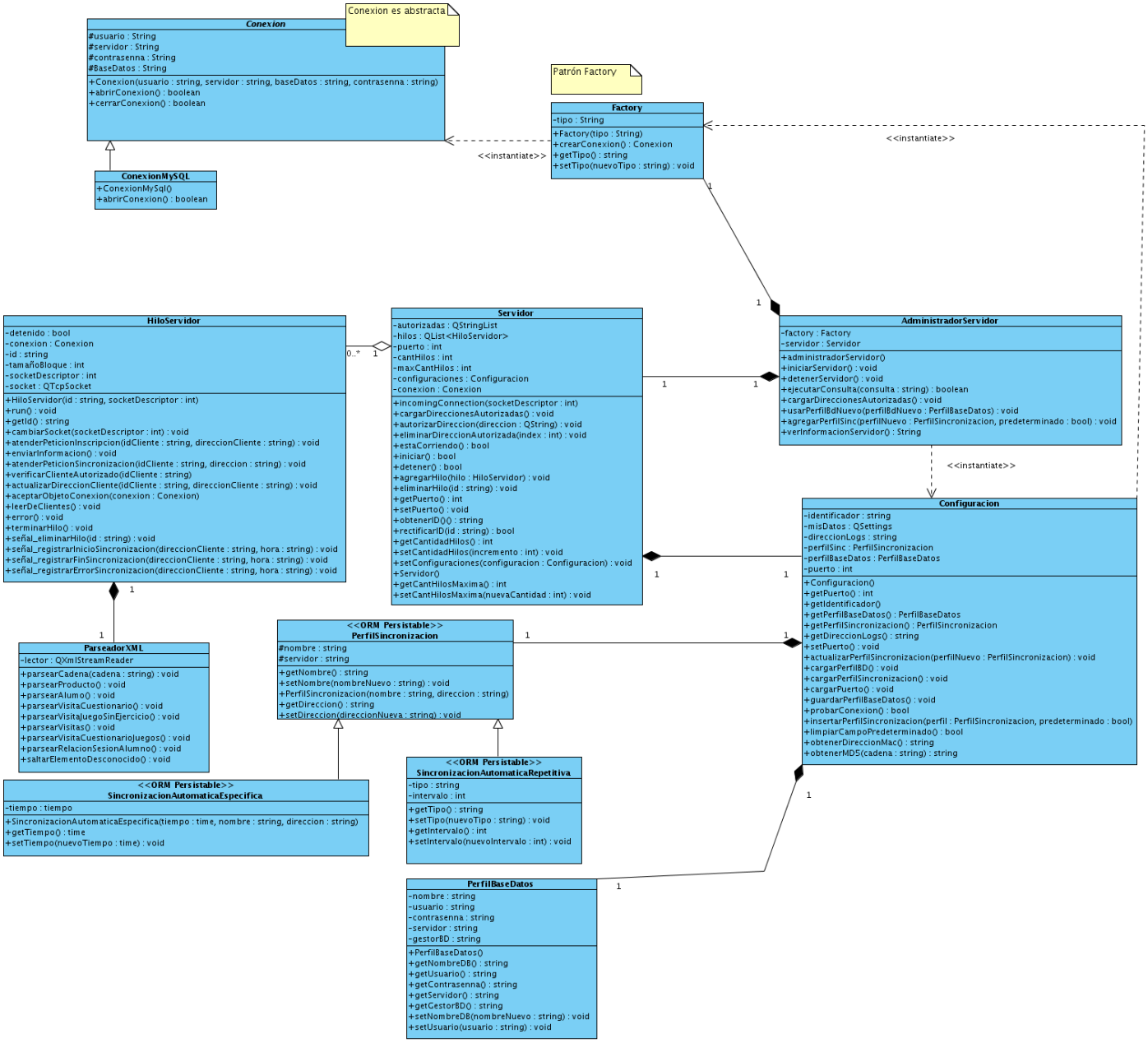

Figura 4.1 Aplicación servidor. Diagrama de clases del sistema

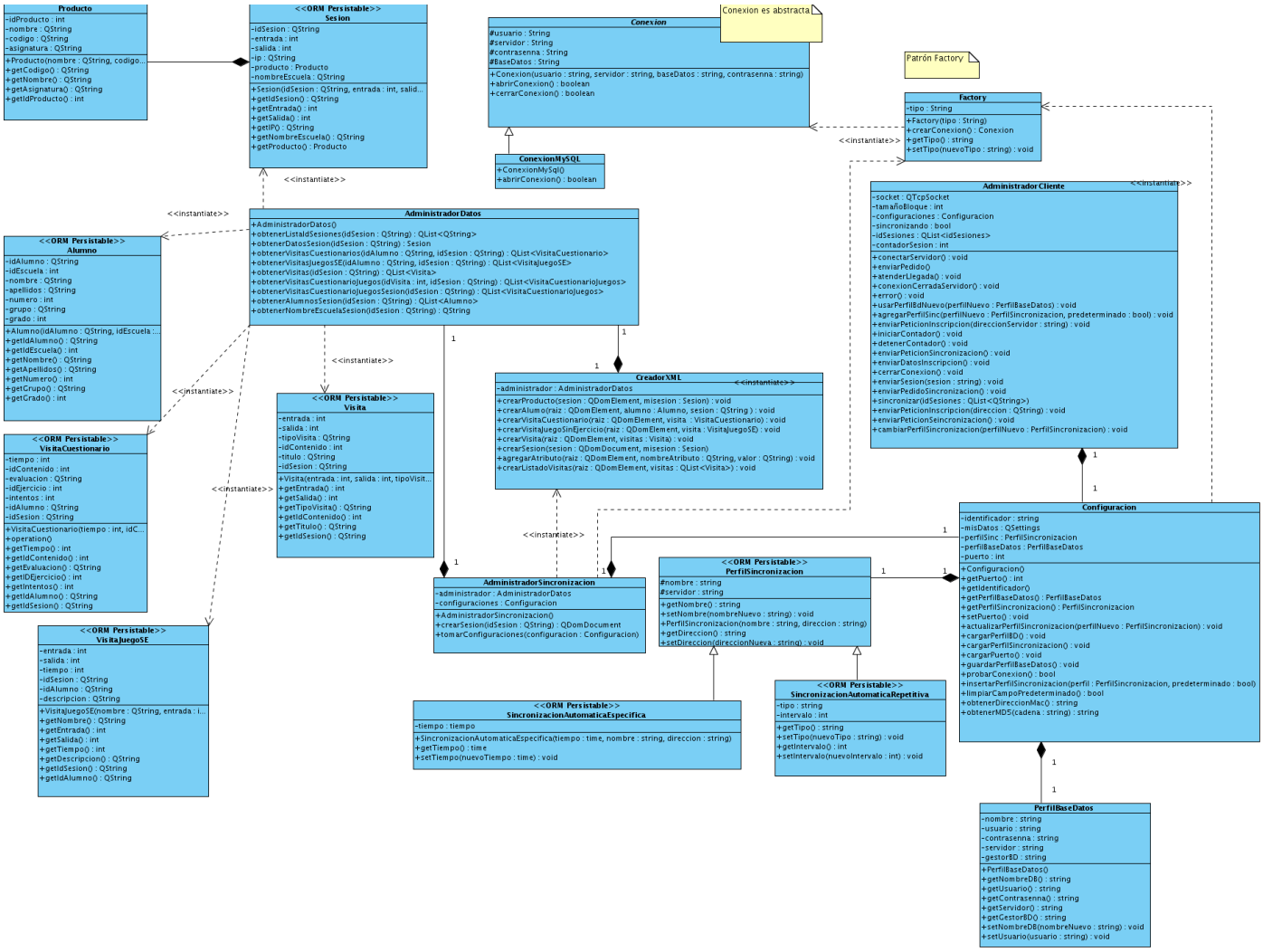

Figura 4.2 Aplicación cliente. Diagrama de clases del sistema.

Cabe destacar que se decidió utilizar el patrón de diseño creacional *Factory* para crear instancias de las clases hijas de la clase *"Conexion"*, se hizo necesario utilizar este patrón para hacer la aplicación más adaptable a posibles cambios en el futuro, en este caso, es útil para que la aplicación se adapte fácilmente a la utilización de otro sistema gestor de base de datos. Pues a través de la clase *"Factory"* se puede obtener una instancia para conectarse a una base de datos en MySQL, Postgres u Oracle, lo único que habría que especificarle es el atributo *"tipo"* de la clase "*Factory*" para que nos devuelva un objeto con la posibilidad de abrir conexión a una base de datos del tipo dado. En este caso sólo se brinda soporte para

MySQL a través de la clase *"ConexionMySql"* que hereda de la clase abstracta *"Conexion".* En un futuro si se decidiera utilizar el gestor de base de datos PostgreSql por ejemplo, el único cambio sería agregar una clase *"ConexionPostgres"* que heredara de la clase *"Conexion"*, la clase *"Factory"* haría el resto del trabajo.

4.1.2 Modelo de despliegue

El modelo de despliegue muestra la distribución física de los dispositivos que interactúan con el sistema, así como el protocolo de comunicación entre los mismos. El sistema que se va a implementar necesita de una computadora donde estará funcionando la aplicación servidor, la misma tendrá instalado la aplicación servidor y el servidor de base de datos MySQL, en otras palabras, no se usará una computadora para que solamente contenga el servidor de base de datos, sino que el mismo estará ejecutándose junto con la aplicación correspondiente. Por otro lado, se necesita de al menos una computadora que haga la función de cliente, pero se debe aclarar que aunque sea cliente posee instalado en la misma PC el servidor de base de datos MySQL, en dicho servidor se guarda la información generada por la interacción de los estudiantes con los productos de Multisaber que tengan instalado. Una vez explicados estos aspectos se puede entender mejor el diagrama de despliegue.

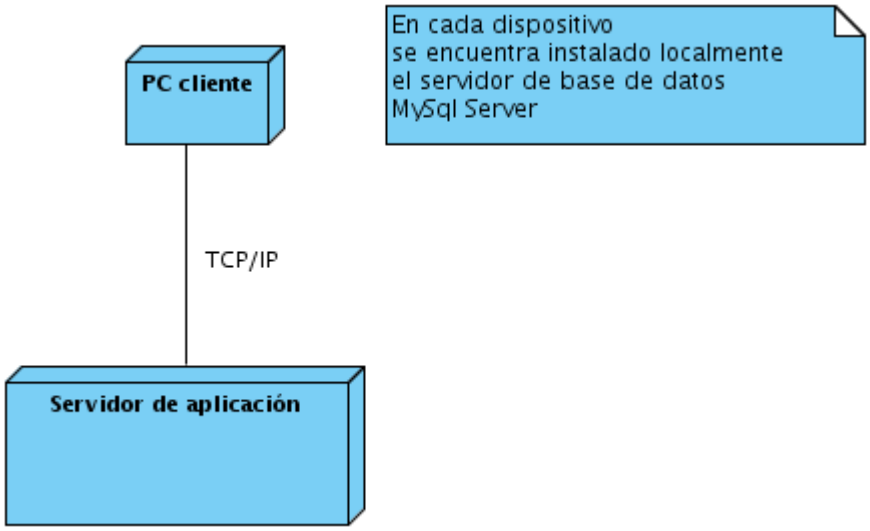

Figura 4.3 Diagrama de despliegue

El protocolo de comunicación es el TCP/IP porque mediante el mismo es que se procederá a enviar la información gestionada por el módulo Resultados, la comunicación entre las aplicaciones cliente y servidor se hará mediante el uso de sockets. [13]

4.1.3 Diagrama de paquetes

El diagrama de paquetes ofrece un nivel de abstracción que permite apreciar cómo se divide el sistema en ciertas agrupaciones de elementos que se les puede denominar paquetes. Este diagrama puede ayudar a repartir entre los desarrolladores la implementación por paquetes.

En la presente investigación se decidió dividir la aplicación en tres paquetes, donde cada paquete representa a una de las capas arquitectónicas en las que se divide, tanto la aplicación cliente como la aplicación servidor, que en este caso son las mismas capas para ambas, por ello poseen el mismo diagrama de paquetes.

[13] Los sockets se definen por el empleo de un protocolo de comunicación, una dirección IP y un puerto. Los mismos permiten que aplicaciones situadas en diferentes máquinas se comuniquen entre sí.

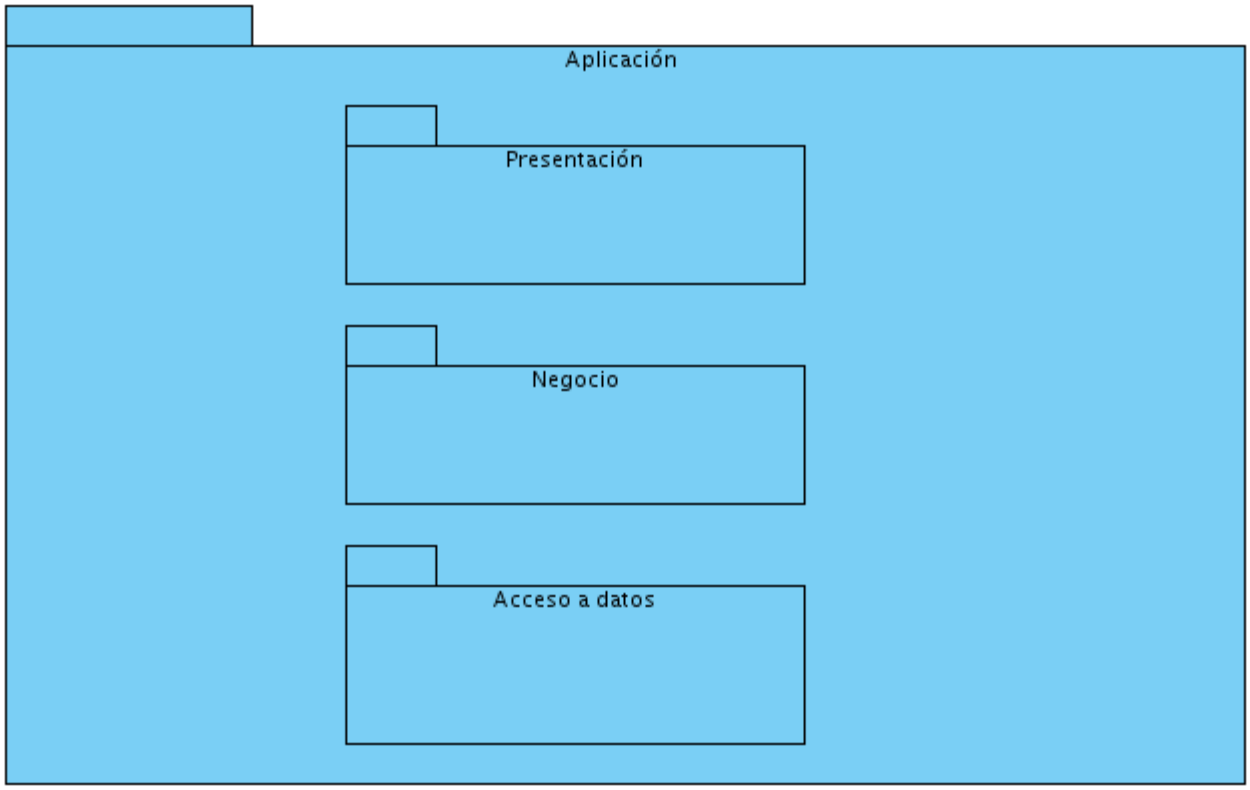

Figura 4.4 Diagrama de paquetes

4.1.4 Diagramas de componentes

Los diagramas de componentes son otros de los artefactos de gran importancia para la arquitectura de un software. Modelan la dependencia entre los diferentes componentes que forman parte de un sistema de software. Se ha decidido crear un diagrama de componentes por cada paquete presente en las aplicaciones en desarrollo.

También se incluyen en los mismos las dependencias de algunas clases que pertenecen a varios módulos de Qt, que sin la inclusión de las mismas no es posible compilar ni la aplicación cliente ni la aplicación servidor. Los módulos de Qt que se utilizan han sido colocados en paquetes distintos y resaltados con otro color.

4.1.4.1 Diagramas de componentes de la aplicación servidor

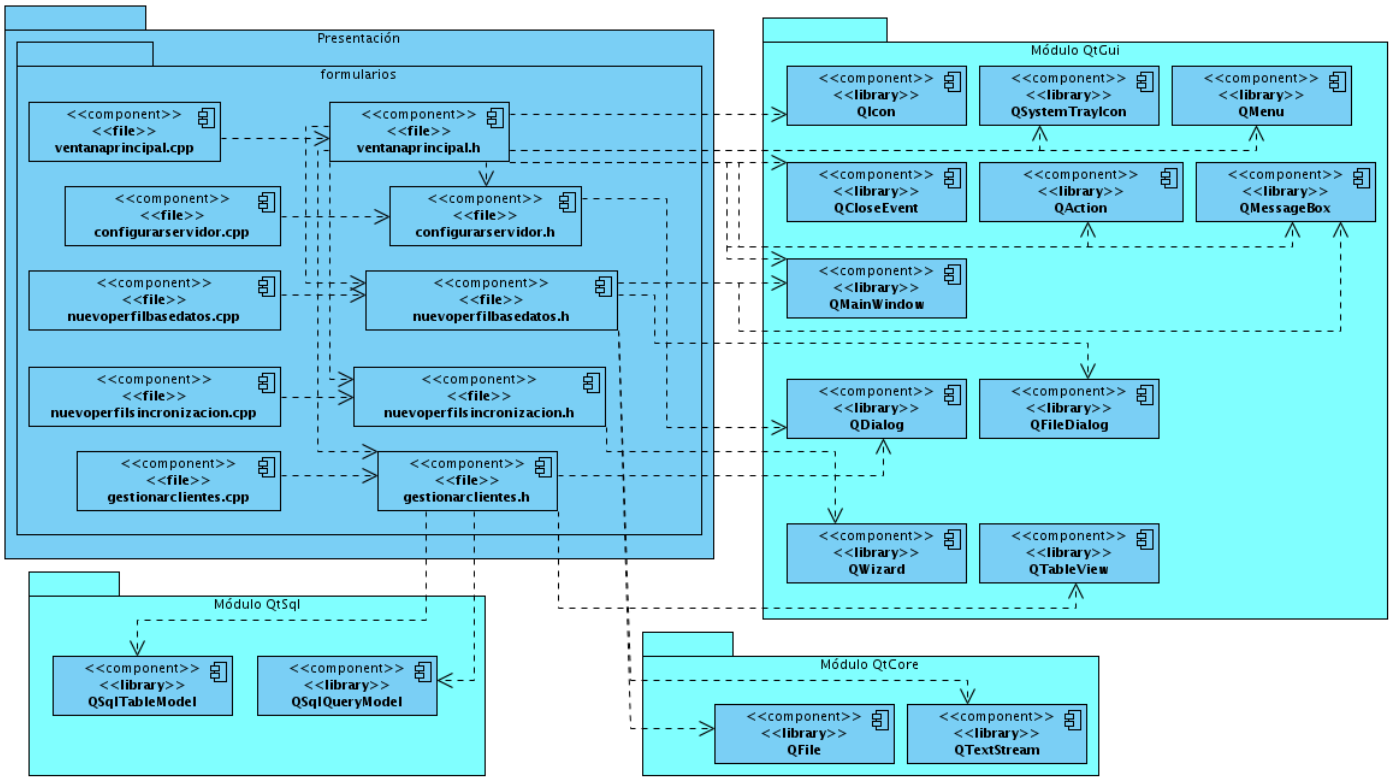

Figura 4.5 Aplicación servidor. Diagrama de componentes. Paquete Presentación

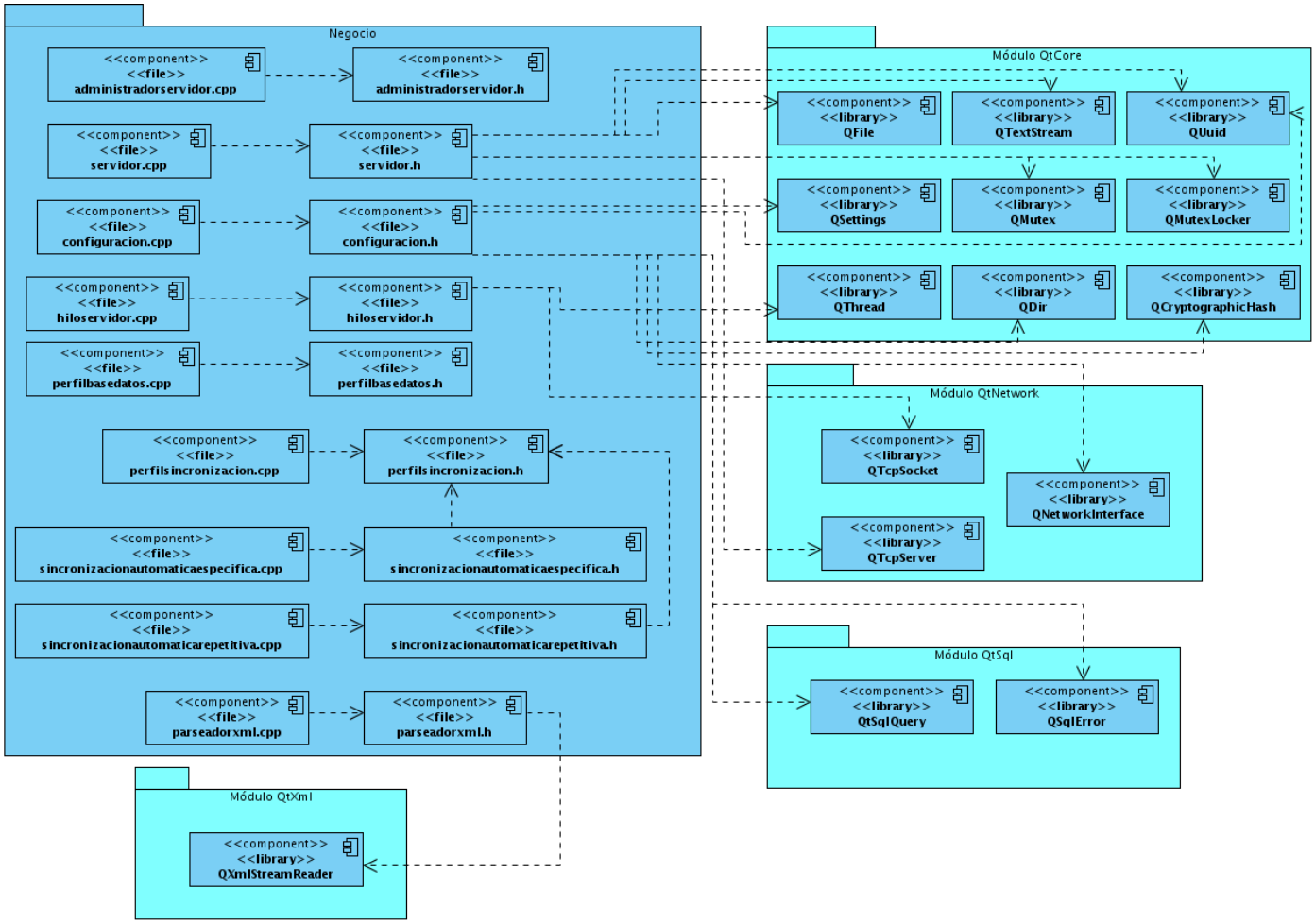

Figura 4.6 Aplicación servidor. Diagrama de componentes. Paquete Negocio

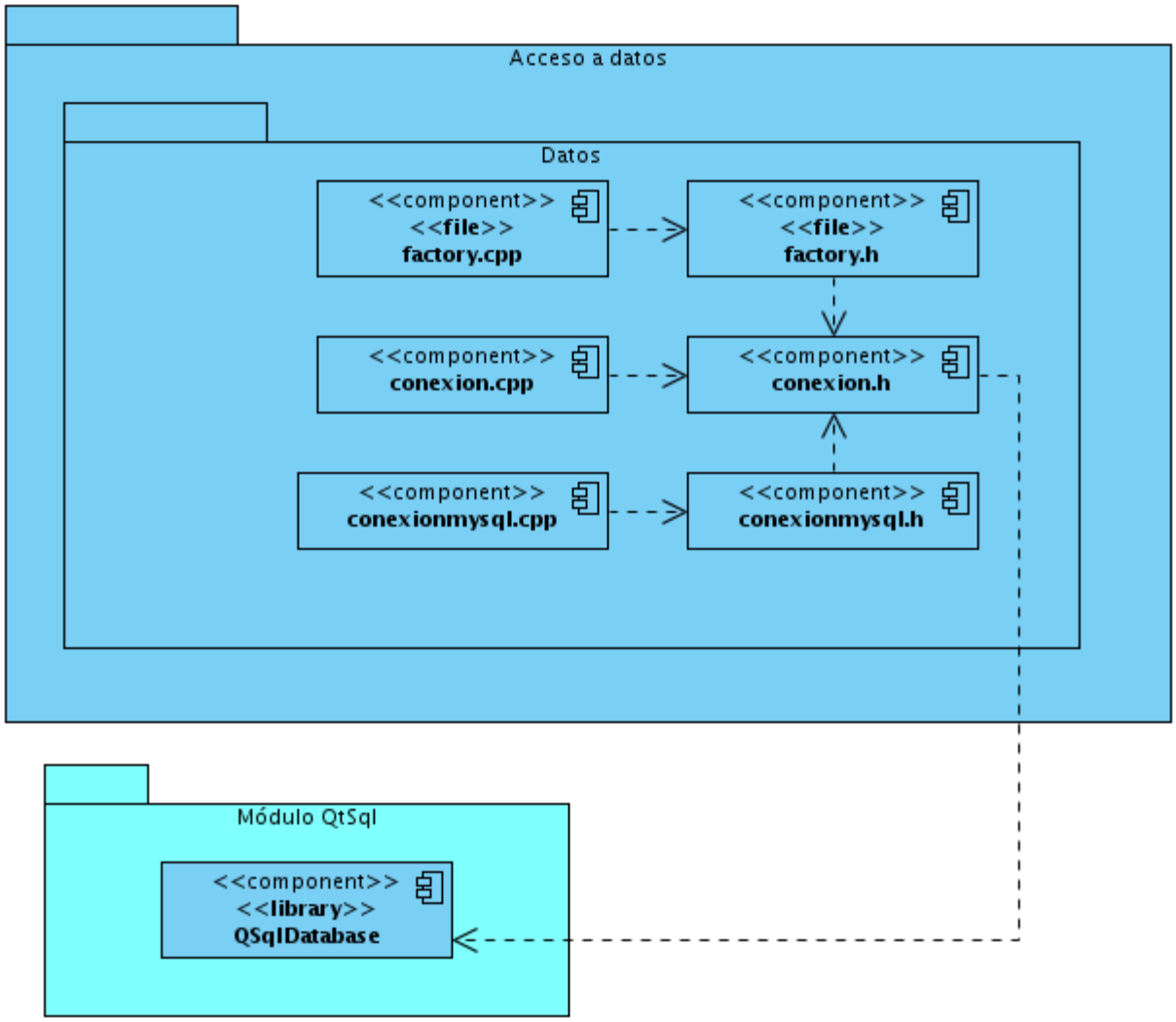

Figura 4.7 Aplicación servidor. Diagrama de componentes. Paquete Acceso a datos

4.1.4.2 Diagramas de componentes de la aplicación cliente

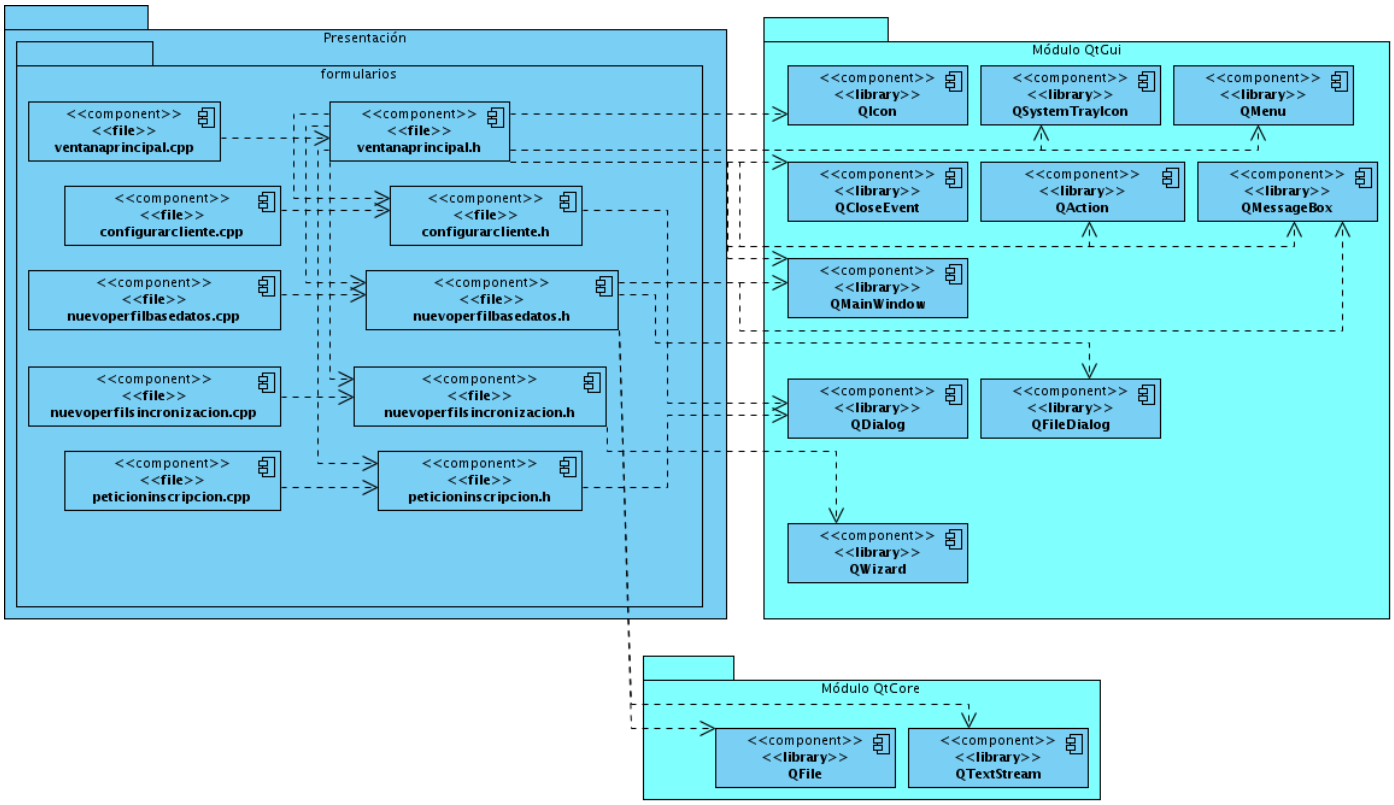

Figura 4.8 Aplicación cliente. Diagrama de componentes. Paquete Presentación

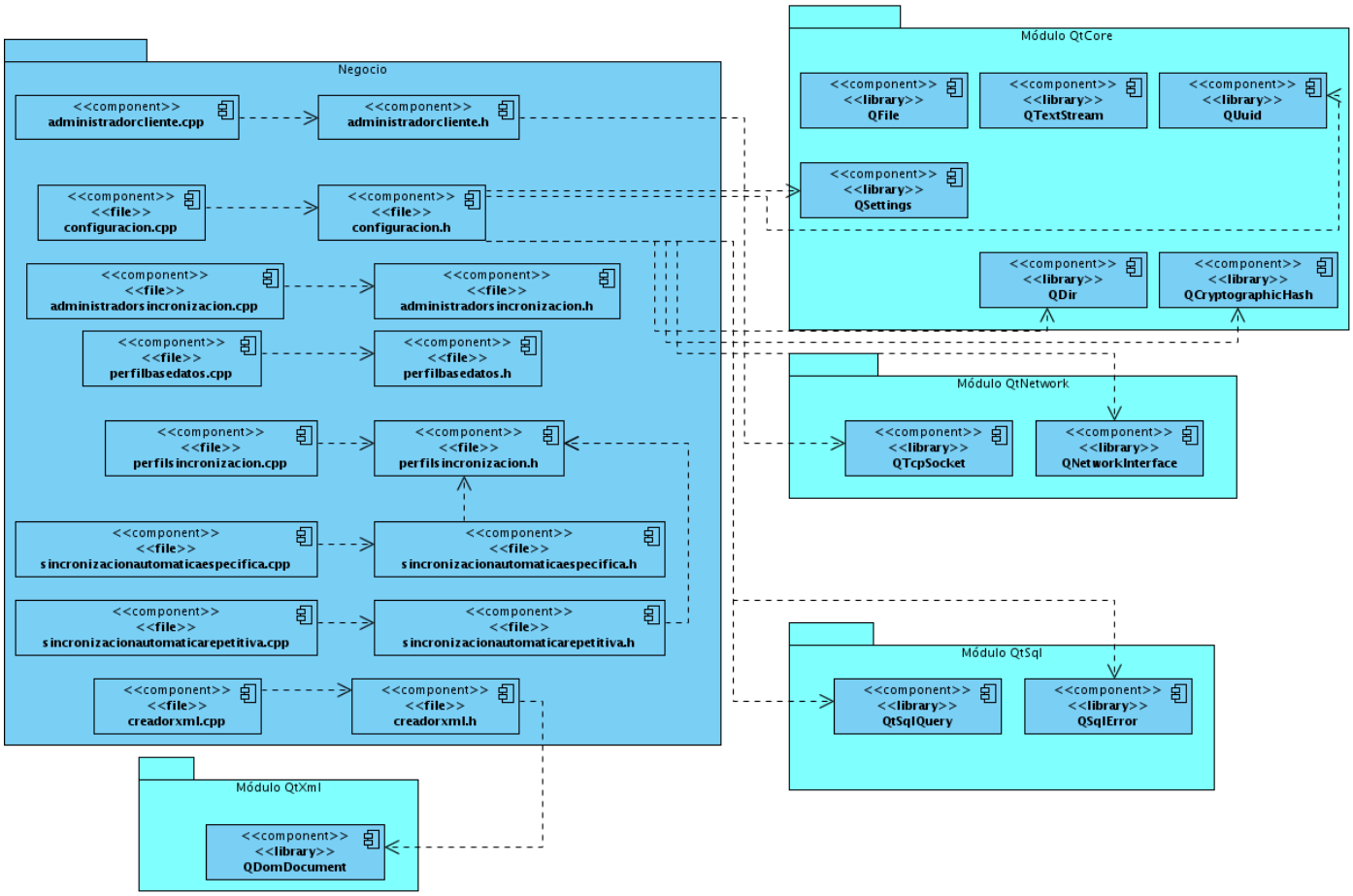

Figura 4.9 Aplicación cliente. Diagrama de componentes. Paquete Negocio

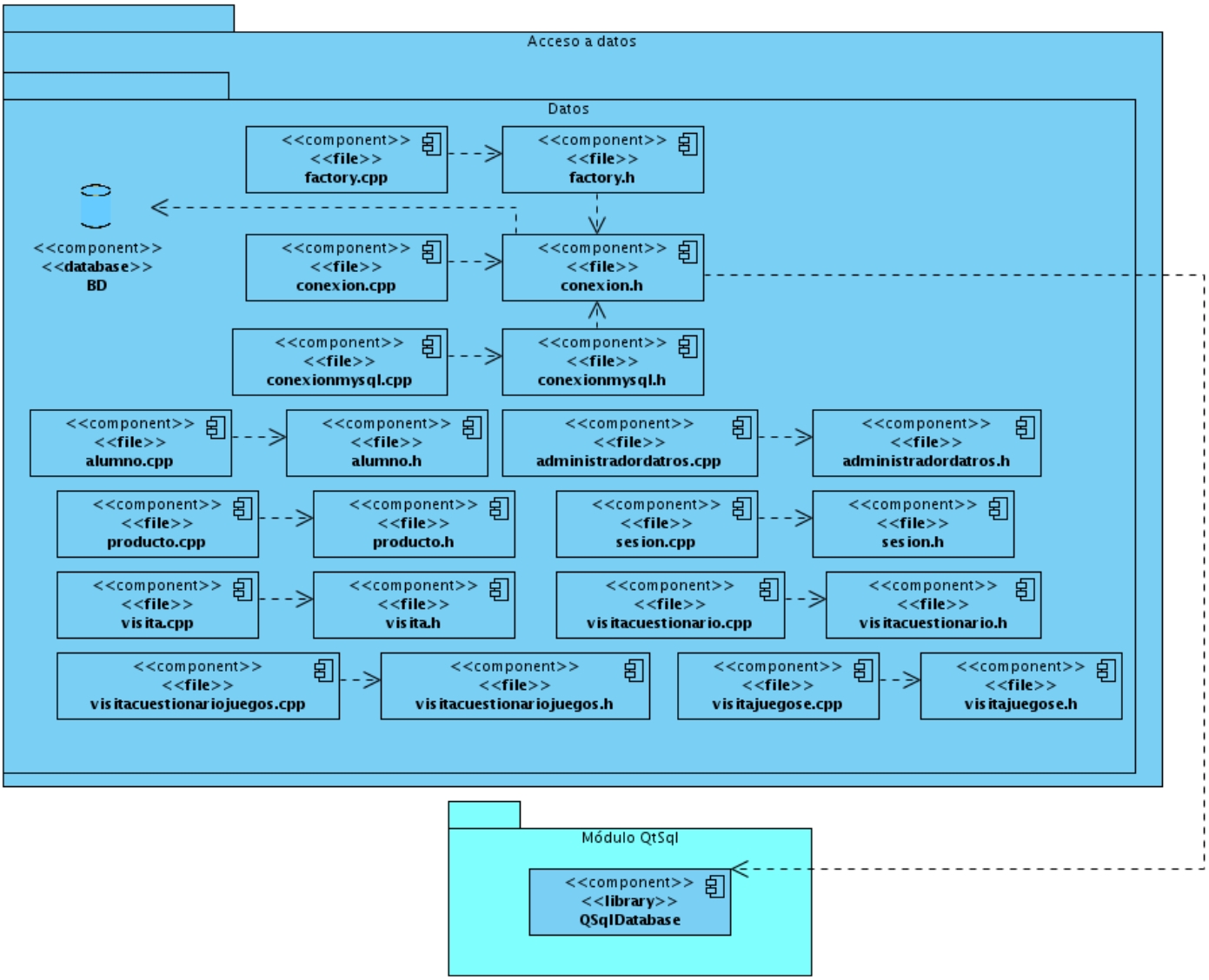

Figura 4.10 Aplicación cliente. Diagrama de componentes. Paquete Acceso a datos

4.2 Implementación

La implementación es la fase en la cual los desarrolladores escriben el código fuente de las aplicaciones informáticas y no se termina hasta que todas las funcionalidades hayan sido implementadas y se haya

probado su correcto funcionamiento.

XP tiene la peculiaridad de que se prueba antes de implementar, es una consecuencia directa de aplicar TDD. Este proceso permite que las pruebas se incorporen tempranamente a los procesos de desarrollo. Las pruebas van de la mano a la implementación, no se debe esperar a que termine la implementación para probar un software. Es un ciclo que ha probado su eficiencia y consiste en los siguientes pasos:

- *"Escribe una prueba.*
- *Compila la prueba. Debe de fallar inicialmente porque la funcionalidad no se ha implementado aún.*
- *Implementa sólo lo suficiente para que compile.*
- *Corre la prueba y mírala fallar.*
- *Implementa sólo lo necesario para pasar la prueba.*
- *Corre la prueba y mírala funcionar.*
- *Refactoriza el código para una mejor claridad y elimina duplicaciones.*
- *Repite el proceso desde el principio. "* (C. Wake, 2000 pág. 14)

4.2.1 1ª iteración

La primera iteración se dedicó a las funcionalidades de mayor importancia, las más prioritarias. La misma tuvo una duración de 21 días y se implementaron las HU: 5 a), 9 a), 1 y 8 en ese mismo orden. Las tareas realizadas fueron las siguientes:

Tarea	
Número de tarea: 1	
Nombre de la tarea: Definir estándar de codificación	
Tipo de tarea: Configuración	**Estimación:** 3
Fecha inicio: 11/2/2010	**Fecha fin:** 13/2/2010
Programador responsable: Rafael Jacobo Hidalgo Urbino	
Descripción: Redacción de un estándar de codificación para la sintaxis del código y sus comentarios.	

Tarea	
Número de tarea: 2	**Número de HU:** 5
Nombre de la tarea: Clase para conectar con el servidor, enviar información y recibir respuesta.	
Tipo de tarea: Implementación	**Estimación:** 6
Fecha inicio: 15/2/2010	**Fecha fin:** 20/2/2010
Programador responsable: Rafael Jacobo Hidalgo Urbino	
Descripción: Desarrollo de las funcionalidades necesarias para que el cliente se conecte al servidor, le envíe información y reciba información desde el mismo utilizando un socket para la comunicación. **Clases:** AdministradorCliente	

Tarea	
Número de tarea: 3	**Número de HU:** 5
Nombre de la tarea: Acceder a la información local	
Tipo de tarea: Implementación	**Estimación:** 1
Fecha inicio: 22/2/2010	**Fecha fin:** 22/2/2010
Programador responsable: Rafael Jacobo Hidalgo Urbino	
Descripción: Desarrollo de las clases necesarias para conectarse a la base de datos local. **Clases:** Factory, Conexión, ConexionMySql	

Tarea	
Número de tarea: 4	**Número de HU:** 9

Nombre de la tarea: Clases para crear un servidor y que el mismo atienda peticiones concurrentemente	
Tipo de tarea: Implementación	**Estimación:** 6
Fecha inicio: 23/2/2010	**Fecha fin:** 1/3/2010
Programador responsable: Rafael Jacobo Hidalgo Urbino	
Descripción: Desarrollo de la clase Servidor y la clase HiloServidor, se debe redefinir el método incomingConnection() para que al detectar una nueva conexión se cree un objeto de la clase HiloServidor y llame a su método start() para iniciar el hilo. **Clases:** Servidor, HiloServidor	

Tarea

Número de tarea: 5	**Número de HU:** 1
Nombre de la tarea: Clases para almacenar un perfil de base de datos	
Tipo de tarea: Implementación	**Estimación:** 1
Fecha inicio: 2/3/2010	**Fecha fin:** 2/3/2010
Programador responsable: Rafael Jacobo Hidalgo Urbino	
Descripción: Desarrollar una clase que almacene los datos que lleva un perfil de conexión de base de datos para que los mismos se puedan cargar por defecto al iniciar la aplicación y modificar luego si se hace necesario. **Clases:** PerfilBasedatos	

Tarea

Número de tarea: 6	**Número de HU:** 1
Nombre de la tarea: Formulario para crear, importar y exportar un nuevo perfil de base de datos.	
Tipo de tarea: Implementación	**Estimación:** 1
Fecha inicio: 3/3/2010	**Fecha fin:** 3/3/2010
Programador responsable: Rafael Jacobo Hidalgo Urbino	
Descripción: Crear un formulario nuevo que al llenarlo se pueda crear un nuevo perfil de base de datos, debe permitir además importar un perfil de base de datos existente desde un archivo, así como exportar hacia un archivo un perfil de base de datos creado. **Clases:** NuevoPerfilBasedatos	

Tarea

Número de tarea: 7	**Número de HU:** 1
Nombre de la tarea: Clase para cargar y salvar aspectos de configuración de la aplicación	
Tipo de tarea: Implementación	**Estimación:** 1
Fecha inicio: 4/3/2010	**Fecha fin:** 4/3/2010
Programador responsable: Rafael Jacobo Hidalgo Urbino	
Descripción: Implementar una clase que permita cargar automáticamente el perfil de base de datos en uso, así como guardar sus cambios para ser cargados automáticamente al iniciar la aplicación. **Clases:** Configuración	

Tarea	
Número de tarea: 8	**Número de HU:** 8
Nombre de la tarea: Clase para administrar el estado del servidor	
Tipo de tarea: Implementación	**Estimación:** 5
Fecha inicio: 5/3/2010	**Fecha fin:** 10/3/2010
Programador responsable: Rafael Jacobo Hidalgo Urbino	
Descripción: Implementar una clase que permita iniciar y detener el servidor, así como que brinde información del estado del mismo: si está corriendo o no, cantidad de conexiones que tiene en un momento dado y el puerto por el cual escucha. **Clases:** AdministradorServidor	

4.2.2 2ª iteración

Tarea	
Número de tarea: 9	**Número de HU:** 2
Nombre de la tarea: Crear nuevo perfil de sincronización	
Tipo de tarea: Implementación	**Estimación:** 5
Fecha inicio: 15/3/2010	**Fecha fin:** 19/3/2010
Programador responsable: Rafael Jacobo Hidalgo Urbino	
Descripción: Implementar clases para crear un nuevo perfil de sincronización manual o automática, además crear un formulario que permita seleccionar las opciones que tendrá el perfil de sincronización. Permitir además que el perfil creado se pueda establecer como predeterminado, o se pueda exportar a un archivo. **Clases:** NuevoPerfilSincronización, PerfilSincronización, SincronizaciónAutomáticaEspecífica, SincronizaciónAutomáticaRepetitiva	

Tarea	
Número de tarea: 10	**Número de HU:** 7
Nombre de la tarea: Autorizar y denegar equipos	
Tipo de tarea: Implementación	**Estimación:** 2
Fecha inicio: 20/3/2010	**Fecha fin:** 22/3/2010
Programador responsable: Rafael Jacobo Hidalgo Urbino	
Descripción: Implementar funcionalidades para autorizar, mostrar equipos en espera de aprobación y los equipos autorizados, permitir denegar la aprobación a un equipo en espera y eliminar a un equipo autorizado con anterioridad. **Clases:** GestionarClientes	

Tarea	
Número de tarea: 11	**Número de HU:** 3
Nombre de la tarea: Mensajes en el área de notificación	
Tipo de tarea: Implementación	**Estimación:** 3
Fecha inicio: 23/3/2010	**Fecha fin:** 25/3/2010
Programador responsable: Rafael Jacobo Hidalgo Urbino	
Descripción: Agregar funcionalidad en el formulario principal de la aplicación cliente y servidor para que muestre un icono de la aplicación en el área de notificación, también que muestre mensajes de tres tipos: informativos, advertencias o de error en esa zona. **Clases:** VentanaPrincipal	

Tarea	
Número de tarea: 12	**Número de HU:** 4
Nombre de la tarea: Petición de inscripción	
Tipo de tarea: Implementación	**Estimación:** 2
Fecha inicio: 26/3/2010	**Fecha fin:** 27/3/2010
Programador responsable: Rafael Jacobo Hidalgo Urbino	

Descripción:

- Agregar un elemento en el menú de la aplicación cliente para realizar una petición de inscripción en el servidor.
- Crear un nuevo formulario que permita escribir la dirección del servidor.
- Agregar funcionalidad en la clase AdministradorCliente para que envíe una petición de inscripción al servidor.
- Agregar funcionalidad en el servidor para que agregue el equipo al listado de equipos en espera.

Clases: PeticiónInscripción, AdministradorCliente

Tarea	
Número de tarea: 13	**Número de HU:** 4
Nombre de la tarea: Atender petición de inscripción en el servidor	
Tipo de tarea: Implementación	**Estimación:** 1
Fecha inicio: 29/3/2010	**Fecha fin:** 29/3/2010
Programador responsable: Rafael Jacobo Hidalgo Urbino	

Descripción:

Agregar funcionalidad en el servidor para que agregue el equipo al listado de equipos en espera.

Clases: HiloServidor

Tarea	
Número de tarea: 14	**Número de HU:** 6
Nombre de la tarea: Sucesos del sistema	
Tipo de tarea: Implementación	**Estimación:** 2
Fecha inicio: 30/3/2010	**Fecha fin:** 31/3/2010
Programador responsable: Rafael Jacobo Hidalgo Urbino	
Descripción: • Agregar variable dirección de logs en la clase Configuración y ponerle una dirección por defecto donde se irán guardando los logs que reporten sucesos de interés que ocurran en la aplicación. • Agregar funcionalidad en la clase Servidor para que los hilos le reporten sucesos a éste y éste a su vez escriba el suceso en el fichero de log, se debe manejar la concurrencia sobre esta funcionalidad para que dos procesos no escriban en el mismo fichero a la misma vez.	
Clases: Configuración, HiloServidor, Servidor	

4.2.3 3ª iteración

En la 3ª iteración se implementaron las HU 5 b) y 9 b) con una duración de 12 días. A continuación se muestran las tareas realizadas.

Tarea	
Número de tarea: 15	**Número de HU:** 5
Nombre de la tarea: Definir la estructura del XML que se usará para enviar las sesiones	
Tipo de tarea: Configuración	**Estimación:** 1
Fecha inicio: 12/4/2010	**Fecha fin:** 12/4/2010
Programador responsable: Rafael Jacobo Hidalgo Urbino	

Descripción: Se define la estructura exacta que tendrá el XML que se crea en el cliente con toda la información referente a una sesión que se va a enviar y que luego se parsea en el servidor para insertar en la base de datos de Multisaber todo lo referente a esa sesión.

Tarea	
Número de tarea: 16	**Número de HU:** 5
Nombre de la tarea: Implementar las clases que brinden acceso a las tablas que se van a sincronizar	
Tipo de tarea: Implementación	**Estimación:** 2
Fecha inicio: 13/4/2010	**Fecha fin:** 14/4/2010
Programador responsable: Rafael Jacobo Hidalgo Urbino	
Descripción: Implementar las clases entidades y las clases con funcionalidades para acceder y modificar las tablas de la base de datos de estas clases entidades.	

Tarea	
Número de tarea: 17	**Número de HU:** 5
Nombre de la tarea: Clase para crear un XML	
Tipo de tarea: Implementación	**Estimación:** 1
Fecha inicio: 15/4/2010	**Fecha fin:** 15/4/2010
Programador responsable: Rafael Jacobo Hidalgo Urbino	
Descripción:	

Implementar clase que permite crear un XML a partir de la información recogida de la base de datos local.

Clases: CreadorXML

Tarea	
Número de tarea: 18	**Número de HU:** 5
Nombre de la tarea: Iniciar sincronización en el cliente	
Tipo de tarea: Implementación	**Estimación:** 2
Fecha inicio: 16/4/2010	**Fecha fin:** 17/4/2010
Programador responsable: Rafael Jacobo Hidalgo Urbino	
Descripción:	
Agregar funcionalidad en el cliente para que se comporte y actúe automáticamente de acuerdo con el perfil de sincronización que esté usando en ese momento y detecte cuando debe empezar la sincronización.	
Clases: AdministradorCliente	

Tarea	
Número de tarea: 19	**Número de HU:** 9
Nombre de la tarea: Parsear XML con información de las sesiones	
Tipo de tarea: Implementación	**Estimación:** 1
Fecha inicio: 19/4/2010	**Fecha fin:** 19/4/2010
Programador responsable: Rafael Jacobo Hidalgo Urbino	
Descripción:	
Implementar la clase ParseadorXML con la responsabilidad de parsear el XML recibido con la información de las sesiones.	

Clases: ParseadorXML

Tarea	
Número de tarea: 20	**Número de HU:** 9
Nombre de la tarea: Adicionar sesión nueva en el servidor	
Tipo de tarea: Implementación	**Estimación:** 5
Fecha inicio: 20/4/2010	**Fecha fin:** 24/4/2010
Programador responsable: Rafael Jacobo Hidalgo Urbino	
Descripción: Agregar funcionalidades para llevar a cabo una transacción para insertar la información recogida del XML parseado en las tablas de la base de datos de Multisaber.	

4.3 Pruebas

La metodología de desarrollo XP incorpora las pruebas tempranamente, gracias a aplicar TDD. El desarrollo dirigido por pruebas permite que se vayan probando constantemente los componentes más pequeños del código fuente: las clases, estas pruebas son realizadas por los desarrolladores y forman parte de las llamadas pruebas de unidad, que constituyen el nivel más bajo de las pruebas que se le aplican al software, como se llevan a cabo directamente sobre el código fuente pertenecen a las llamadas pruebas de caja blanca.

En el segundo nivel de pruebas se encuentran las pruebas de integración. Estas pruebas se enfocan en encontrar errores en las interfaces de los componentes y en asegurar que todos los componentes operen correctamente. El tercer nivel corresponde a las pruebas del sistema. Las pruebas del sistema se ejecutan cuando el software funciona como un todo y todos los componentes de software y hardware han sido integrados. Por último, se encuentran las pruebas de aceptación, las cuales se realizan para asegurar que el software está listo para ser desplegado, dichas pruebas las realizan los usuarios finales.

Las pruebas más importantes que se aplican en un proceso de desarrollo basado en la metodología XP son las pruebas de unidad, al aplicar TDD y las pruebas de aceptación, las cuales constituyen acuerdos a los que llegan los clientes con los desarrolladores, un contrato que asegura que las funcionalidades pactadas han sido correctamente implementadas. Las pruebas de aceptación no se realizan solamente al final cuando el software está completo, sino que entre iteraciones se van probando todas las HU que se implementaron en la iteración anterior. Estas pruebas se documentan antes de implementar las HU en una iteración. Este proceso permite que la corrección de los errores detectados al probar el software durante y después de una iteración pueda agregarse como nuevas HU a incorporar en la siguiente iteración.

Las pruebas de unidad realizadas en la presente investigación fueron guardadas en unos archivos de texto dentro de una carpeta denominada *"Pruebas"* ubicada en la carpeta donde se encuentran organizados los ficheros de código fuente de la aplicación. Dichas pruebas están compuestas puramente por código fuente.

A continuación las pruebas de aceptación realizadas en la presente investigación.

4.3.1 Pruebas de aceptación

Caso de Prueba de Aceptación	
Código: HU1_P1	**Historia de Usuario:**1
Nombre: Gestionar perfiles de conexión a base de datos	
Descripción: El usuario puede especificar las opciones requeridas para poder conectarse a su base de datos local, puede además cargar las opciones a partir de un archivo existente o exportar su configuración hacia un archivo.	
Condiciones de ejecución: Aplicación en ejecución	
Entrada/Pasos de ejecución: La aplicación debe darle al usuario la posibilidad de seleccionar la opción Nuevo perfil de base de datos, el usuario selecciona si desea llenar este perfil de forma manual o abrir uno existente desde una dirección física. Una vez que haya llenado las opciones necesarias podrá exportar este perfil hacia un archivo o utilizarlo en la aplicación para poder conectarse a la base de datos local.	
Resultado esperado: Se gestionan los perfiles de conexión a la base de datos.	

Evaluación de la prueba: Prueba satisfactoria

Caso de Prueba de Aceptación	
Código: HU1.1_P1	**Historia de Usuario:** 1
Nombre: Crear un perfil de conexión a base de datos llenando manualmente los campos.	
Descripción: Se crea un perfil de conexión a base de datos, los campos necesarios son tecleados por el usuario.	
Condiciones de ejecución: Aplicación en ejecución	
Entrada/Pasos de ejecución: Se escoge la opción Nuevo perfil de base de datos, seleccionar la opción Especificaré datos manualmente, se completan los campos mostrados y se escogen los botones para la acción que se desee realizar, por ejemplo, si escoge el botón Usar, el perfil que está siendo usado cambiará por el nuevo perfil creado, si escoge el botón Exportar guardará el nuevo perfil en la dirección escogida por el usuario.	
Resultado esperado: Se ha creado un nuevo perfil de base de datos.	
Evaluación de la prueba: Prueba satisfactoria	

Caso de Prueba de Aceptación	
Código: HU1.2_P1	**Historia de Usuario:** 1
Nombre: Exportar a un archivo un perfil de conexión de base de datos	
Descripción: Se crea un nuevo perfil de conexión a base de datos y se guarda en la dirección escogida por el usuario.	
Condiciones de ejecución: Debe haberse creado un perfil de conexión de base de datos.	
Entrada/Pasos de ejecución: Se escoge la opción Nuevo perfil de base de datos, seleccionar cualquiera de las opciones mostradas y dar clic en el botón Exportar.	

Resultado esperado: Se exporta un perfil de conexión a base de datos.
Evaluación de la prueba: Prueba satisfactoria

Caso de Prueba de Aceptación	
Código: HU1.3_P1	**Historia de Usuario:** 1
Nombre: Cargar desde un archivo existente un perfil de conexión de base de datos	
Descripción: Abrir un perfil desde una ubicación física ya sea dispositivo externo o el disco duro.	
Condiciones de ejecución: Que exista un perfil desde una dirección física para ser abierto.	
Entrada/Pasos de ejecución: Se escoge la opción Nuevo perfil de base de datos, seleccionar la opción Cargar uno desde archivo y dar clic en el botón Importar, luego se busca el perfil deseado y se selecciona el botón Abrir.	
Resultado esperado: Se ha cargado un perfil de conexión a base de datos desde una dirección física.	
Evaluación de la prueba: Prueba satisfactoria	

Caso de Prueba de Aceptación	
Código: HU1.4_P1	**Historia de Usuario:** 1
Nombre: Cargar automáticamente el perfil de conexión de base de datos usado.	
Descripción: Cuando la aplicación se carga, se carga automáticamente el último perfil de base de datos que se utilizó, si no se había utilizado ninguno con anterioridad se carga uno por defecto.	
Condiciones de ejecución: Ejecutar la aplicación	
Entrada/Pasos de ejecución:	
Resultado esperado: Se carga automáticamente un perfil de conexión de base de datos.	

Evaluación de la prueba: Prueba satisfactoria

Caso de Prueba de Aceptación	
Código: HU1.5_P1	**Historia de Usuario:** 1
Nombre: Permitir regresar al perfil de conexión de base de datos por defecto	
Descripción: Permite regresar al perfil de base de datos que por defecto trae la aplicación la primera vez que se ejecuta en la sesión de un usuario del sistema operativo.	
Condiciones de ejecución: Aplicación en ejecución	
Entrada/Pasos de ejecución: En el menú de la aplicación el usuario selecciona la opción Editar, luego selecciona la opción Preferencias. Se le muestra un panel con una lista de opciones, selecciona la opción Conexión local, luego da clic en el botón Cargar valores predeterminados.	
Resultado esperado: Se ha permitido regresar al perfil de conexión de base de datos que trae la aplicación por defecto.	
Evaluación de la prueba: Prueba satisfactoria	

Caso de Prueba de Aceptación	
Código: HU2_P1	**Historia de Usuario:** 2
Nombre: Gestionar perfiles de sincronización	
Descripción: Prueba para la funcionalidad de realizar múltiples acciones con los perfiles de sincronización, tales como: crear un perfil para sincronizar manualmente o automáticamente, también puede guardar un perfil de sincronización existente hacia un archivo.	
Condiciones de ejecución: Aplicación en ejecución	
Entrada/Pasos de ejecución: Se selecciona la opción Crear nuevo perfil de sincronización y se llenarán los datos pedidos específicamente para cada tipo de	

sincronización.
Resultado esperado: El perfil es creado correctamente.
Evaluación de la prueba: Prueba satisfactoria

Caso de Prueba de Aceptación	
Código: HU2.1_P1	**Historia de Usuario:** 2
Nombre: Crear perfil para la sincronización manual	
Descripción: La aplicación debe darle la posibilidad al usuario de crear nuevo perfil de sincronización, se selecciona la opción sincronización manual y posteriormente se llenan los campos: Nombre del perfil y dirección del servidor.	
Condiciones de ejecución: La aplicación debe estar en ejecución.	
Entrada/Pasos de ejecución: Se selecciona la opción Crear nuevo perfil de sincronización, seleccionar opción sincronización manual y llenar los campos Nombre del perfil y Dirección del servidor. Luego se muestra un mensaje de felicitación al usuario indicando que se creó el perfil satisfactoriamente y se le brinda la posibilidad de exportarlo hacia un archivo y de Agregarlo a su lista de perfiles de sincronización.	
Resultado esperado: Se crea el perfil correctamente.	
Evaluación de la prueba: Prueba satisfactoria	

Caso de Prueba de Aceptación	
Código: HU2.2_P1	**Historia de Usuario:** 2
Nombre: Crear perfil para la sincronización automática.	
Descripción: La aplicación debe darle la opción al usuario de crear un nuevo perfil de sincronización para que se haga de forma automática.	
Condiciones de ejecución: La aplicación debe estar en ejecución.	
Entrada/Pasos de ejecución: Se selecciona la opción Crear nuevo perfil de	

sincronización, luego se escoge la opción sincronización automática y se completan los campos que se muestran: Nombre del perfil, Dirección del servidor, intervalo de tiempo u hora específica. Luego se muestra un mensaje de felicitación al usuario indicando que se creó el perfil satisfactoriamente y se le brinda la posibilidad de exportarlo hacia un archivo y de Agregarlo a su lista de perfiles de sincronización
Resultado esperado: Se crea un perfil para la sincronización automática.
Evaluación de la prueba: Prueba satisfactoria

Caso de Prueba de Aceptación	
Código: HU2.3_P1	**Historia de Usuario:** 2
Nombre: Exportar un perfil de sincronización existente hacia un archivo	
Descripción: Se exporta un perfil creado hacia un archivo.	
Condiciones de ejecución: Debe de haberse creado un perfil para la sincronización.	
Entrada/Pasos de ejecución: Seleccionar la opción Crear nuevo perfil de sincronización, seleccionar una de las opciones de sincronización y dar clic en el botón exportar el cual guardará el nuevo perfil en la dirección que el usuario especifique.	
Resultado esperado: Se ha exportado un perfil de sincronización hacia un archivo.	
Evaluación de la prueba: Prueba satisfactoria	

Caso de Prueba de Aceptación	
Código: HU2.4_P1	**Historia de Usuario:** 2
Nombre: Cargar un perfil de sincronización a partir de un archivo existente	
Descripción: Se carga un perfil de sincronización ya existente.	
Condiciones de ejecución: Debe existir al menos un perfil de sincronización en una ubicación física.	
Entrada/Pasos de ejecución: Selecciona la opción Abrir perfil, se muestra una interfaz con los datos del perfil seleccionado y se muestran las opciones de agregarlo a	

la lista de perfiles y de usarlo a partir de ese momento.
Resultado esperado: Se ha abierto correctamente el perfil seleccionado.
Evaluación de la prueba: Prueba satisfactoria

Caso de Prueba de Aceptación	
Código: HU3_P1	**Historia de Usuario:** 3
Nombre: Mostrar mensajes en el área de notificación del sistema	
Descripción: Se le muestran al usuario mensajes informativos en la bandeja del sistema.	
Condiciones de ejecución: Que ocurra algún evento que requiera mostrarle un mensaje al usuario en el área de notificación.	
Entrada/Pasos de ejecución: Se realiza alguna acción por parte del usuario que requiera que el sistema le muestre un mensaje.	
Resultado esperado: Se muestra un mensaje en el área de notificación del sistema.	
Evaluación de la prueba: Prueba satisfactoria	

Caso de Prueba de Aceptación	
Código: HU4.1_P1	**Historia de Usuario:** 4
Nombre: Hacer petición de inscripción.	
Descripción: El usuario de la aplicación cliente puede realizar una petición de inscripción al servidor hacia donde desea enviar periódicamente la información gestionada por el módulo Resultados.	
Condiciones de ejecución: El servicio de escuchas del servidor debe de estar en ejecución.	
Entrada/Pasos de ejecución: El usuario navega por el menú de la aplicación cliente y selecciona la opción Cliente, luego selecciona Enviar petición de inscripción. Se le muestra un diálogo donde debe escribir la dirección del servidor. Luego selecciona el	

botón Enviar.
Resultado esperado: La dirección IP del cliente es agregada a la lista de clientes en espera de aprobación del servidor.
Evaluación de la prueba: Prueba satisfactoria

Caso de Prueba de Aceptación	
Código:HU5_P1	**Historia de Usuario:** 5
Nombre: Realizar sincronización	
Descripción: Prueba para la funcionalidad de realizar una sincronización, que puede ser manual o automática.	
Condiciones de ejecución: Debe estar abierta la aplicación cliente y el servicio de escucha de peticiones del servidor debe estar en ejecución.	
Entrada/Pasos de ejecución: Se realiza la sincronización ya sea manual o automática, según las opciones del perfil de sincronización que el cliente está utilizando.	
Resultado esperado: Se ha realizado la sincronización satisfactoriamente.	
Evaluación de la prueba: Prueba satisfactoria	

Caso de Prueba de Aceptación	
Código: HU5.1_P1	**Historia de Usuario:** 5
Nombre: Realizar sincronización diaria a una hora específica.	
Descripción: Se realiza la sincronización de forma automática una vez al día a una hora específica.	
Condiciones de ejecución: La aplicación cliente debe estar abierta y estar configurada para realizar la sincronización automáticamente a una hora del día.	
Entrada/Pasos de ejecución: El sistema automáticamente comienza la sincronización de la información sin que el usuario influya en el inicio de este proceso.	

Resultado esperado: Se ha realizado la sincronización dada una hora específica correctamente.
Evaluación de la prueba: Prueba satisfactoria

Caso de Prueba de Aceptación	
Código: HU5.2_P1	**Historia de Usuario:** 5
Nombre: Realizar sincronización cada un intervalo de tiempo.	
Descripción: Se realiza la sincronización de forma automática varias veces en el día, cada un intervalo de tiempo.	
Condiciones de ejecución: La aplicación cliente debe estar en ejecución y configurada para realizar la sincronización de forma automática cada un intervalo de tiempo, que puede estar dado en minutos o en horas.	
Entrada/Pasos de ejecución: El sistema inicia automáticamente la sincronización sin que el usuario intervenga influya en el inicio de este proceso.	
Resultado esperado: Se realiza la sincronización cada un intervalo de tiempo especificado.	
Evaluación de la prueba: Prueba satisfactoria	

Caso de Prueba de Aceptación	
Código: HU5.3_P1	**Historia de Usuario:** 5
Nombre: Realizar sincronización manual	
Descripción: El usuario selecciona la opción Sincronizar ahora para que el sistema comience el proceso de sincronización de la información gestionada por el módulo Resultados de la Colección Multisaber	
Condiciones de ejecución: La aplicación cliente debe estar en ejecución y haber sido configurada con anterioridad para sincronizar la información hacia el servidor. La computadora cliente debe de haberse autorizado con anterioridad en el servidor para	

poder realizar la sincronización.

Entrada/Pasos de ejecución: El usuario navega por el menú de la aplicación y selecciona la opción Cliente, luego selecciona la opción Sincronizar ahora, esto permite que se inicie el proceso de sincronización.

Resultado esperado: Se realiza la sincronización manual correctamente.

Evaluación de la prueba: Prueba satisfactoria

Caso de Prueba de Aceptación	
Código: HU6_P1	**Historia de Usuario:**6
Nombre: Reportar sucesos del sistema en un fichero de logs.	
Descripción: El sistema guardará en un fichero los logs generados al ocurrir sucesos de interés que necesiten ser registrados para poder ser analizados en un futuro.	
Condiciones de ejecución: Tiene que haber ocurrido un suceso de interés en el sistema.	
Entrada/Pasos de ejecución: Se provoca la ocurrencia de algún suceso que deba ser registrado en los logs de la aplicación.	
Resultado esperado: Se guardan los datos del suceso ocurrido en el fichero de logs de la aplicación.	
Evaluación de la prueba: Prueba satisfactoria	

Caso de Prueba de Aceptación	
Código: HU6.1_P1	**Historia de Usuario:** 6
Nombre: Reportar inicio de sincronización	
Descripción: Se guarda la fecha, hora y la dirección IP desde donde se ha iniciado una sincronización.	
Condiciones de ejecución: El servicio de escucha de peticiones del servidor debe de estar en ejecución.	

Entrada/Pasos de ejecución: Se inicia la sincronización hacia el servidor.
Resultado esperado: Se ha guardado en el fichero de logs del servidor la información que refleja que se ha iniciado la sincronización.
Evaluación de la prueba: Prueba satisfactoria.

Caso de Prueba de Aceptación	
Código: HU6.2_P1	**Historia de Usuario:** 6
Nombre: Reportar fin de sincronización	
Descripción: Se guarda la fecha, hora y dirección IP desde donde se ha finalizado la sincronización.	
Condiciones de ejecución: Servicio de escucha de peticiones del servidor en ejecución y estar una sincronización en curso.	
Entrada/Pasos de ejecución: Se finaliza la sincronización.	
Resultado esperado: Se ha guardado en el fichero de logs del servidor la información que refleja que se ha finalizado la sincronización desde una dirección IP.	
Evaluación de la prueba: Prueba satisfactoria	

Caso de Prueba de Aceptación	
Código: HU6.3_P1	**Historia de Usuario:** 6
Nombre: Reportar sincronización incompleta.	
Descripción: Se informa que ha sucedido una interrupción en la sincronización.	
Condiciones de ejecución: Servicio de escuchas de peticiones del servidor en ejecución y una sincronización iniciada pero no terminada.	
Entrada/Pasos de ejecución: Se interrumpe la sincronización en el cliente.	
Resultado esperado: Se ha reportado la sincronización incompleta.	
Evaluación de la prueba: Prueba satisfactoria	

Caso de Prueba de Aceptación	
Código: HU7_P1	**Historia de Usuario:**7
Nombre: Gestionar los equipos autorizados a sincronizar.	
Descripción: El usuario realiza operaciones para gestionar los equipos autorizados a sincronizar, tales como: autorizar y denegar equipos.	
Condiciones de ejecución: Aplicación servidor en ejecución y el sistema gestor de base de datos en ejecución.	
Entrada/Pasos de ejecución: El profesor acepta o rechaza los equipos que él considere que deben o no ser autorizados para realizar el envío de la información al servidor.	
Resultado esperado: Se obtuvieron los resultados correctamente.	
Evaluación de la prueba: Prueba satisfactoria	

Caso de Prueba de Aceptación	
Código: HU7.1_P1	**Historia de Usuario:** 7
Nombre: Autorizar equipo	
Descripción: Se autoriza un equipo que está en espera de aprobación para poder realizar sincronizaciones con el servidor.	
Condiciones de ejecución: Debe existir un equipo en espera de aprobación.	
Entrada/Pasos de ejecución: El profesor selecciona el equipo que desee sincronizar y lo adiciona a la lista de los equipos autorizados.	
Resultado esperado: Se agrega correctamente el equipo pendiente a la lista de equipos autorizados.	
Evaluación de la prueba: Prueba satisfactoria	

Caso de Prueba de Aceptación

Código: HU7.2_P1	**Historia de Usuario:** 7
Nombre: Denegar equipo	
Descripción: Se elimina un equipo de la lista de equipos autorizados o de la lista de equipos en espera.	
Condiciones de ejecución: Debe existir al menos un equipo en el listado de equipos pendientes o en el listado de equipos autorizados. El servidor de base de datos local debe de estar en ejecución.	
Entrada/Pasos de ejecución: El profesor accede a la lista de equipos autorizados y en espera de ser aprobados, selecciona uno y oprime el botón Eliminar.	
Resultado esperado: Se eliminó un equipo correctamente.	
Evaluación de la prueba: Prueba satisfactoria	

Caso de Prueba de Aceptación

Código: HU7.4_P1	**Historia de Usuario:** 7
Nombre: Listar equipos en espera de aprobación	
Descripción: Se muestra una lista con los equipos en espera de aprobación.	
Condiciones de ejecución: Servidor de base de datos local en ejecución.	
Entrada/Pasos de ejecución: El usuario navega por el menú de la aplicación y selecciona la opción Servidor, luego selecciona la opción Administrar autorizaciones y el sistema debe mostrarle un listado de equipos en espera de aprobación.	
Resultado esperado: Se muestra la lista de equipos que se encuentran en espera de ser aprobados.	
Evaluación de la prueba: Prueba satisfactoria	

Caso de Prueba de Aceptación

Código: HU7.5_P1	**Historia de Usuario:** 7
Nombre: Listar equipos autorizados	
Descripción: Se muestra la lista de equipos autorizados por el profesor.	
Condiciones de ejecución: Servidor de base de datos local en ejecución.	
Entrada/Pasos de ejecución: El usuario navega por el menú de la aplicación y selecciona la opción Servidor, luego selecciona la opción Administrar autorizaciones y el sistema debe mostrarle un listado de equipos autorizados.	
Resultado esperado: Se muestra un listado con los equipos aprobados.	
Evaluación de la prueba: Prueba satisfactoria	

Caso de Prueba de Aceptación

Código: HU7.6_P1	**Historia de Usuario:** 7
Nombre: Eliminar equipo autorizado	
Descripción: Se elimina un equipo autorizado.	
Condiciones de ejecución: Al menos debe existir un equipo autorizado y el servidor de base de datos local debe estar en ejecución.	
Entrada/Pasos de ejecución: De la lista de los equipos el profesor elimina los equipos que él considere que no deben estar en la lista de equipos autorizados.	
Resultado esperado: Se elimina correctamente un equipo autorizado.	
Evaluación de la prueba: Prueba satisfactoria	

Caso de Prueba de Aceptación

Código: HU8.1_P1	**Historia de Usuario:** 8
Nombre: Iniciar servidor manualmente	
Descripción: El usuario inicia el servicio de escucha de peticiones del servidor	
Condiciones de ejecución: Que el puerto que utiliza la aplicación servidor se pueda	

abrir en el servidor y que el mismo no esté siendo ya utilizado por otra aplicación.

Entrada/Pasos de ejecución: El usuario navega por el menú de la aplicación y selecciona primero el menú Servidor, luego la opción Iniciar servicio de escucha. Opcionalmente puede apretar la combinación de teclas Ctrl + I para lograr el mismo propósito.

Resultado esperado: Se inicia el servicio de escucha y se le muestra un mensaje en el área de notificación indicando que el servidor se ha iniciado correctamente.

Evaluación de la prueba: Prueba satisfactoria

Caso de Prueba de Aceptación

Código: HU8.2_P1 **Historia de Usuario:** 8

Nombre: Detener servidor manualmente

Descripción: El usuario puede detener el servicio de escucha que se encuentra en ejecución en el servidor

Condiciones de ejecución: Que el servicio de escucha del servidor haya sido iniciado con anterioridad.

Entrada/Pasos de ejecución: El usuario navega por el menú de la aplicación y selecciona primero el menú Servidor, luego la opción Detener servicio de escucha. Opcionalmente puede apretar la combinación de teclas Ctrl + D para lograr el mismo propósito.

Resultado esperado: Se detiene el servicio de escucha y se le muestra un mensaje en el área de notificación indicando que el servidor se ha detenido correctamente.

Evaluación de la prueba: Prueba satisfactoria

Caso de Prueba de Aceptación

Código: HU8.3_P1 **Historia de Usuario:** 8

Nombre: Mostrar información del estado del servidor

Descripción: El usuario puede ver la cantidad de conexiones del servidor en un momento dado y el puerto por el cual está escuchando peticiones.
Condiciones de ejecución: Que el servicio de escucha del servidor haya sido iniciado con anterioridad.
Entrada/Pasos de ejecución: El usuario navega por el menú de la aplicación y selecciona primero el menú Servidor, luego la opción Detener servicio de escucha. Opcionalmente puede apretar la combinación de teclas Ctrl + D para lograr el mismo propósito.
Resultado esperado: Se detiene el servicio de escucha y se le muestra un mensaje en el área de notificación indicando que el servidor se ha detenido correctamente.
Evaluación de la prueba: Prueba satisfactoria

Caso de Prueba de Aceptación	
Código: HU8.4_P1	**Historia de Usuario:** 8
Nombre: Cambiar el puerto de escucha	
Descripción: Como medida de seguridad el usuario puede cambiar el puerto por el cual escucha el servidor.	
Condiciones de ejecución: ----	
Entrada/Pasos de ejecución: El usuario navega por el menú de la aplicación y selecciona Editar, luego Preferencias. Se le muestra una ventana con una lista de opciones de configuración a las que puede acceder, de ellas selecciona la opción Opciones del servidor. A continuación le aparecerán varias pestañas a la derecha, en la pestaña de Avanzado marca la opción Cambiar puerto de escucha, escribe un número para el nuevo puerto de escucha y oprime el botón Cambiar.	
Resultado esperado: Se cambia el puerto de escucha de la aplicación.	
Evaluación de la prueba: Prueba satisfactoria	

Caso de Prueba de Aceptación	
Código: HU9_P1	**Historia de Usuario:** 9
Nombre: Atender peticiones de sincronización	
Descripción: El servidor atiende las peticiones de sincronización que le llegan.	
Condiciones de ejecución: Que el servicio de escucha de peticiones del servidor se encuentre en ejecución.	
Entrada/Pasos de ejecución: La aplicación cliente se conecta al servidor, envía la petición de sincronización. El servidor verifica que el cliente esté autorizado y en caso afirmativo le informa al cliente cual es el Id de la última sesión que éste le envió. A partir de este momento el cliente procede a enviar las sesiones a partir de la última que le había enviado al servidor.	
Resultado esperado: Se adicionó en la base de datos del servidor la información proveniente del cliente.	
Evaluación de la prueba: Prueba satisfactoria	

Caso de Prueba de Aceptación	
Código: HU9.3_P1	**Historia de Usuario:** 9
Nombre: Mostrar un aviso en la bandeja del sistema al terminarse la sincronización desde una PC.	
Descripción: Se muestra mensaje indicando fin de la sincronización.	
Condiciones de ejecución: Debe de estar habilitada el área de notificación en el entorno de escritorio donde se ejecuta la aplicación. La aplicación servidor debe estar en ejecución y una sincronización desde un cliente terminada.	
Entrada/Pasos de ejecución: En el área de notificación del sistema donde está ejecutándose la aplicación servidor se muestra un mensaje indicando que ha terminado la sincronización desde una dirección IP.	
Resultado esperado: Se obtuvieron los resultados correctamente.	

Evaluación de la prueba: Prueba satisfactoria

4.4 Conclusiones

En el presente capítulo se abordaron los temas que hacen alusión a la implementación de la solución y a las pruebas realizadas al software obtenido. Fueron mostradas las tareas realizadas en cada una de las iteraciones, que permitieron brindar una solución a la presente investigación.

Además, se mostraron los diagramas utilizados por el equipo de desarrollo para representar las distintas partes del sistema. Por otro lado, el desarrollo basado en pruebas apoyado por la realización de las pruebas de aceptación, demostró ser una estrategia fiable para obtener software probado y funcional.

Conclusiones

La culminación de la presente investigación permitió arribar a las siguientes conclusiones:

- A través de diferentes técnicas de recopilación de información, se identificaron los requisitos de la aplicación en conformidad con el cliente, constatándose la seguridad de poder desarrollar un software usable en su primera versión.
- A partir de los requisitos identificados, según la metodología definida y usando UML como lenguaje de modelado, se realizó el análisis y diseño de la aplicación, permitiendo abarcar a todas las funcionalidades y capacidades con las que el sistema debía cumplir.
- La implementación del sistema se realizó siguiendo el Plan de iteraciones trazado, el cumplimiento estricto del mismo permitió obtener un sistema de software capaz de utilizarse en la sincronización de la información gestionada por el módulo Resultados, cumpliendo con el cronograma inicial de la presente investigación.
- La estrategia de pruebas seguida por el equipo de desarrollo posibilitó obtener un sistema probado desde diferentes puntos de vista. Las pruebas realizadas arrojaron resultados positivos.

Recomendaciones

Al resultado obtenido de la presente investigación se le pueden agregar nuevas funcionalidades que aportarían una experiencia mejor para los usuarios finales. Por eso se recomienda incorporar las siguientes funcionalidades:

- Agregar soporte para realizar una comunicación cifrada con SSL.
- Agregar un visor de logs que permita leerlos desde la misma aplicación y realizar operaciones con los mismos.
- El servidor debe funcionar a nivel de sistema y no a nivel de usuario, como un demonio de GNU\Linux.
- Adicionar soporte para otros sistemas gestores de base de datos.
- En los perfiles de sincronización configurar qué datos específicamente se van a sincronizar.

Glosario

IDE: del inglés *Integrated Development Environment*, en español Entorno de Desarrollo Integrado, es un programa que ofrece una serie de herramientas que facilitan el trabajo de los desarrolladores de software para programar sus programas.

Metodología de desarrollo de software: a grandes rasgos una metodología de desarrollo de software define en un proyecto **quién** debe hacer **qué** cosa, **cuándo** se le dará cumplimiento y **cómo** lo va a hacer.

Framework: es un conjunto de librerías o clases que pueden ser empleadas por los programadores para reutilizar su código en sus programas.

Herramienta CASE: es un software que permite a los desarrolladores modelar parte o todos los componentes de una aplicación. Generalmente a través de diagramas. Las siglas CASE provienen de *Computer Aided Software Engineering*, que en castellano significan Ingeniería de Software asistida por computadora.

Sistema de control de versiones: "*Un sistema de control de versiones es una herramienta […] que mantendrá un seguimiento de los cambios a tus archivos y te ayuda a coordinar el trabajo de diferentes desarrolladores que trabajan sobre diferentes partes de tu sistema al mismo tiempo.*" (Pilon, y otros, 2007 pág. 188)

UML: Del inglés *Unified Modeling Language*, en castellano Lenguaje Unificado de Modelado es una de las herramientas más importantes en el mundo actual del desarrollo de sistemas. Es un lenguaje visual para especificar, construir y documentar los artefactos de los sistemas de software y es aplicado en sistemas orientados a objetos.

Referencias bibliográficas

Beck, Kent y Fowler, Martin. 2000. *Planning Xtreme Programming.* 1ra. Edición. s.l. : Addison Wesley, 2000.

C. Wake, William. 2000. *Extreme Programming Explored.* 2000.

Curtis, Keith. 2009. *After the Software Wars.* Seatle : s.n., 2009.

Henning, Michi y Spruiell, Mark. 2009. *Distributed Programming with Ice.* 2009.

Jacobson, Ivar, Booch, Grady y Rumbaugh, James. 2000. *El Proceso Unificado de Desarrollo de Software.* Madrid : Addison Wesley, 2000.

Kniberg, Henrik. 2007. *Scrum y XP desde las trincheras.* s.l. : C4Media, 2007.

Pilon, Dan y Miles, Russ. 2007. *Head First Software Development.* s.l. : O' Reilly Media, 2007.

Santoro, Nicolás. 2007. *Design and Analysis of Distributed Algorithms.* s.l. : John Wiley & Sons, Inc, 2007.

Stallman, Richard M. 2004. *Software Libre para una Sociedad Libre.* 1ra. Edición en castellanos. s.l. : Traficantes de Sueños, 2004.

Bibliografía

Beck, Kent y Fowler, Martin. 2000. *Planning Xtreme Programming*. 1ra. Edición. s.l.: Addison Wesley, 2000.

Blanchette, Jasmin y Summerfield, Mark, Febrero de 2008. *C++ GUI Programming with Qt 4.* Prentice Hall.2008. ISBN 978-0-13-714397-9

C. Wake, William. 2000. *Extreme Programming Explored*. 2000.

Curtis, Keith. 2009. *After the Software Wars*. Seattle: s.n., 2009.

D. McLaughlin, Brett, Pollice, Gary y West, David, 2007. *Head First Object Oriented Analysis & Design.* O'Reilly Media, Inc. 2007. ISBN-10:0-596-00867.

Ezust, Alan y Ezust, Paul, Agosto de 2006. *An Introduction to Design Patterns in C++ with Qt 4.* Prentice Hall.2006. ISBN-10: 0-13-187905-7

Freeman, Eric y Freeman, Elisabeth, 2008. *Head First Design Patterns.* O'Reilly Media, Inc. 2008.

Henning, Michi y Spruiell, Mark. 2009. *Distributed Programming with Ice.* 2009.

Holzner, Steve, 2006. *Design Patterns for Dummies.* Indianapolis: Wiley Publishing, Inc. 2006

Jacobson, Ivar, Booch, Grady y Rumbaugh, James. 2000. *El Proceso Unificado de Desarrollo de Software*. Madrid : Addison Wesley, 2000.

Kniberg, Henrik. 2007. *Scrum y XP desde las trincheras.* C4Media, 2007.

Molkentin, Daniel, 2007. *The Book of Qt 4: The Art of Building Qt Applications.* San Francisco: No Starch Press, Inc. 2007.

Pilon, Dan y Miles, Russ. 2007. *Head First Software Development.* O' Reilly Media, 2007.

Santoro, Nicolás. 2007. *Design and Analysis of Distributed Algorithms.* John Wiley & Sons, Inc, 2007.

Schmuller, Joseph, 2000. *Aprendiendo UML en 24 horas.* México: Pearson Educación.2000. ISBN: 968-444-463-X .

Stallman, Richard M. 2004. *Software Libre para una Sociedad Libre.* 1ra. Edición en castellanos. Traficantes de Sueños, 2004.

Wells, Don. 2009. *Extreme Programming: A gentle introduction*. [En línea] 28 de septiembre de 2009. [Citado el: 3 de abril de 2010.] http://www.extremeprogramming.org.