Universidad de las Ciencias Informáticas
Facultad # 5

Título: Plugin para la integración de Qt Creator con Subversion.

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

Autora: Amal Saif Fonte.

Tutor: M.Sc. Osvaldo Pereira Barzaga.

Co-tutores: Ing. Rubén Alcolea Núñez**.**
Ing. Luis Guillermo Silva Rojas.

La Habana, junio de 2013

DECLARACIÓN DE AUTORÍA

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

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

_____

Amal Saif Fonte

Autora

_____	_____	_____
M.Sc. Osvaldo Pereira Barzaga	Ing. Luis Guillermo Silva Rojas	Ing. Rubén Alcolea Núñez
Tutor	**Co-tutor**	**Co-tutor**

DATOS DE CONTACTO

Tutor: M.Sc. Osvaldo Pereira Barzaga.
Edad: 28.
Ciudadanía: Cubano.
Institución: Instituto de Cibernética, Matemática y Física (ICIMAF).
Título: Máster en Informática Aplicada.
Categoría Docente: Instructor.
E-mail: opereira@icimaf.cu

M.Sc. Osvaldo Pereira Barzaga. Graduado en el año 2008 con Ing. en Ciencias Informáticas, Universidad de las Ciencias Informáticas (UCI). Con 8 años de experiencias en los temas de visualización médica y procesamiento digital de imágenes desde el 2008-2009 se desempeña como líder del Proyecto Simulador Quirúrgico. En el 2009-2011 es el líder del área temática de visualización científica y líder del proyecto VISMEDIC. En el año 2010 se gradúa de master en la disciplina de Informática Aplicada. Durante el 2012 cumple misión internacionalista en Venezuela, desempeñándose como arquitecto general del software GALBA-CAD, desarrollado por el proyecto, Centro de Diseño y Simulación de Estructuras Mecánicas (CDSEM), en el marco de la empresa mixta entre Petróleos de Venezuela Sociedad Anónima (PDVSA) y Guardián del Alba. Actualmente posees la categoría docente de instructor y trabaja en el grupo de procesamiento de imágenes digitales del Dpto. de Matemática Disciplinaria en el Instituto de Cibernética, Matemática y Física (ICIMAF).

Co-tutor: Ing. Rubén Alcolea Núñez.
Edad: 26.
Ciudadanía: Cubano.
Institución: Universidad de las Ciencias Informáticas (UCI).
Título: Ingeniero en Ciencias Informáticas.
Categoría Docente: Instructor.
E-mail: ralcolea@uci.cu.

Graduado como Ing. en Ciencias Informáticas en la UCI en el año 2011. Tiene cuatro años de experiencias en el procesamiento digital de imágenes y la visualización médica. Actualmente es arquitecto del proyecto Vismedic, perteneciente a la Línea Visualización Científica del centro CEDIN. Es miembro del Colectivo de Entrenadores del Movimiento ACM-ICPC en la UCI.

Co-tutor: Ing. Luis Guillermo Silva Rojas.
Edad: 26.
Ciudadanía: Cubano.
Institución: Universidad de las Ciencias Informáticas (UCI).
Título: Ingeniero en Ciencias Informáticas.
Categoría Docente: Instructor.
E-mail: lgsilva@uci.cu.

Graduado en la UCI como Ingeniero en Ciencias Informáticas en el año 2011. Tiene cuatro años de experiencia en procesamiento de imágenes digitales y visualización de volúmenes. Actualmente es Jefe del Proyecto Vismedic, perteneciente a la Línea Visualización Científica del centro CEDIN. Es miembro del Grupo de Investigación de Visualización y Realidad Virtual (ViViRG) de dicho centro.

AGRADECIMIENTOS

A todas esas personas que hicieron posible la realización de este trabajo, a las que me ayudaron, me apoyaron y soportaron mis momentos de estrés. A todos los que se preocuparon por mí, les agradezco con todo mi corazón.

A mi madre:
Que se han preocupado por el de cursar de mi carrera en todo momento, me ha dado todo su apoyo y confianza.

A mis abuelos:
Me siento muy orgullosa de tenerlos como mis padres. Por quererme y darme tanto cariño, gracias mimi, gracias abuelito, los amo tanto que no alcanzan las palabras para expresarlo.

A mi hermanita:
Que no pude estar cerca de ella en los momentos más difíciles de su adolescencia y que como hermana, era mi deber, pero me encontraba en la escuela y nunca me ha reprochado nada. Te quiero y espero ser ejemplo para ti.

A mi tío Reinaldo:
Que me enseñó a ser una persona responsable y preocupada. Eres mi ejemplo a seguir.

A mi tutor Osvaldo:
Por el apoyo incondicional y por ser antes que tutor, amigo.

A mis Co-tutores Rubén y Guille:
Por estar siempre preocupados por mis resultados y darme apoyo en todo momento. Ustedes son como unos padres en la escuela, siempre dando los mejores consejos. Me han enseñado que con sacrificio y esfuerzo puedo lograr mis objetivos.

A mis amigas:
A mis amigas Yamilka, Madonna, Arianna y Aliané que siempre han estado a mi lado en todos los momentos tanto buenos como malos, siempre me han apoyado y las quiero como si fueran mis hermanitas.

A mis amistades:
A todos los amigos que he hecho en esta universidad, no podría mencionarlos a todos, pero en especial a mis amigas de apartamento, por haberse convertido en parte de mi familia. A mis compañeros de tesis Alina, Adrian y Madonna que siempre nos apoyamos y nos ayudamos unos a otros para cumplir nuestros objetivos. A todos los profes que han contribuido a mi formación, transmitiéndome sus conocimientos.

DEDICATORIA

A mi madre:

A la persona que siempre ha estado a mi lado de una forma u otra dándome apoyo y esperando siempre lo mejor de mí. Quien siempre supo que era capaz de realizar cualquier cosa que me propusiera siempre que lo creyera y pusiera todo el esfuerzo.

A mis abuelos:

A quienes siempre me cuidaron como su hija menor y me dieron mucho cariño y todo lo que necesitaba. Espero que se sientan orgullosos de mí.

A mi tío Reinaldo:

A esa persona que me trató siempre como su hija y quien considero como un padre. Me dio siempre apoyo, confió en mí y hoy espero ser un orgullo para él. Aunque estás lejos siempre pienso en ti.

RESUMEN

Para mejorar el trabajo colaborativo en los grandes proyectos se hace necesario el uso de los sistemas de control de versiones. Los mismos se utilizan ampliamente en los proyectos de desarrollo de software para mantener las versiones del código fuente aunque su aplicación no solo está limitada a esta actividad, sino que permiten gestionar documentos, imágenes y todo tipo de ficheros.

En el presente trabajo se propone la implementación de un *plugin* para Qt Creator, el cual permita conectarse con el sistema de control de versiones Subversion (SVN), para facilitar el trabajo de los desarrolladores y cubrir las deficiencias fundamentales que presenta el *plugin* actual con que cuenta el Qt Creator, haciendo más usable el *plugin* además de hacer cómodo para los desarrolladores implementar desde Qt Creator utilizando Subversion. Con el objetivo de que los desarrolladores no se confundan, se brinda la posibilidad de una representación visual del estado de los ficheros dentro de la copia de trabajo local y así conocer en qué estado se encuentra cada archivo.

Palabras clave: *plugin***,** Qt Creator, Subversion.

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN

Los grandes avances tecnológicos y el auge de la informática hoy en día han permitido la existencia de empresas y compañías dedicadas enteramente a la producción de software, las cuales han utilizado estos conocimientos para el perfeccionamiento de la producción y su desarrollo, convirtiéndose la producción de software en uno de los principales renglones de la economía de muchos países.

Para mejorar el trabajo colaborativo en los grandes proyectos y permitir que varias personas puedan trabajar sobre un mismo proyecto de manera concurrente se hace necesario el uso de los **sistemas de control de versiones.** Los cuales permiten la comunicación más eficiente entre los desarrolladores, ya sea para mantener y recuperar datos de un trabajo, hacer operaciones entre archivos (adicionar, mover, cambiar), como para poder comparar el contenido de un archivo con versiones anteriores. Lo que hace que los sistemas de control de versiones se conviertan en una potente herramienta para el equipo de trabajo.

Cuba ha creado organismos y empresas para solventar la necesidad creciente que existe de desarrollar productos informáticos para la comercialización y demanda nacional o internacional.

Entre las principales entidades productoras de software se encuentra la Universidad de Ciencias Informáticas (UCI). En la facultad 5, específicamente en el Centro de Informática Industrial (CEDIN), se emplea el *Framework* Qt como plataforma de desarrollo para la mayoría de los proyectos productivos. Esto hace de Qt Creator, uno de los entornos de desarrollo integrados (IDE), más apropiado para la implementación de los mismos. Uno de los proyectos que utiliza Qt Creator es Vismedic, el cual tiene como objetivo desarrollar aplicaciones que sirvan de apoyo a los médicos en el proceso de diagnóstico a través de imágenes médicas digitales. Actualmente el proyecto cuenta con un Visualizador 2D de imágenes DICOM y se encuentra en fase de prueba el Visualizador 3D.

Con el objetivo de mejorar el nivel de usabilidad de Qt Creator en el desarrollo de software colaborativo o en comunidades, sus desarrolladores lo han dotado de varios complementos (*plugins*) para la integración con los sistemas de control de versiones más utilizados, por ejemplo: Git, Subversion, Mercurial y CVS. En el caso específico del *plugin* para la conexión con el Subversion (SVN), los desarrolladores del proyecto Vismedic han manifestado el bajo nivel de usabilidad comparado con los que poseen otros IDEs como Visual Studio o Eclipse. Entre sus deficiencias se encuentran:

- No permite una retroalimentación visual del estado de los ficheros dentro de la copia de trabajo local.
- Algunos recursos no se adicionan de manera natural al control de versiones.
- No presenta una herramienta visual para resolver los problemas de conflictos entre diferentes versiones.

Debido a esto se hace necesario el uso de otras herramientas como clientes de control de versiones, tales como: Tortoise, RapidSvn y KDESvn. Por lo que trae como desventaja que se atrase el trabajo de los desarrolladores, que los mismos se les haga un poco difícil implementar ya que deben realizar las operaciones de SVN, mediante otra herramienta externa y el tiempo de desarrollo de un proyecto puede afectarse atrasando la finalización del mismo.

Dada la situación problémica antes abordada se planteada el siguiente **problema de investigación**: ¿Cómo lograr una mejor integración entre el Qt Creator y el Subversion que satisfaga los requerimientos de usabilidad que demandan los desarrolladores de Vismedic?
El **objeto de estudio** de la investigación se define como el sistema de control de versiones Subversion, el **campo de acción** queda enmarcado en el desarrollo de *plugins* para la integración de Qt Creator con el sistema de control de versiones Subversion.
Se define como **objetivo general** implementar un plugin para Qt Creator que permita una mejor integración con el Subversion.
Para dar solución al problema de investigación se definen las siguientes **tareas de investigación:**

1. Elaboración del marco teórico a partir del estado del arte actual referente a los sistemas de control de versiones.
2. Identificación de las deficiencias que presenta el *plugin* actual de Subversion en Qt Creator para los desarrolladores del proyecto Vismedic.
3. Caracterización de la arquitectura de Qt Creator para la integración con el *plugin* propuesto.
4. Diseño del *plugin* para la integración de Subversion con Qt Creator.
5. Implementación del *plugin* propuesto para Qt Creator.
6. Integración con Qt Creator y validación del *plugin* propuesto mediante el correcto funcionamiento de las principales funcionalidades de la aplicación.

Para la realización de la investigación y elaboración del presente trabajo se emplearon varios **métodos científicos de investigación**, entre los cuales se pueden mencionar los siguientes:

Métodos teóricos:
Histórico – Lógico: Método teórico mediante el cual se constatará como ha sido la trayectoria histórica real, la evolución y desarrollo de los aspectos principales del sistema de control de versiones Subversion.
Analítico – Sintético: Este método teórico será utilizado en la investigación para buscar la esencia de lo que existe en el mundo acerca de los sistemas de control de versiones, el *plugin* para la integración de Qt Creator con el sistema de control de versiones Subversion y los rasgos que los caracterizan.

Métodos empíricos:

Consulta de fuentes de información: Método empírico utilizado para la consulta de las fuentes bibliográficas durante la investigación.

La estructura del documento está definida de la siguiente manera:

- **Capítulo 1. Fundamentación teórica:** Se exponen los conceptos fundamentales relacionados con el tema de investigación y se presentan las bases teóricas fundamentales relacionadas con los sistemas de control de versiones, fundamentalmente con el Subversion. Se realiza un análisis sobre las funcionalidades de Subversion y la usabilidad y deficiencias del *plugin* presente en Qt Creator para la integración con el sistema de control de versiones Subversion.

- **Capítulo 2. Solución propuesta:** Se propone una solución teórico-técnica al problema planteado y se explica la manera de crear un *plugin* para Qt Creator, así como los métodos fundamentales de los *plugins*. Se mencionan las metodologías y herramientas de desarrollo empleadas en la realización de este trabajo.

- **Capítulo 3. Características del sistema:** Durante este capítulo se describe el sistema desde la perspectiva de Ingeniería de Software. Se presenta el modelo de dominio del problema. Se realiza la captura de requisitos y el modelo de casos de uso del sistema. Posteriormente se muestra el diagrama de clases del diseño y los diagramas de secuencia correspondientes a los casos de uso.

- **Capítulo 4. Implementación y validación de los resultados:** En este capítulo se abordan los temas relacionados con la implementación del sistema, basándose en el trabajo desarrollado en los capítulos anteriores. Se modelará el diagrama de componentes para la implementación y posteriormente se validan las principales funcionalidades garantizando el correcto funcionamiento del *plugin*.

Glosario de Términos: Se elaboró un Glosario de Términos con el objetivo de facilitar la comprensión del lenguaje utilizado.

Capítulo 1 . Fundamentación teórica

En este capítulo se exponen los conceptos fundamentales relacionados con la integración del sistema de control de versiones Subversion con Qt Creator y se presentan las definiciones fundamentales relacionadas con los sistemas de control de versiones. Se profundiza en los conceptos de Subversion. También se realiza un análisis sobre las funcionalidades de Subversion y la usabilidad y deficiencias del *plugin* presente en Qt Creator para la integración con el sistema de control de versiones Subversion.

1.1 Sistemas de control de versiones

Un sistema de control de versiones o sistema de control de revisiones, es una combinación de tecnologías y prácticas para seguir, gestionar y controlar los cambios realizados en los ficheros del proyecto, en particular en el código fuente, en la documentación y en las páginas web.

El proceso de control de versiones es la gestión de los diversos cambios que se realizan sobre los elementos de algún producto o una configuración del mismo. Los sistemas de control de versiones facilitan la administración de las distintas versiones de cada producto desarrollado, así como las posibles especializaciones realizadas. Ejemplos de este tipo de herramientas son: CVS, Subversion, SourceSafe, ClearCase, Bazaar, Mercurial y Perforce.

Los sistemas de control de versiones son ampliamente utilizados en los proyectos de desarrollo de software para mantener las versiones del código fuente. No obstante, su aplicación no está limitada a esta actividad, sino que permiten gestionar documentos, imágenes y ficheros de numerosos tipo [1].

El proceso de control de versiones puede realizarse de forma manual, pero es muy aconsejable disponer de herramientas que faciliten esta gestión dando lugar a los llamados **sistemas de control de versiones (SCV)**, (del inglés *System Version Control*).

Un SCV funciona de la siguiente manera: Lo primero es crear una copia de trabajo local, para esto es necesario descargarla del servidor. Cada programador realiza los cambios necesarios en el código fuente para la tarea que se le ha encomendado. Cuando los cambios están listos, se envían al servidor, de modo que el resto pueda recibirlos en cualquier momento, y así trabajar sobre dichos cambios cuando tengan que realizar cualquier otra tarea. En caso de que varios programadores trabajen sobre el mismo fichero o ficheros, el sistema lo detectará, y actuará para evitar posibles conflictos. Se pueden dar dos casos:

- Los programadores han trabajado en porciones de código diferentes: No se han solapado las líneas en las que han trabajado, así que es probable que sea suficiente aplicar ambos cambios sobre el fichero. Casi todos los sistemas de control de versiones detectan esta situación y realizan la unión de los cambios de forma automática, notificándolo al usuario para que tenga constancia.
- Los programadores han trabajado en líneas de código comunes, modificando, eliminando o añadiendo líneas en la misma porción de código: En estos casos, el

sistema suele señalar que hay un conflicto entre ambos cambios, y generalmente genera un fichero intermedio convenientemente marcado para que se puedan revisar ambos cambios de forma simultánea, y así quedarse con un cambio u otro, o con una combinación de los dos, realizando la unión manualmente.

Los sistemas de control de versiones se pueden clasificar en dos grandes grupos atendiendo a la arquitectura utilizada para el almacenamiento del código:

Centralizados:

En un sistema de control de versiones centralizado todas las fuentes y sus versiones están almacenados en un único directorio (llamado repositorio de fuentes) de un ordenador (servidor). Todos los desarrolladores que necesiten trabajar con esas fuentes, deben pedirle al sistema de control de versiones una copia local para trabajar. En ella realizan los cambios necesariosy cuando están listos, le ordenan al sistema de control de versiones que guarde las fuentes modificadas como una nueva versión. Algunos ejemplos son CVS y Subversion.

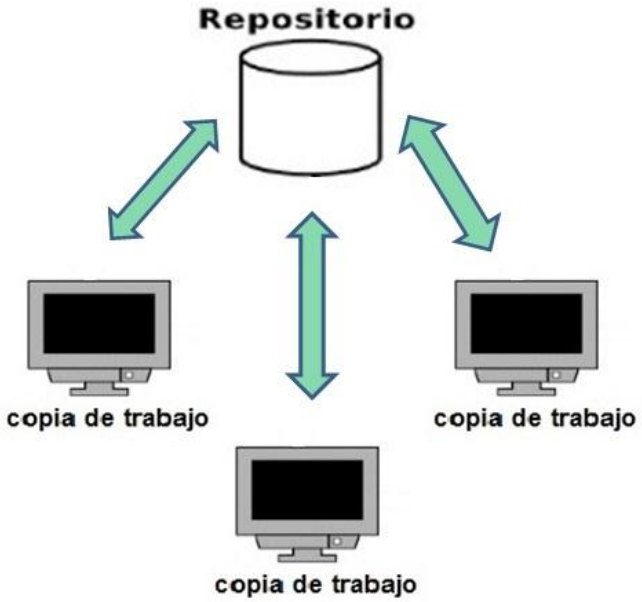

Fig. 1 Centralizado

Distribuidos:

En un sistema de control de versiones distribuido no hay un repositorio central. Todos los desarrolladores tienen su propia copia del repositorio, con todas las versiones y toda la historia. Según van desarrollando, sus fuentes y versiones van siendo distintas. Sin embargo, los sistemas de control de versiones distribuidos permiten que en cualquier momento dos desarrolladores cualesquiera puedan "sincronizar[1]" sus repositorios. Ejemplos: Git y Mercurial.

[1] Hacer coincidir

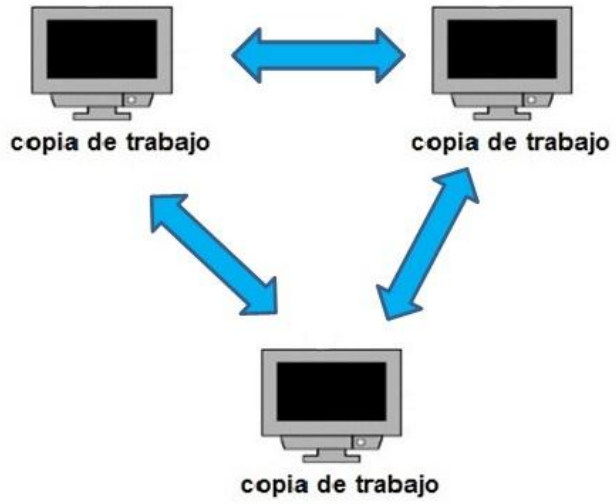

Fig. 2 Distribuido

Aunque no es estrictamente necesario, suele ser muy útil la generación de informes con los cambios introducidos entre dos versiones, informes de estado, marcado con nombre identificativo de la versión de un conjunto de ficheros, etc.

1.1.1 Definiciones

A continuación se exponen los conceptos más importantes en los sistemas de control de versiones. Además de la descripción o la traducción, en la mayoría de los casos se han incluido los nombres en inglés, frecuentemente se emplean de esta forma, tanto en la documentación como en las aplicaciones. La terminología empleada suele variar entre sistemas, a continuación se describen algunos términos de uso común.

Repositorio:
El **repositorio** es el lugar en el que se almacenan los datos actualizados e históricos de cambios, frecuentemente en un servidor. En ocasiones se le denomina depósito o *depot*. Puede ser un sistema de archivos en un disco duro o un banco de datos.

Módulo:
Conjunto de directorios o archivos dentro del repositorio que pertenecen a un proyecto común.

Revisión:
Una **revisión** es cada una de las versiones disponibles en el repositorio. Este término se utiliza refiriéndose tanto a ficheros individuales como a conjuntos de ficheros o incluso al repositorio completo. Suelen identificarse mediante un número.

Etiquetas (*tags*):
La etiqueta permite asignar un nombre fácil de recordar o significativo de manera colectiva a varios ficheros a la vez (por ejemplo, a todos los archivos correspondientes a una misma versión publicada de un programa).

Rama (*branch*)
Es una copia del proyecto, bajo el control de versiones, pero aislado, de forma que los cambios realizados en esta rama no afecten al resto del proyecto y viceversa, excepto cuando los cambios sean deliberadamente "unidos" de un lado al otro. Las ramas también son conocidas como "líneas de desarrollo". Incluso cuando un proyecto no tiene ramas específicas se considera que el desarrollo se está produciendo en la rama principal, también conocida como "línea primaria" o "*trunk*".

Conflicto
Sucede cuando dos o más personas intentan realizar diferentes cambios en la misma porción de código. Los sistemas de control de versiones detectan estos conflictos automáticamente y notifican a los involucrados de que sus cambios entran en conflicto. Es entonces tarea de estas personas resolver el conflicto y comunicar esa resolución al sistema de control de versiones.

Diferencia entre el repositorio y la copia de trabajo:
El **repositorio** es un almacén central de datos donde se guarda la información para el correcto funcionamiento de los SCV centralizados. Esto permite que los archivos subidos al repositorio por un usuario, estén inmediatamente disponibles para los demás usuarios que se conecten a dicho repositorio.
La **copia de trabajo** es un directorio local que contiene una colección de archivos y directorios. En la copia de trabajo se encuentra una copia de los archivos del repositorio. En esta copia de trabajo se pueden crear, borrar o actualizar los archivos existentes, teniendo en cuenta que es un área de trabajo privada.

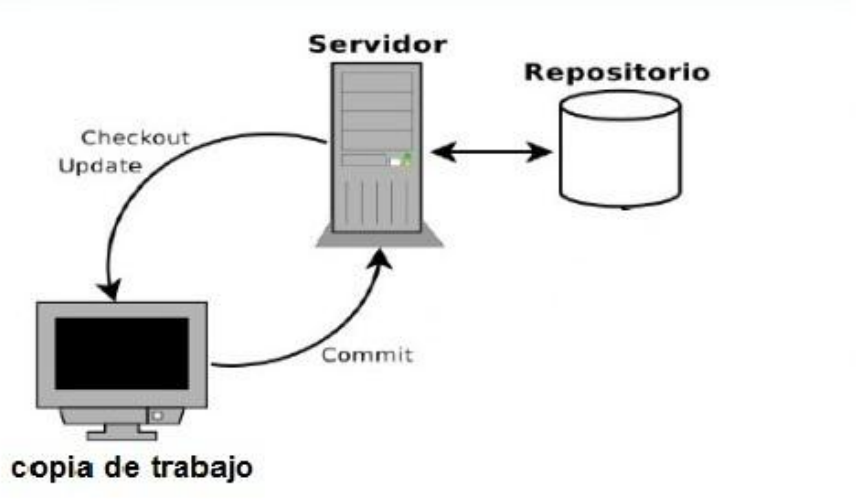

Fig. 3 Diferencia entre repositorio y copia de trabajo local.

Las operaciones más comunes que los clientes pueden realizar en los sistemas de control de versiones son:

- Posibilidad de añadir, borrar, modificar y mover, cada uno de los elementos (ficheros y directorios) que se encuentran bajo el control de versiones.
- "***check-out***": Permite obtener una copia local de trabajo (correspondiente a la última revisión o bien a otra anterior) que puede ser examinada o modificada por el cliente.
- "***check-in***" (*commit*): es la operación mediante la que se integran en el repositorio los cambios realizados en la copia local.
- "***update***" (o "*sync*"): mediante esta opción, se actualiza en la copia local los cambios que otros usuarios han consolidado en el repositorio.

1.1.2 Características

La principal característica de los sistemas de control de versiones es que mantiene la historia de los cambios y modificaciones que se han realizado sobre las versiones a lo largo del desarrollo de un proyecto. De esta forma, el sistema es capaz de acceder a las versiones antiguas de los datos, lo que permite examinar el histórico de cambios o recuperar versiones anteriores de un fichero, incluso aunque haya sido borrado.

Un sistema de control de versiones debe proporcionar:

- Mecanismos de almacenamiento para los elementos a gestionar: documentos de texto, imágenes, archivos binarios.
- Modificaciones de los elementos almacenados, cambios parciales o totales, como renombrar, borrar, añadir o mover elementos.

- Registro histórico de las acciones realizadas con cada elemento o en conjunto, normalmente pudiendo revertir los cambios hacia estados anteriores [2].

1.1.3 Importancia

Los sistemas de control de versiones son importantes en los procesos de desarrollo porque ayuda virtualmente en algunos aspectos al dirigir un proyecto:

- ✓ Comunicación entre los desarrolladores.
- ✓ Manejo de los lanzamientos.
- ✓ Administración de fallos.
- ✓ Estabilidad entre el código.
- ✓ Autorización en los cambios de los desarrolladores.

Se puede trabajar en distintas características de forma simultánea, guardando los cambios en cada una de ellas y uniéndolos al desarrollo principal si ya han sido lo suficientemente probadas, o sencillamente crear una nueva versión para corregir un ***bug*** (error) que se acaba de detectar en producción, sin comprometer lo que ya se ha realizado. Se puede ver quién realizó un determinado cambio, y cuándo lo hizo.

Los sistemas de control de versiones pueden acceder al repositorio a través de redes, lo que es de gran importancia ya que permite ser usado por personas que se encuentran en distintos ordenadores. La capacidad para que varias personas puedan modificar y administrar el mismo conjunto de datos desde sus respectivas ubicaciones fomenta la colaboración.

1.2 Sistemas de control de versiones soportados por Qt Creator

Qt Creator se integra con los siguientes sistemas de control de versiones y de los cuales se estará abordando en el capítulo:

- CVS
- Git
- Mercurial
- Perforce
- Subversion

1.2.1 CVS

CVS es un sistema de control de versiones de código abierto, que ha estado presente desde la década de 1980, y todavía es utilizado por muchas organizaciones de hoy en día. La principal complicación con CVS es que maneja de forma incorrecta los ficheros binarios. CVS está diseñado fundamentalmente para el control de los archivos de texto, y por defecto se asume que cualquier archivo o recurso que se adicione es un archivo de texto.

A diferencia de la mayoría de los sistemas de control de versiones modernos, CVS no tiene soporte para la operación atómica. Por ejemplo, cuando se envían varios archivos (transferirlos al servidor), es posible que la operación se complete sólo para algunos de estos archivos, y no se completará para el resto (debido a los conflictos, por ejemplo). Como regla, es suficiente

corregir la situación y repetir la operación para los archivos restantes (no para todos los archivos). Es decir, los archivos se pueden comprobar en dos pasos. No se observan casos de los daños en el repositorio debido a la ausencia de esta funcionalidad [3].
CVS está limitado al versionado de ficheros, operaciones como copiar y renombrar; las cuales pueden ocurrir sobre ficheros, pero que realmente son cambios al contenido del directorio en el que se encuentran; no son soportadas por CVS. Adicionalmente, en CVS no es posible reemplazar un fichero versionado con algo nuevo que lleve el mismo nombre sin que el nuevo elemento herede el historial del fichero antiguo, que quizás sea completamente distinto al anterior.

Este sistema de control de versiones no aporta nuevas funcionalidades a la solución por lo cual no es utilizado. Además de no controlar de manera correcta los ficheros binarios y no tiene soporte para operaciones atómicas, lo que dificulta a los desarrolladores de Vismedic hacer uso del mismo.

1.2.2 Git

Git es un sistema desarrollado principalmente para Linux y cuenta con altas velocidades para el mismo. Git también viene equipado con una amplia variedad de herramientas para ayudar a los usuarios a analizar el registro de revisiones del desarrollo del proyecto. Algunas de las desventajas que tiene son las siguientes: No es muy usado para los desarrolladores individuales y la compatibilidad limitada de Windows en comparación con Linux.
Algunas de las características de Git:

- Diseñado para manejar proyectos grandes.
- Es extremadamente rápido.
- Autentificación criptográfica del historial.
- Formato de archivo muy sencillo y compacto [4].

Este sistema de control de versiones, no se utiliza en la solución propuesta, pues el mismo es distribuido y las funcionalidades no son iguales que las utilizadas para el *plugin* propuesto que es centralizado.

1.2.3 Mercurial

Mercurial es una herramienta de control de versiones distribuido. Fue hecho originalmente para competir con Git para el desarrollo del kernel de Linux. Es un SCV de código abierto. La curva de aprendizaje es poco pronunciada ya que inicialmente la cantidad de comandos es baja y son parecidos a los de SVN. La documentación para Mercurial también es más completa y facilita el aprendizaje de su funcionamiento. Tiene buen rendimiento en cuanto a la velocidad de ejecución de comandos y operaciones y no muy bueno en la gestión de espacio en disco ya

que siempre Mercurial está estructurado de tal forma que siempre se añaden objetos al repositorio [5].

Para la solución propuesta no se tiene en cuenta este sistema de control de versiones, puesto que es distribuido y el mismo no cuenta con las funcionalidades utilizadas para la solución.

1.2.4 Perforce

Es un sistema de control de versiones centralizado. El servidor gestiona una base de datos central que contiene uno o más repositorios con versiones de los ficheros. Los clientes importan los ficheros del repositorio a su área de trabajo para trabajar en ellos, y posteriormente los devuelven modificados, agrupados en listas de cambio. Algunas de las características de Perforce son las siguientes:

- El cliente hace presentaciones gráficas de diferencias, fusiones y herramientas de reconciliación.
- Se presenta una visión gráfica de históricos y ramificaciones.
- La interfaz administrativa funciona de modo gráfico.
- Soporte para control distribuido de versiones.

Este sistema de control de versiones no aporta nuevas funcionalidades a la solución, por lo cual no es utilizado.

1.2.5 Subversion

Subversion fue escrito ante todo para acceder al control de versiones aproximadamente de la misma manera que CVS lo hace, pero sin los problemas o falta de utilidades que más frecuentemente molestan a los usuarios de CVS. Subversion mantiene las ideas fundamentales de CVS por lo que sus características e interfaces son muy similares a las de este. Este sistema de control de versiones pertenece al movimiento de software libre utiliza una licencia Apache estilo BSD. Fue desarrollado por la empresa CollabNet Inc, es compatible con los sistemas operativos basados en Unix, con Windows y con Mac Os X [6].

Subversion es un sistema de control de versiones en el nivel de archivos que controla las carpetas tanto como los ficheros. Incluye las acciones de copiado, renombrado y eliminado como operaciones de versionado. Es un sistema de control de versiones de código fuente abierto. Maneja ficheros y directorios a través del tiempo. El repositorio es como un servidor de ficheros ordinario, excepto porque recuerda todos los cambios que se realizan a los ficheros y directorios del mismo. Esto le permite recuperar versiones antiguas de sus datos o examinar el historial de cambios de los mismos.

Subversion es un sistema general que puede ser usado para administrar conjuntos de ficheros. Esos ficheros pueden ser código fuente, para otros, cualquier cosa desde la lista de la compra de comestibles hasta combinaciones de video digital y más allá.

Subversion es el sistema de control de versiones utilizado por los desarrolladores del proyecto Vismedic, perteneciente a la Facultad 5, por lo cual a continuación se exponen algunas funcionalidades y características del mismo para profundizar más sobre SVN.

Acción	Descripción	Comando
Crear copias de trabajo	Permite obtener una copia de trabajo de los archivos fuente en su última versión, es un proceso que copia los archivos desde el repositorio dentro del sistema de archivos.	***svn checkout***
Agregar ficheros	Permite agregar los archivos o directorios desde la copia de trabajo.	***svn add***
Eliminar archivos	Permite borrar del repositorio. Si lo que se desea borrar es un fichero, se borrará inmediatamente de la copia de trabajo local. Si es un directorio, este no es borrado, pero Subversion lo programa para borrarlo. Cuando se envíen los cambios, será borrado de la copia de trabajo y del repositorio.	***svn delete***
Deshacer cambios locales	Permite deshacer algún cambio antes de subir una versión.	***svn revert***
Ver diferencia	Permite mostrar la diferencia de un archivo específico de la copia de trabajo local respecto al archivo en el repositorio	***svn diff***
Describir información	Permite descubrir información sobre la historia de un fichero o directorio. El mismo proporcionará un registro de quién hizo cambios a un fichero o directorio, en qué revisión cambió, la hora y fecha de esa revisión, y, si fue proporcionado, el mensaje de registro que acompañaba al envío.	***svn log***
Subir una versión	Permite luego de hacer los cambios a los archivos o directorios locales, enviar estos cambios al repositorio***.***	***svn commit***

Sincronizar con el repositorio	Permite actualizar la copia de trabajo local de un desarrollador y recibir cualquier cambio hecho desde la última actualización de este, por otros desarrolladores en el proyecto.	***svn update***

Tabla 1 Algunas acciones de Subversion.

Las principales características de SVN son:

Versionado de directorios

Subversion implementa un sistema de versionado de ficheros que mantiene la historia de versiones sobre árboles de directorios completos a través del tiempo. Ambos, ficheros y directorios, se encuentran bajo el control de versiones.

Historial de versiones

Con Subversion, se puede añadir, borrar, copiar, y renombrar ficheros y directorios. Y cada fichero nuevo añadido comienza con un historial nuevo, limpio y completamente suyo.

Envíos atómicos

Una colección cualquiera de modificaciones o bien entra por completo al repositorio, o bien no lo hace en absoluto. Esto permite a los desarrolladores construir y enviar los cambios como fragmentos lógicos e impide que ocurran problemas cuando sólo una parte de los cambios se envía con éxito.

Versionado de metadatos

Cada fichero y directorio tiene un conjunto de propiedades (claves y sus valores) asociado a él. Es posible crear y almacenar cualquier par arbitrario de clave/valor que desee. Las propiedades son versionadas a través del tiempo, al igual que el contenido de los ficheros.

Elección de las capas de red

Subversion tiene una noción abstracta del acceso al repositorio, lo que posibilita implementar nuevos mecanismos de red más flexibles para sus usuarios. Subversion puede conectarse al servidor HTTP Apache como un módulo de extensión. Esto proporciona a Subversion una gran ventaja en estabilidad e interoperabilidad y acceso instantáneo a las características existentes que ofrece este servidor entre las cuales se encuentran: autenticación, autorización, compresión de la conexión. También tiene disponible un servidor de Subversion independiente y más ligero.

Manipulación consistente de datos

Subversion expresa las diferencias del fichero usando un algoritmo de diferenciación binario, que funciona idénticamente con ficheros de texto (legibles para humanos) y ficheros binarios (ilegibles para humanos). Ambos tipos de ficheros son almacenados igualmente comprimidos en el repositorio, y las diferencias son transmitidas en ambas direcciones a través de la red.

Ramificación y etiquetado eficientes

El costo de ramificación y etiquetado no necesita ser proporcional al tamaño del proyecto. Subversion crea ramas y etiquetas simplemente copiando el proyecto. De este modo, estas operaciones toman solamente una cantidad de tiempo pequeña y constante.

Hackability

Subversion está implementado como una colección de bibliotecas compartidas en C con APIs bien definidas. Esto hace a Subversion extremadamente fácil de mantener y reutilizable por otras aplicaciones y lenguajes.

1.3 Entorno de desarrollo integrado Qt Creator

Qt Creator es un entorno de desarrollo integrado para la creación de aplicaciones con el *framework* Qt. Qt está diseñado para desarrollar aplicaciones e interfaces de usuario una vez y desplegarlas en múltiples sistemas operativos tanto móviles como de escritorio. Qt Creator proporciona herramientas para desarrollar sus tareas a lo largo de todo el ciclo de vida desarrollo de las aplicaciones.

Las características clave de Qt Creator permiten a los programadores realizar las siguientes tareas:

- Empezar a desarrollar aplicaciones con Qt rápida y fácilmente con el asistente de proyectos, y acceder rápidamente a proyectos recientes y sesiones.
- Diseñar aplicaciones de interfaz de usuario basadas en widgets Qt con el editor integrado, Qt Designer.
- Desarrollar aplicaciones con el editor de código avanzado de C++ que provee nuevas y poderosas características para el completamiento de código y viendo el esquema de archivos (que es, la jerarquía de símbolos de un archivo)
- Compilar, correr y desarrollar proyectos Qt que se dirigen a múltiples plataformas de escritorio y móviles, como Microsoft Windows, Mac OS X, Linux, Symbian, MeeGo, y Maemo.
- Usar herramientas de análisis de código para verificar la administración de memoria en sus aplicaciones.
- Acceder fácilmente a información con el módulo del sistema de ayuda contextual de Qt.

1.3.1 Integración de Qt Creator con SVN

La arquitectura de Qt Creator está diseñada para soportar, de manera natural, la adición de *plugins*, las funcionalidades proporcionadas por Qt Creator se ofrece por los *plugins*. El *plugin* de Qt Creator principal se llama "núcleo" (*Core*).

El núcleo de Qt Creator es básicamente un "cargador de *plugin*". Toda la funcionalidad está implementada en los *plugins* como antes se menciona. Los *plugins* implementa la interfaz ExtensionSystem :: IPlugin.

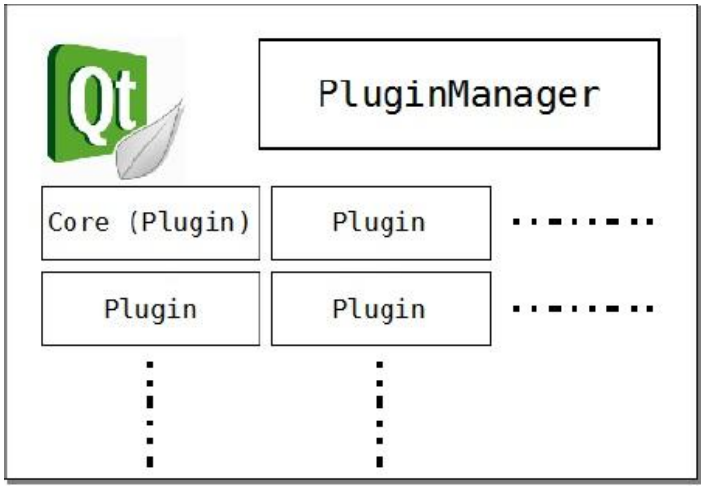

Fig. 4 Aspectos Prácticos de Qt Creator.

En el nivel más fundamental un *plugin* es una biblioteca compartida (DLL en Windows, SO archivo en Linux, el archivo dylib en Mac). Desde el punto de vista de un desarrollador de *plugin* es un módulo que:

1. Implementa la interfaz ExtensionSystem::IPlugin en una clase.
2. Las exportaciones de la clase Plugin utilizando el macro Q_EXPORT_PLUGIN.
3. Proporciona un archivo pluginspec que proporciona información sobre el *plugin* meta.
4. Expone uno o más objetos que puedan ser de algún interés para otros *plugins*.
5. Busca la disponibilidad de uno o varios objetos expuestos por otros *plugins*.

Uno de los componentes clave del "núcleo" es el *ActionManager*. *ActionManager* es responsable del registro de menús, elementos de menú y métodos de teclas abreviados. Así que si se quiere añadir un nuevo elemento de menú o menú, se tendría que utilizar *ActionManager*.

La siguiente figura muestra el ID a utilizar para cada uno de los menús de Qt Creator. Cada uno de los ID se define como *const* variables *char* * estáticas dentro del espacio de nombres de *Core*.

Menu	ID
File	Core::Constants::M_FILE
File -> New	Core::Constants::M_FILE_NEW
File -> Open	Core::Constants::M_FILE_OPEN
File -> Recent Files	Core::Constants::M_FILE_RECENTFILES
Edit	Core::Constants::M_EDIT
Edit -> Advanced	Core::Constants::M_EDIT_ADVANCED
Tools	Core::Constants::M_TOOLS
Window	Core::Constants::M_WINDOW
Window Panes	Core::Constants::M_WINDOW_PANES
Help	Core::Constants::M_HELP

Fig. 5 ID para los menús.

1.4 Usabilidad del *plugin* que integra a Qt Creator con SVN

La Organización Internacional para la Estandarización (ISO) ofrece dos definiciones de usabilidad:
La usabilidad es la capacidad del producto de software de ser entendido, aprendido, usado y atractivo al usuario, cuando es utilizado bajo las condiciones especificadas. (ISO/IEC 9126).

Es la eficacia, eficiencia y satisfacción con la que un producto permite alcanzar objetivos específicos a usuarios específicos en un contexto de uso específico. (ISO/IEC 9241).

Un producto se considera fácil de aprender y de usar en términos del tiempo que toma el usuario para llevar a cabo su objetivo, el número de pasos que tiene que realizar para ello y el éxito que tiene en predecir la acción apropiada para llevar a cabo. Para desarrollar productos usables hay que entender los objetivos del usuario, hay que conocer los trabajos y tareas del usuario que el producto automatiza o modifica.

Las pruebas de usabilidad son una técnica utilizada para evaluar un producto para ello, los usuarios representativos. Su objetivo es identificar los problemas de usabilidad, recoger datos cuantitativos sobre el desempeño de los participantes (por ejemplo, el tiempo de trabajo en las tasas de error), y determinar la satisfacción del participante con el producto [7].
El *plugin* actual con el que cuenta Qt Creator para conectarse con el sistema de control de versiones Subversion presenta algunas deficiencias por lo que para trabajar con los sistemas de control de versiones se emplean otras herramientas como clientes de control de versiones tales como: Tortoise, RapidSvn y KDESvn. Es por eso que se hace necesario un *plugin* que pueda integrar al Qt Creator con el Subversion para hacer uso del mismo y permitir a los

desarrolladores trabajar con el Subversion desde el Qt Creator cuando estén implementando sin necesidad de usar algún cliente. A continuación se describen algunas deficiencias fundamentales presentes en el *plugin* con el que cuenta actualmente el Qt Creator para la integración con el SVN las cuales lo hacen poco usable:

El *plugin* actual no muestra a los desarrolladores en qué estado se encuentran los ficheros y directorios de su copia de trabajo, esto tiende confundir a los desarrolladores ya que no conocen si algún fichero ha sido cambiado. El mismo no presenta una herramienta visual para resolver los problemas de conflictos para diferentes versiones. Si se trabaja desde Qt Creator y ocurre el caso en que dos miembros del equipo modifiquen la misma región de un elemento de configuración determinado, sucede lo que se conoce como conflicto, estos conflictos deben ser solucionados de manera manual. Para poder luego de resolver el conflicto manualmente actualizar el SVN y que no determine que ocurre un conflicto, es decir, que informe que el conflicto fue solucionado se debe hacer desde consola. Esto es una dificultad para los desarrolladores ya que con una herramienta visual para dar solución a este problema no se necesitaría la utilización de comandos en consola y se agilizara el trabajo.

1.5 Consideraciones parciales

Hasta este punto se analizaron los conceptos fundamentales concernientes a los sistemas de control de versiones. Se sentaron las bases necesarias para la comprensión del objeto de estudio y su campo de acción, se mostraron las principales características del Subversion y sus funcionalidades así como una breve descripción de los SCV soportados por Qt Creator. Se abordó sobre usabilidad y se analizaron las principales deficiencias que presenta el *plugin* con que cuenta actualmente el Qt Creator para la integración con el SVN, por lo cual se hace poco usable el mismo.

Se ha llegado a la conclusión que el *plugin* de Qt Creator para la integración con el SVN carece de varias funcionalidades que son necesarias para los desarrolladores, por lo que para darle solución al problema se pretende implementar un *plugin* que cubra esas deficiencias y que a la vez sea fácil de usar para los desarrolladores.

Capítulo 2 . Solución propuesta

En el presente capítulo se propone una solución al problema planteado, partiendo de una explicación detallada de la creación de *plugin* para el Qt Creator desde su código fuente así como se describen algunos métodos importantes para la implementación de *plugins* para Qt Creator. Se mencionan las metodologías y herramientas de desarrollo empleadas en la realización de este trabajo.

2.1 Herramientas, lenguajes, metodología

Los siguientes epígrafes muestran las principales herramientas que asistieron el proceso de creación de los diagramas y la programación de la solución propuesta, los lenguajes que se utilizaron así como la metodología de desarrollo de software empleada durante la realización del presente trabajo.

2.1.1 Herramientas

Para implementar la solución propuesta se utilizaron algunas herramientas que se describirán a continuación para tener una idea más profunda sobre la implementación realizada.

Como herramienta de modelado se empleó Visual Paradigm, creado para asistir el proceso de Ingeniería de Software, este se basa en UML y soporta el ciclo de vida completo del desarrollo de software. Es un producto de probada calidad que soporta varios idiomas y con licencia gratuita y comercial. La herramienta está diseñada para una amplia gama de usuarios interesados en construir sistemas de software fiables con el uso del paradigma orientado a objetos, incluyendo actividades como ingeniería de software, análisis de sistemas y análisis de negocios.

Para el diseño y programación de la aplicación se utilizó el *framework* Qt. Este es un *framework* de alta compatibilidad, tiene características que lo hacen muy versátil y el código C++ que utiliza tiene un alto rendimiento. El código fuente está disponible y así como una muy buena documentación que hace de este una opción muy buena para los desarrolladores. Tiene una arquitectura muy flexible que permite diseñar aplicaciones sin mucho esfuerzo y con una gran calidad. Tiene un apoyo técnico de alta calidad y sigue el principio de reutilizar el código para crear más y hacer despliegues sin importar el lugar.

2.1.2 Lenguajes

Los lenguajes que se utilizaron incluyen el lenguaje principal de desarrollo, y el lenguaje para modelar el análisis y diseño de la aplicación.

El lenguaje de programación seleccionado fue C++. Este es uno de los lenguajes más potentes para desarrollar aplicaciones porque permite programar a un alto nivel y tiene mecanismos tales como la herencia y el polimorfismo que le brindan al desarrollador gran flexibilidad para diseñar. Es un lenguaje orientado a objetos (OO) lo cual permite encapsular los datos y los métodos en clases, esto posibilita obtener un código más seguro y con una mejor organización.

Lenguaje Unificado de Modelado (de inglés, *Unified Modeling Language,* UML) es un lenguaje de modelado visual que se usa para especificar, construir, documentar y visualizar artefactos de un sistema de software. El mismo está compuesto por diversos elementos gráficos que se combinan para conformar diagramas. Su objetivo es visualizar, especificar, construir y documentar los artefactos que se crean durante el proceso de desarrollo. Este brinda además la posibilidad de modelar sistemas utilizando técnicas orientadas a objetos.

2.1.3 Metodología

Se escogió como metodología de desarrollo de software el Proceso Unificado de Desarrollo (RUP). Esta robusta metodología unifica los mejores elementos de las metodologías anteriores y está preparada para guiar el desarrollo de prácticamente todo tipo de proyectos. Utiliza el Lenguaje Unificado de Modelado (UML) para preparar todos los esquemas de un sistema de software. Otra de las ventajas que tiene RUP es que está dirigido por casos de uso, centrado en la arquitectura e iterativo e incremental [8].

RUP establece lo que denomina buenas prácticas como forma de trabajo adecuada para la consecución de objetivos que se pueden ir perfeccionando. Documenta los requisitos y los cambios de los requisitos, analizando el impacto de cambios antes de aceptarlos. Emplea arquitecturas basadas en componentes para maximizar el aprovechamiento de desarrollos previos o componentes preconstruidos.

2.2 *Plugin* para integración de Subversion con Qt Creator

Para agregarle un *plugin* al Qt Creator se debe tener en cuenta que antes de que realice cualquier funcionalidad el mismo debe mostrarse en la lista de *plugin* (*Plugin list*) que aparece en la siguiente figura.

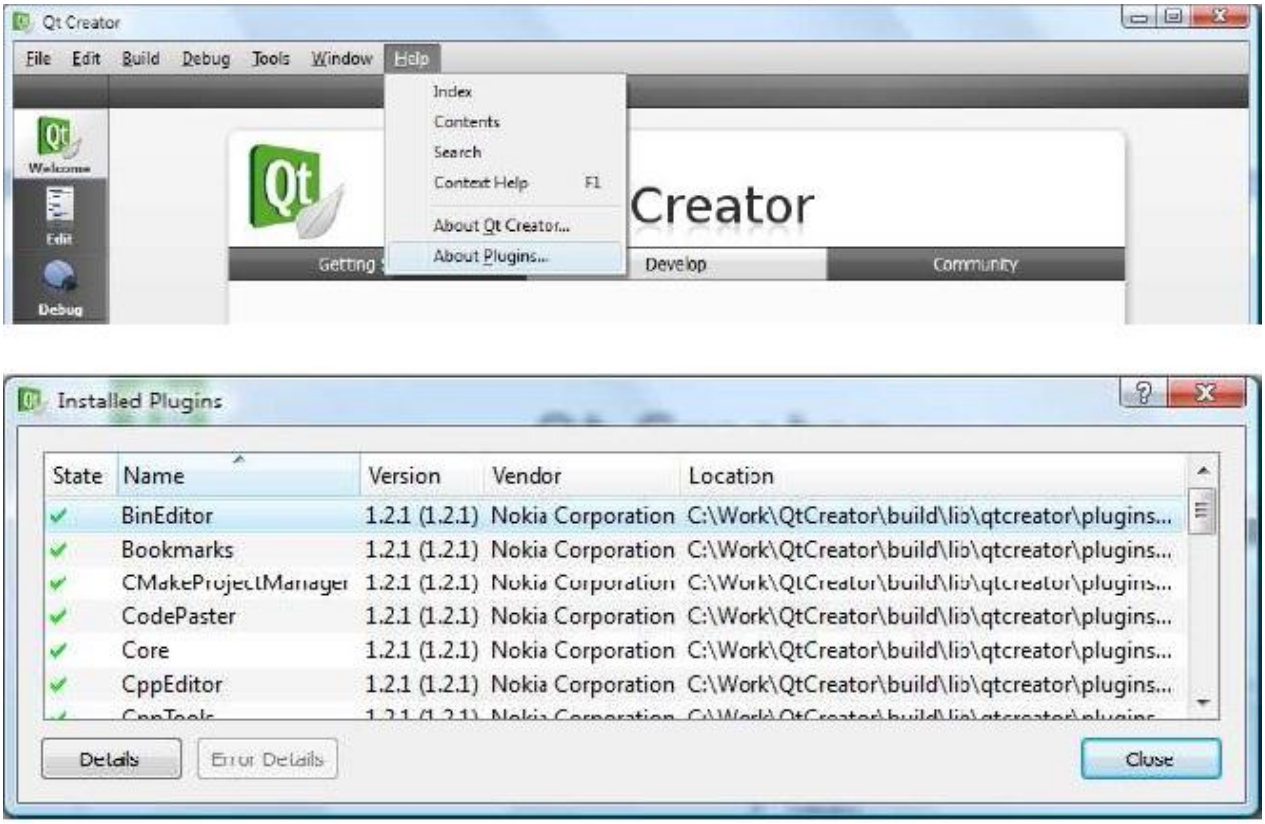

Fig. 6 Lista de *plugin* de Qt Creator.

Para crear el *plugin* entonces en el código fuente de Qt Creator se crea una carpeta en $ $ QT_CREATOR_ROOT / src / plugins. El código fuente del *plugin* se pone en este directorio.

Se debe crear primero el archivo *nombre_plugin.pro* con el siguiente contenido.

```
TEMPLATE = lib
TARGET = nombre_plugin
include(../../qtcreatorplugin.pri)
DESTDIR = $$IDE_PLUGIN_PATH/VCreateLogic
PROVIDER = VCreateLogic
include(../../plugins/coreplugin/coreplugin.pri)
HEADERS += nombre_plugin.h
SOURCES += nombre_plugin.cpp
OTHER_FILES += nombre_plugin.pluginspec
```

En el archivo de proyecto del *plugin* se configuran los siguientes aspectos:

1. Se declara que el *plugin* es una biblioteca. La salida será nombre_plugin.dll
2. El *plugin* se configura para hacer uso de la configuración definida en qtcreatorplugin.pri.

3. El *plugin* se configura para hacer uso de la configuración definida en coreplugin.pri
4. Proporciona información acerca de archivos cabeceras y archivos fuentes que componen el *plugin.*

En el archivo $ $ QT_CREATOR_ROOT / src / plugins / nombre_plugin.pro se deben incluir las siguientes líneas al final del archivo.

SUBDIRS += plugin_ nombre_plugin

plugin_ nombre_plugin.subdir = nombre_plugin

Las líneas anteriores se aseguran de que la próxima vez que se construya Qt Creator, el *plugin* se compila junto con el resto de *plugins* de Qt Creator.

2.2.1 Implementación del *plugin*.

Los *plugins* implementan la interfaz IPlugin. A continuación se describe como el *plugin* implementa esta interfaz.

En $ $ QT_CREATOR_ROOT / src / plugins / nombre_plugin *nombre_plugin.h* introduzca el siguiente código.

```
#ifndef NOMBRE_PLUGIN _H
#define NOMBRE_PLUGIN _H
#include <extensionsystem/iplugin.h>

class nombre_plugin: public ExtensionSystem::IPlugin
{
public:
  nombre_plugin ();
  ~ nombre_plugin ();
  void extensionsInitialized();
  bool initialize(const QStringList & arguments, QString * errorString);
  void shutdown();
};
#endif // NOMBRE_PLUGIN _H
```

Como se puede ver en la clase anterior se implementa la interfaz IPlugin y nada más. Seguidamente se implementan las funciones.

```
#include “nombre_plugin.h"
#include <QtPlugin>
#include <QStringList>
```

```
nombre_plugin :: nombre_plugin ()
{
  // nada
}
nombre_plugin::~ nombre_plugin ()
{
  // nada
}

bool nombre_plugin::initialize(const QStringList& args, QString *errMsg)
{
  Q_UNUSED(args);
  Q_UNUSED(errMsg);
  return true;
}
```

El *plugin* cuenta con varios métodos, entre los cuales se encuentra el método **initialize()** el cual se llama cuando Qt Creator quiere que el *plugin* se inicialice. Esta función es ideal utilizarse para inicializar el estado interno del *plugin* y registrar las acciones u objetos con Qt Creator. La función se invoca después que todas las dependencias de este *plugin* han sido cargadas.

Cada *plugin* debe acompañar un archivo **pluginspec** que proporciona algunos metadatos del *plugin*.

El archivo **pluginspec** proporciona los siguientes campos de información:

1. Nombre del *plugin*, que también se utiliza como el nombre del archivo de la biblioteca que proporciona la implementación de *plugin*.
(Ejemplo: nombre_plugin.dll en Windows, libnombre_plugin.so en Unix)
2. Versión del *plugin*.
3. Versión requerida de Qt Creator.
4. Nombre del proveedor.
5. Derechos de autor.
6. Texto de la licencia.
7. Descripción.
8. URL del proveedor del *plugin*.
9. Lista de Dependencia - proporciona todos los *plugins* que este *plugin* depende. Qt Creator asegura que las dependencias se cargan e inicializan antes de este *plugin*.

El archivo **pluginspec** debe estar en el mismo directorio que el archivo de proyecto del *plugin*.

2.2.2 "Exponer" un objeto de un plugin

Un objeto expuesto es una instancia de QObject (o una de sus subclases) disponible para que otros *plugins* hagan uso del mismo.

Hay tres maneras de exponer un objeto de un *plugin:*

- IPlugin::addAutoReleasedObject(QObject*).
- IPlugin::addObject(QObject*).
- PluginManager::addObject(QObject*).

El IPlugin::addObject () y IPlugin::addAutoReleasedObject () esencialmente invocan al método PluginManager::addObject ().

Los *plugins* pueden exponer a casi cualquier objeto. Normalmente los objetos que proporcionan algún tipo de funcionalidad utilizada por otros *plugins* están expuestos. Las funcionalidades en Qt Creator se definen por medio de las interfaces. A continuación se enumeran algunas interfaces.

• Core :: INavigationWidgetFactory.
• Core :: IEditor.
• Core :: IOptionsPage.
• Core :: IOutputPane.
• Core :: IWizard.

Si un *plugin* tiene objetos que implementan una interfaz, entonces tal objeto tiene que ser expuesto. Por ejemplo, si un *plugin* implementa la interfaz INavigationWidgetFactory en un objeto y lo expuso, el Core utilizará automáticamente el objeto de mostrar el widget que proporciona como widget de navegación. El siguiente fragmento de código ofrece una QTableWidget con un simple widget de navegación a través de una implementación del Core :: INavigationWidgetFactory.

```
#include <coreplugin/inavigationwidgetfactory.h>

class NavWidgetFactory : public Core::INavigationWidgetFactory
{
public:
  NavWidgetFactory();
  ~NavWidgetFactory();
  QString displayName() const;
  virtual int priority() const {return 1;}
  virtual Core::Id id() const {return Core::Id("subversion Vismedic");}
  Core::NavigationView createWidget();
};
```

```
#include <QTableWidget>
NavWidgetFactory::NavWidgetFactory() { }
NavWidgetFactory::~NavWidgetFactory() { }
```

```
Core::NavigationView NavWidgetFactory::createWidget()
{
  Core::NavigationView view;
  view.widget = new QTableWidget(50, 3);
  return view;
}
  QString NavWidgetFactory::displayName()
{
  return "Spreadsheet ";
}
```

En el método initialize() del *plugin* se pondrían las siguientes líneas de código:

```
bool MyPlugin::initialize(const QStringList& args, QString *errMsg)
{
  Q_UNUSED(args);
  Q_UNUSED(errMsg);
  addAutoReleasedObject(new NavWidgetFactory);
  return true;
}
```

La siguiente imagen muestra el efecto del código anterior:

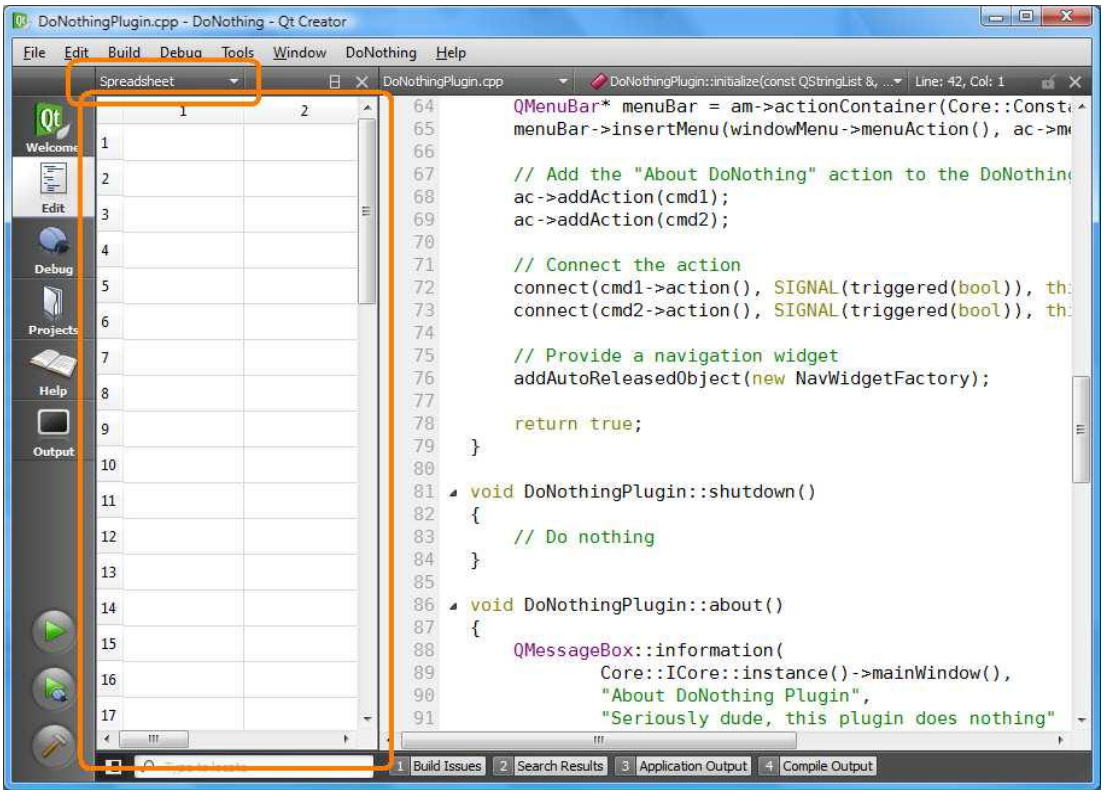

Fig. 7 Vista del *plugin*.

Para realizar nuevos cambios al *plugin* y que al seleccionarlo muestre otro widget que no sea una QTableWidget solo se necesita cambiar el método NavWidgetFactory::createWidget() del código anterior. A continuación un ejemplo:

```
Core::NavigationView NavWidgetFactory::createWidget()
{
    Core::NavigationView view;
    model = new SVNFileSystemModel;

    tree = new QTreeView;
    view.widget = tree;

    ProjectExplorer::ProjectExplorerPlugin* projectExplorer =
     ProjectExplorer::ProjectExplorerPlugin::instance();

    if(projectExplorer)
    {
        ProjectExplorer::Project* currentProject = projectExplorer->currentProject();
        if(currentProject)
        {
            QString dir = currentProject->projectDirectory();
            m_watcher.addPath(dir);
            const QModelIndex index = model->setRootPath(dir);

            connect(&m_watcher, SIGNAL(fileChanged(QString)), this,
              SLOT(fileChanged(QString)));
            tree->setContextMenuPolicy(Qt::CustomContextMenu);
            connect(tree, SIGNAL(customContextMenuRequested(const QPoint &)), this,
              SLOT(contextualMenu(const QPoint &)));
            tree->setModel(model);
            tree->setRootIndex(index);
        }
    }
    return view;
}
```

El resultado del código anterior es el representado en la Fig. 8 donde se muestran los archivos que contiene un proyecto determinado:

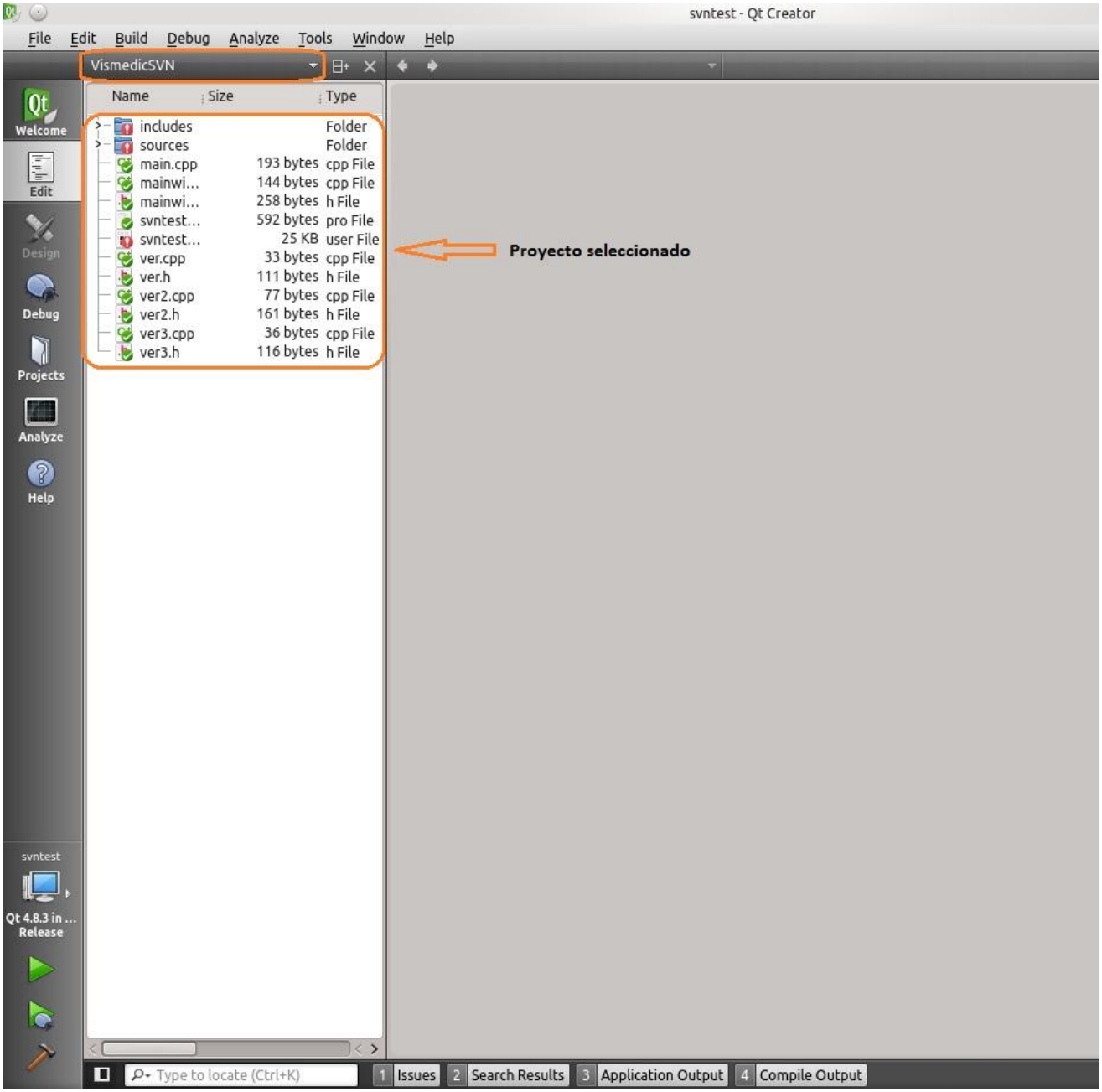

Fig. 8 *Plugin* propuesto.

2.3 Consideraciones parciales

En este capítulo se propone una solución al problema planteado, se describen los pasos necesarios para crearle un *plugin* a Qt Creator así como los métodos genéricos que contienen los *plugins.* También se explica cómo mostrar un widget que se proporciona como widget de navegación a través de una implementación del Core::INavigationWidgetFactory, todo en función de comprender cómo se crea un *plugin* y cómo se puede cambiar según las necesidades del mismo.

Capítulo 3 . Características del sistema

Durante este capítulo se describe el sistema desde la perspectiva de Ingeniería de Software, usando el Proceso Unificado de Desarrollo como metodología. Se presenta el modelo de dominio del problema. Posteriormente se muestran los requisitos funcionales y no funcionales detectados durante la captura de requisitos. Se definen los actores del sistema y se especifica cada caso de uso del sistema. Además se muestra el diagrama de clases del diseño y se modelan diferentes diagramas de secuencia del diseño correspondiente a los caso de uso.

3.1 Modelo de dominio

El Modelo de Dominio se realiza si no se logran determinar los procesos de negocio con fronteras bien definidas, es decir, si no se puede ver claramente quiénes son las personas que realizan cada proceso de negocio, quiénes son los beneficiados con cada uno de estos procesos, pero además quiénes son las personas que desarrollan las actividades en cada uno de estos procesos.

El modelo de dominio contempla la representación de conceptos u objetivos que son importantes dentro de un problema. Estos conceptos representan y simbolizan objetos del mundo real que se encuentran dentro del negocio y ofrecen a su vez un entendimiento del problema en cuestión.

Se puede describir de la siguiente forma: el desarrollador utiliza el Qt Creator para la implementación de los proyectos, el cual contiene el *plugin* de Subversion mediante el cual se solicitan peticiones al SVN y este a su vez envía las respuestas. Debido a las deficiencias que presenta el Plugin SVN, en ocasiones el desarrollador tiene la necesidad de utilizar un cliente SVN para ejecutar funcionalidades que el *plugin* actual para SVN no tiene. En la Fig. 9 se muestra la descripción del negocio mediante un modelo de dominio. A continuación se describen los conceptos que forman parte del modelo de negocio.

El **Desarrollador** es la persona que se encarga de la implementación de los proyectos.

El **Qt Creator** es el entorno de desarrollo integrado que los desarrolladores utilizan para la implementación del proyecto.

El **SVN** permite gestionar y controlar los cambios realizados en el proyecto.

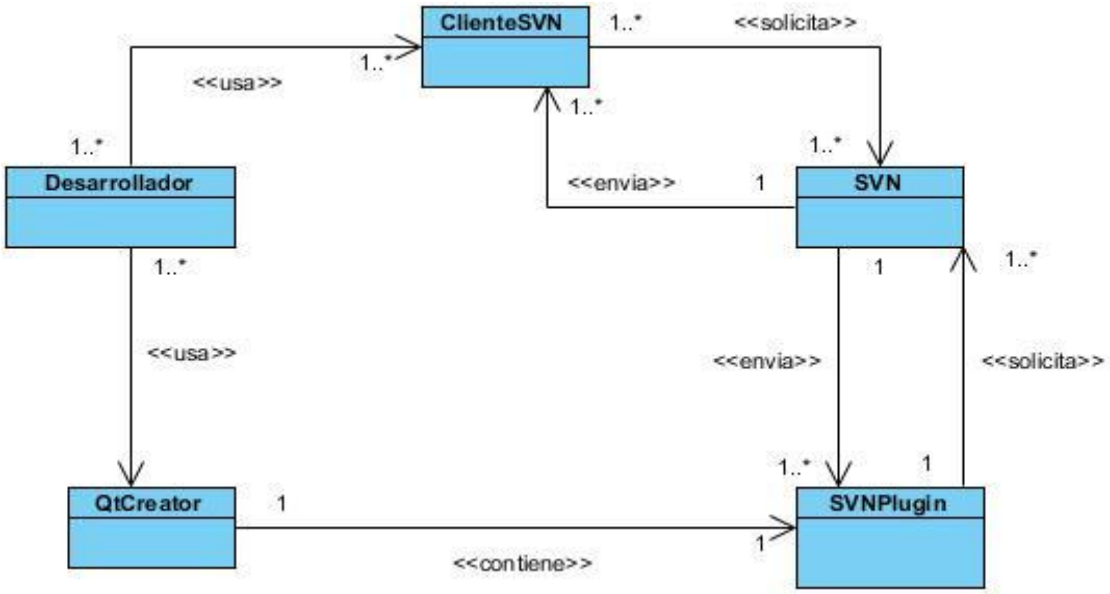

Fig. 9 Modelo de Dominio.

Un **ClienteSVN** es una herramienta que permite a los desarrolladores comunicarse con el SVN.

El **SVNPlugin** es un complemento que se relaciona con el SVN para aportarle funciones al Qt Creator.

3.2 Captura de requisitos

Los requisitos de software son condiciones o capacidades que debe cumplir un sistema para lograr que el mismo funcione adecuadamente. La obtención de los requisitos es de gran importancia debido a que es el hilo conductor de todo desarrollo de software. Una obtención de requisitos con calidad demuestra que el trabajo realizado culminará con éxito. Seguidamente se exponen los requisitos funcionales y no funcionales, por los cuales se rige el desarrollo del sistema.

3.2.1Requisitos funcionales.

Los requerimientos funcionales son capacidades o condiciones que el sistema debe cumplir. Se mantienen invariables sin importar con qué propiedades o cualidades se relacionen.

RF1 Actualizar copia de trabajo (*update*).
RF2 Ignorar fichero (*ignore*).
RF3 Resolver conflicto (*resolved*).
RF4 Ver el estado de la copia de trabajo local (*status*).
RF5 Añadir al repositorio (*add*).
RF6 Borrar del repositorio (*delete*).
RF7 Revertir cambios (*revert*).

RF8 Ver archivos ignorados (*IgnoredFiles*)
RF9 Limpiar copia de trabajo (Cleanup).

3.2.2Requisitos no funcionales

Los requerimientos no funcionales son propiedades o cualidades que el producto debe tener. Debe pensarse en estas propiedades como las características que hacen al producto usable, atractivo, confiable o rápido. Son fundamentales para lograr el éxito del producto y normalmente están vinculados a requerimientos funcionales.

1. **De software.**
   Sistema operativo Ubuntu 12.04 o superior.
   Entorno de desarrollo integrado Qt Creator 2.5.2 o superior.

2. **De Hardware.**
   La memoria RAM mínima será de 512MB.

3. **De Apariencia o Interfaz Gráfica de Usuario.**
   La interfaz gráfica de usuario debe proporcionar, de forma coherente y sencilla, interactividad para todas las funcionalidades de la aplicación. Debe tener una retroalimentación visual del estado de los ficheros dentro de la copia de trabajo local, mostrando con diferentes iconos si los ficheros están actualizados o no.

4. **De Soporte.**
   Se brindará soporte para el sistema operativo Ubuntu 12.04 o superior y para el IDE Qt Creator 2.5.2 o superior.

5. **Restricciones en el Diseño e Implementación.**
   Se empleará como lenguaje de programación C++ y el Qt Creator como entorno de desarrollo integrado.

3.3 Modelo de Casos de Uso

La forma en que los actores usan el sistema es representada a través de los casos de uso. En este epígrafe se identifican los actores del sistema que se desea desarrollar así como los casos de uso. Además se hace la descripción textual de los casos de uso del sistema.

3.3.1 Actores del sistema

Los actores del sistema son entidades externas al sistema que guardan una relación con este y que le demandan una o más funcionalidades. Estos pueden ser personas (usuarios) pero

también los sistemas externos. En este caso particular quien hará uso del *plugin* serán los desarrolladores del proyecto Vismedic.

Actor	Justificación
Desarrollador	Interactúa con el *plugin* durante la ejecución de sus funcionalidades. Añade o elimina ficheros del repositorio, puede ver el estado de algún fichero determinado en la copia de trabajo así como actualizar los ficheros de la copia de trabajo con los cambios del repositorio.

Tabla 2 Actor de sistema.

3.3.2 Diagrama de casos de uso del sistema

El Diagrama de Casos de Uso del Sistema (DCUS) representa gráficamente los casos de uso (CU) y su interacción con los actores.

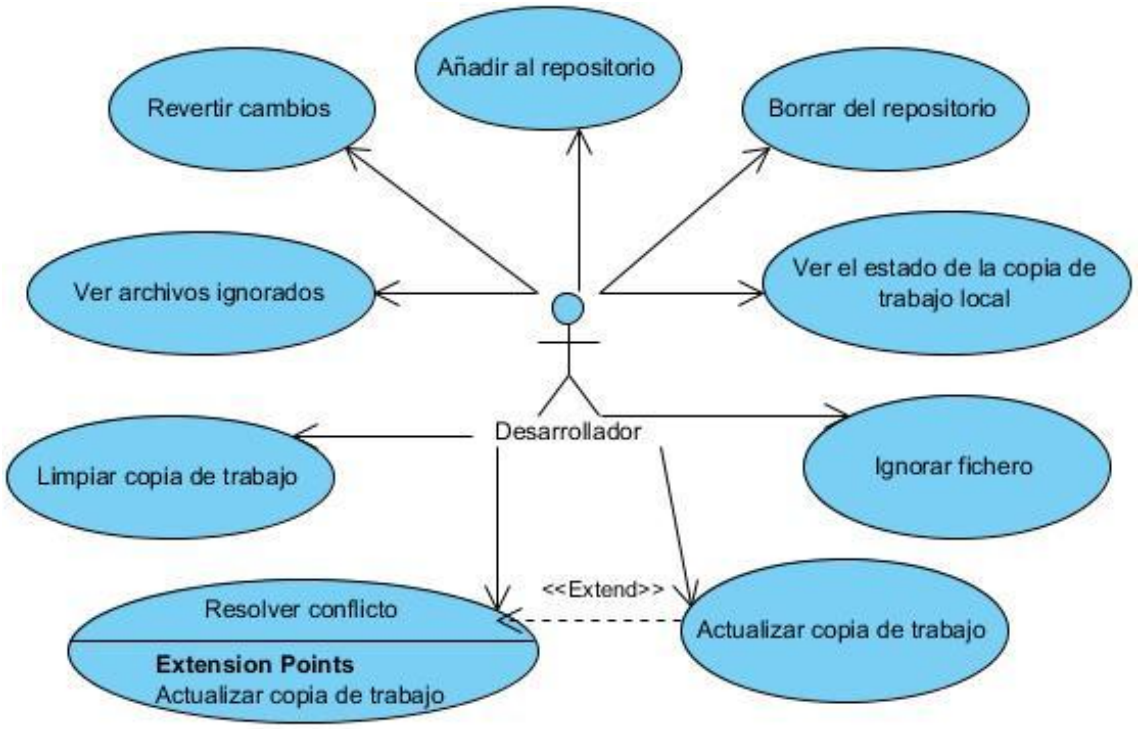

Fig. 10 Diagrama de Caso de Uso del Sistema.

3.3.3 Descripción de los casos de uso del sistema

Cada caso de uso tiene una descripción de las funcionalidades que realiza. A continuación se relacionan las tablas correspondientes a las descripciones de los casos de uso detectados y se argumentan los flujos operacionales de cada uno.

Caso de Uso:	Actualizar copia de trabajo.
Actores:	Desarrollador

Resumen:	Permite al usuario actualizar la copia de trabajo con el repositorio	
Precondiciones:		
Referencias	RF1	
Prioridad	Crítica	
Flujo Normal de Eventos		
Acción del Actor		**Respuesta del Negocio**
1. El desarrollador solicita, actualizar la copia de trabajo.		1.1 El sistema verifica si hay ficheros en conflicto ejecutando el comando *svn update*. 1.2 El sistema actualiza los ficheros de la carpeta seleccionada o el fichero seleccionado de la copia de trabajo agregando los cambios respecto al repositorio.
Flujos Alternos		
Acción del Actor		**Respuesta del Negocio**
1.1		1.1 El sistema muestra los ficheros en conflicto.
Poscondiciones	El sistema actualiza la copia de trabajo con el repositorio agregando los cambios.	

Tabla 3 Descripción del CU Actualizar copia de trabajo.

Caso de Uso:	Ignorar fichero.	
Actores:	Desarrollador	
Resumen:	Permite al usuario ignorar un fichero para que no sea subido al repositorio.	
Precondiciones:		
Referencias	RF2	
Prioridad	Crítica	
Flujo Normal de Eventos		
Acción del Actor		**Respuesta del Negocio**
1. El desarrollador solicita, Ignorar fichero.		1.1 El sistema le da la propiedad de ignorado al fichero.
Poscondiciones	El sistema le da la propiedad al fichero de ignorado para que no sea subido al repositorio.	

Tabla 4 Descripción del CU Ignorar fichero.

Caso de Uso:	Resolver conflicto.	
Actores:	Desarrollador	
Resumen:	Permite al usuario actualizar el SVN para que un conflicto lo detecte como resuelto.	
Precondiciones:	Se detecta que existe un conflicto y el desarrollador le dio solución al mismo manualmente.	
Referencias	RF3	
Prioridad	Crítica	
Flujo Normal de Eventos		
Acción del Actor		**Respuesta del Negocio**
1. El desarrollador solicita, Resolver conflicto.		1.1 Muestra un mensaje que le permite al usuario confirmar si desea resolver o no el conflicto.
2. Confirma que desea resolver el conflicto seleccionando el botón Ok.		2.1 El sistema no considerará por más tiempo que el fichero está en estado de conflicto utilizando el comando *svn resolved.*
Flujos Alternos		
Acción del Actor		**Respuesta del Negocio**
1.1 El desarrollador selecciona el botón Cancel.		1.1 El sistema muestra un mensaje “El conflicto no ha sido resuelto”.
Poscondiciones	El sistema no detecta por más tiempo que el fichero está en conflicto.	

Tabla 5 Descripción del CU Resolver conflicto.

Caso de Uso:	Ver el estado de la copia de trabajo local.	
Actores:	Desarrollador	
Resumen:	Permite al usuario ver el estado que tienen los archivos en la copia de trabajo.	
Precondiciones:		
Referencias	RF4	
Prioridad	Crítica	
Flujo Normal de Eventos		
Acción del Actor		**Respuesta del Negocio**
1. El desarrollador solicita, ver el estado de determinado archivo o directorio de la copia de trabajo local.		1.1 Si selecciona un archivo el sistema muestra archivo con el estado en que se encuentra en la copia de trabajo. 1.2 Si selecciona un directorio el sistema muestra los archivos

	pertenecientes a la carpeta seleccionada con el estado en que se encuentra en la copia de trabajo.
Poscondiciones	El sistema muestra el estado de la copia de trabajo.

Tabla 6 Descripción del CU Ver estado de la copia de trabajo.

Caso de Uso:	Añadir al repositorio
Actores:	Desarrollador
Resumen:	Permite al usuario añadir un archivo o fichero al repositorio.
Precondiciones:	
Referencias	RF5
Prioridad	Crítica
Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Negocio**
1. El desarrollador solicita, añadir al repositorio.	1.1 El sistema programa para que sea añadido el directorio o fichero al repositorio.
Poscondiciones	El sistema añade al repositorio.

Tabla 7 Descripción del CU Añadir al repositorio.

Caso de Uso:	Borrar del repositorio
Actores:	Desarrollador
Resumen:	Permite al usuario borrar un archivo o fichero al repositorio.
Precondiciones:	
Referencias	RF6
Prioridad	Crítica
Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Negocio**
1. El desarrollador solicita, borrar del repositorio.	1.1 Si es un directorio el sistema programa para que sea borrado, si es un fichero lo borra del repositorio.
Poscondiciones	El sistema borra del repositorio.

Tabla 8 Descripción del CU Borrar del repositorio.

Caso de Uso:	Revertir cambios
Actores:	Desarrollador
Resumen:	Permite al usuario revertir los cambios que se ha hecho en la copia de trabajo.

Precondiciones:	
Referencias	RF7
Prioridad	Crítica
Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Negocio**
1. El desarrollador solicita, revertir cambios.	1.1 El sistema elimina los cambios del fichero seleccionado dejándolo como se encuentra en el repositorio.
Poscondiciones	El sistema elimina los cambios.

Tabla 9 Descripción del CU Revertir cambios.

Caso de Uso:	Ver archivos ignorados
Actores:	Desarrollador
Resumen:	Permite al usuario ver el estado de los archivos incluyendo los ignorados.
Precondiciones:	
Referencias	RF8
Prioridad	Crítica
Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Negocio**
1. El desarrollador solicita, ver archivos ignorados.	1.1 Si selecciona un archivo el sistema muestra los ficheros que pertenecen a la carpeta donde se encuentra el archivo seleccionado con el estado en que se encuentra en la copia de trabajo incluyendo los ignorados. 1.2 Si selecciona un directorio el sistema muestra los archivos pertenecientes a la carpeta seleccionada con el estado en que se encuentra en la copia de trabajo incluyendo los ignorados.
Poscondiciones	El sistema muestra los ficheros con su estado incluyendo los ignorados.

Tabla 10 Descripción del CU Ver archivos ignorados.

Caso de Uso:	Limpiar copia de trabajo
Actores:	Desarrollador

Resumen:	Permite al usuario limpiar la copia de trabajo, eliminando bloqueos en el proceso.
Precondiciones:	
Referencias	RF8
Prioridad	Crítica
Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Negocio**
1. El desarrollador solicita, limpiar la copia de trabajo.	1.1 Si selecciona un archivo el sistema desbloquea los archivos bloqueados pertenecientes a la carpeta donde se encuentra el archivo seleccionado. 1.2 Si selecciona un directorio el sistema desbloquea los archivos bloqueados pertenecientes a la carpeta seleccionada.
Poscondiciones	El sistema elimina los bloqueos de la copia de trabajo.

Tabla 11 Descripción del CU Limpiar copia de trabajo.

3.4 Diseño del sistema

Los diagramas de clases de diseño muestran las clases con sus atributos y métodos y la forma en que se relacionan entre sí. En el flujo de trabajo de Diseño también se realizan los diagramas de interacción que muestran la comunicación y las relaciones entre los objetos.

En la Fig. 11 se muestra el Diagrama de Paquetes general del sistema. Para obtener el sistema se ha hecho necesario el uso de otros módulos, por lo que se ha dividido en tres paquetes fundamentales: QtCreator, Subversion y Qt, el presente trabajo se centra en el paquete de Subversion2.

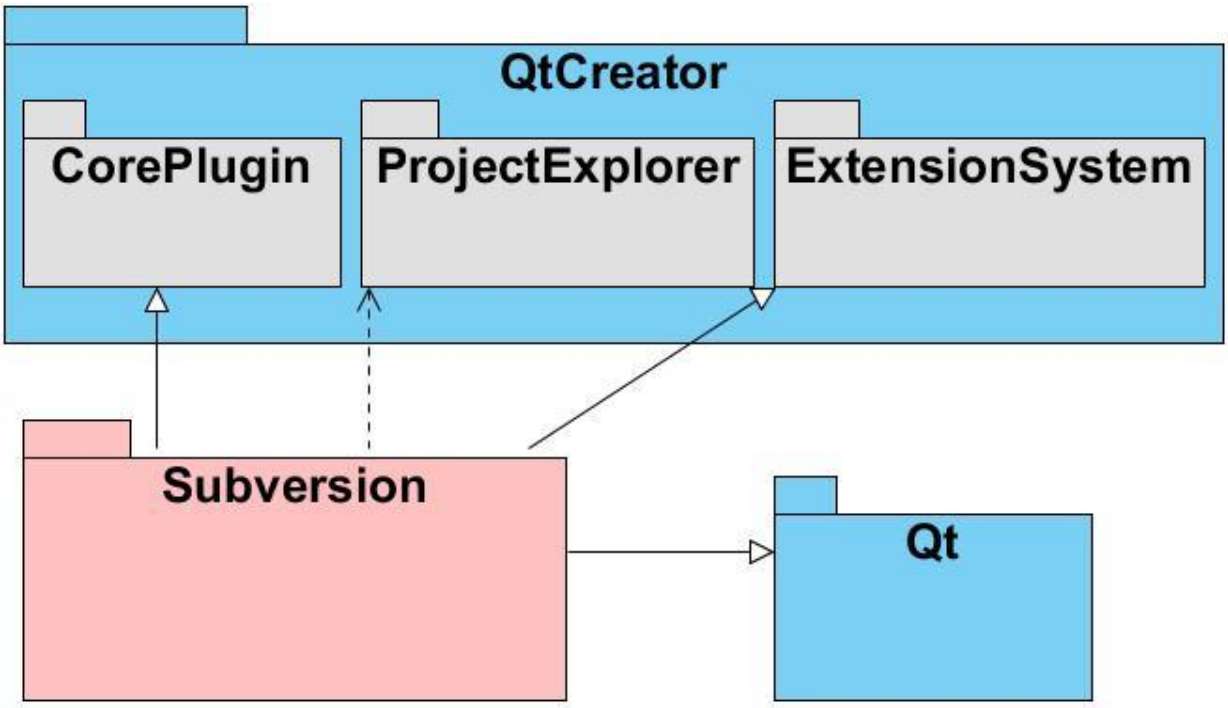

Fig. 11 Diagrama de Paquetes del Sistema.

3.4.1Diagrama de clases del diseño

En la Fig. 12 se muestra el diagrama de clases del diseño correspondiente al sistema desarrollado, nótese que el *plugin* se inicializa en la clase Subversion2plugin mientras que las funcionalidades del Subversion se encuentran implementadas en la clase NavWidgetFactory. El mismo utiliza clases propias del Qt Creator y del *framework* Qt por lo cual los paquetes se relacionan.

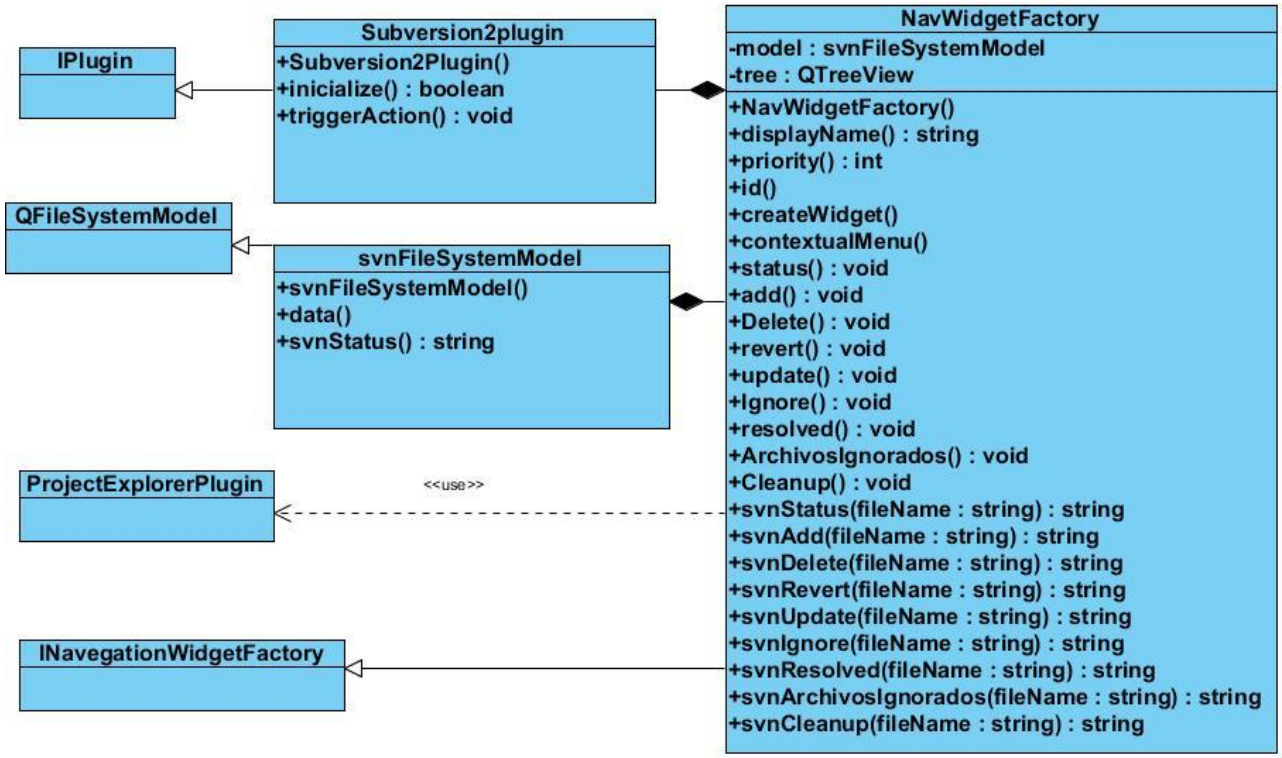

Fig. 12 Diagrama de clases del diseño.

3.4.2Diagrama de secuencias del diseño

A continuación se muestran los diagramas de secuencias empleados en el diseño del sistema.

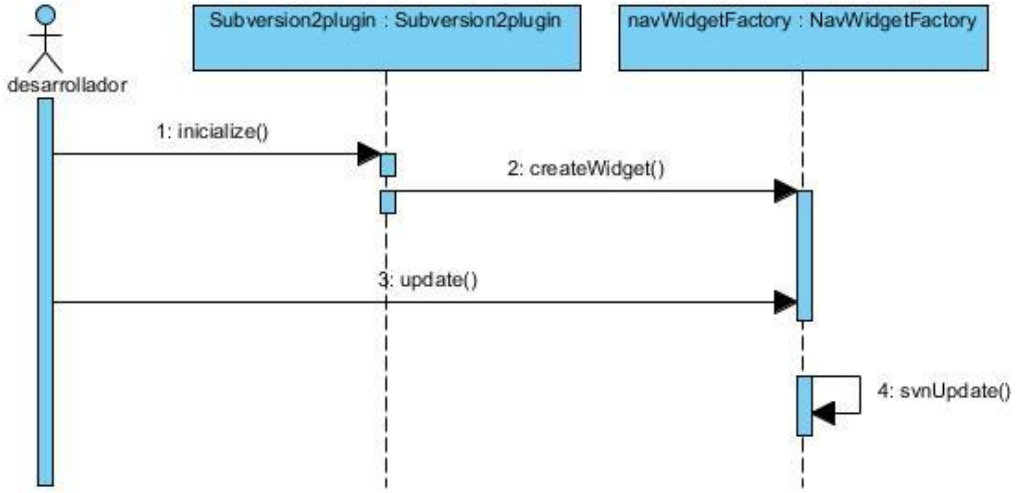

Fig. 13 Diagrama de secuencia del caso de uso actualizar copia de trabajo.

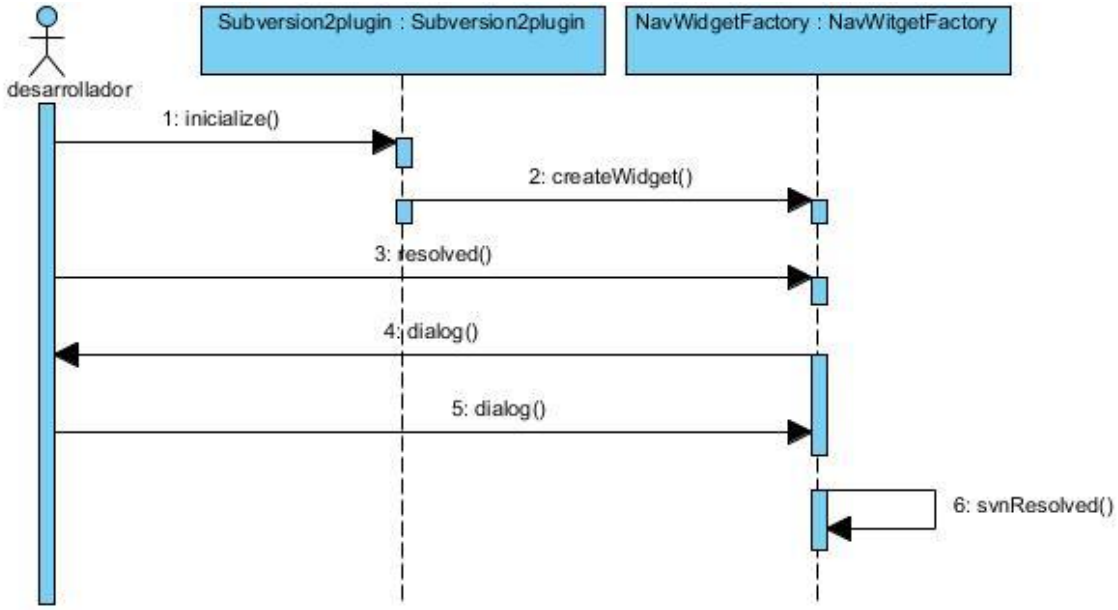

Fig. 14 Diagrama de secuencia del caso de uso resolver conflicto.

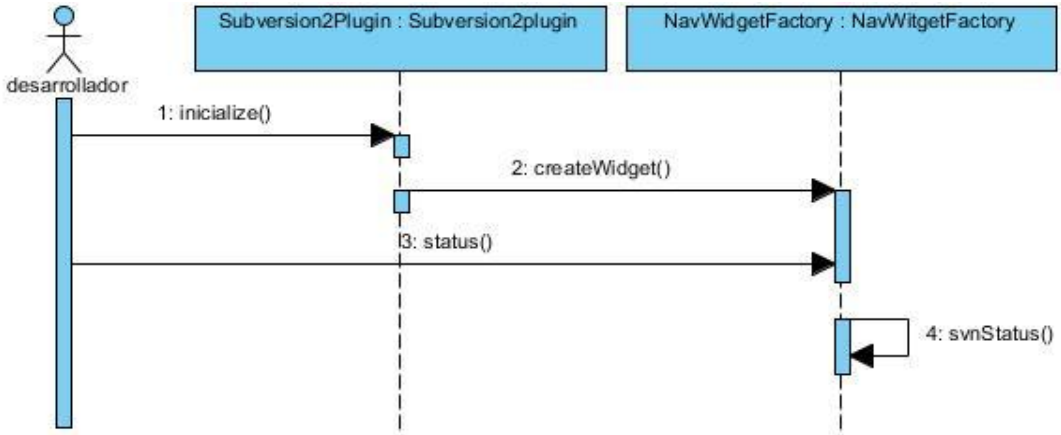

Fig. 15 Diagrama de secuencia del caso de uso ver el estado de la copia de trabajo local.

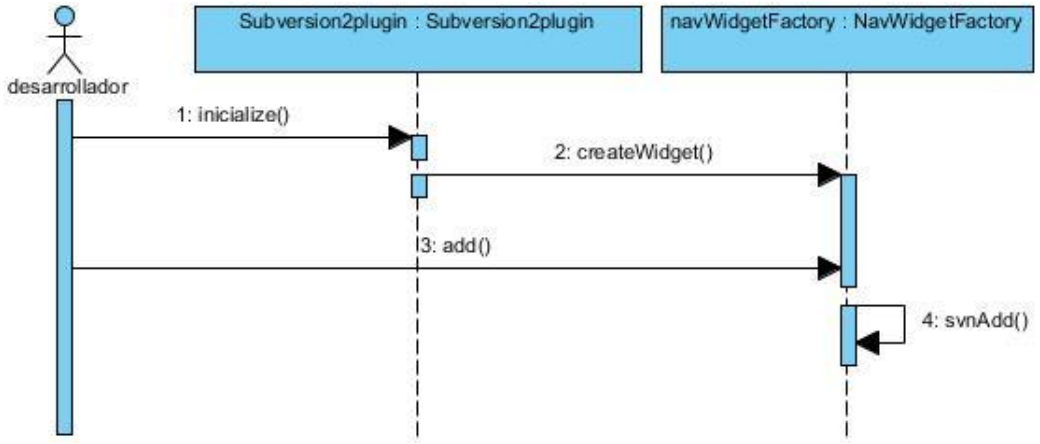

Fig. 16 Diagrama de secuencia del caso de uso añadir al repositorio.

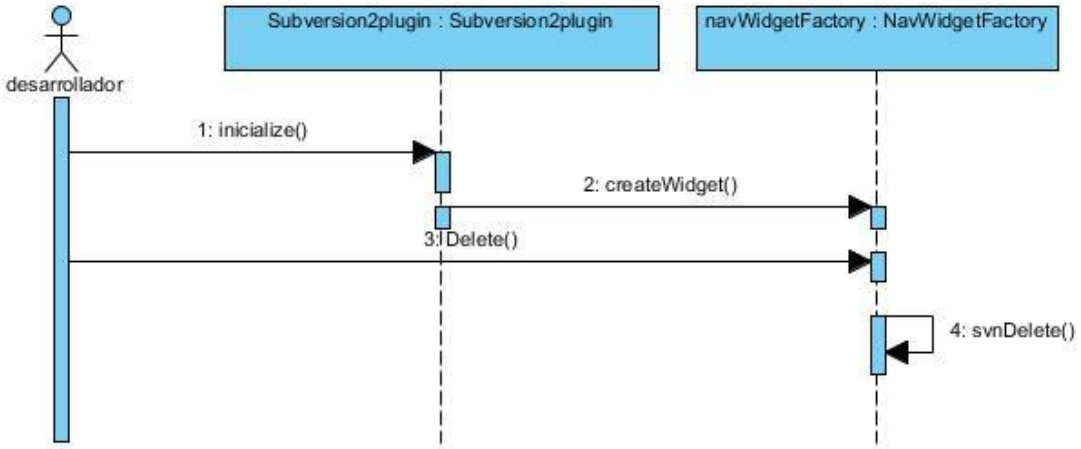

Fig. 17 Diagrama de secuencia del caso de uso borrar del repositorio.

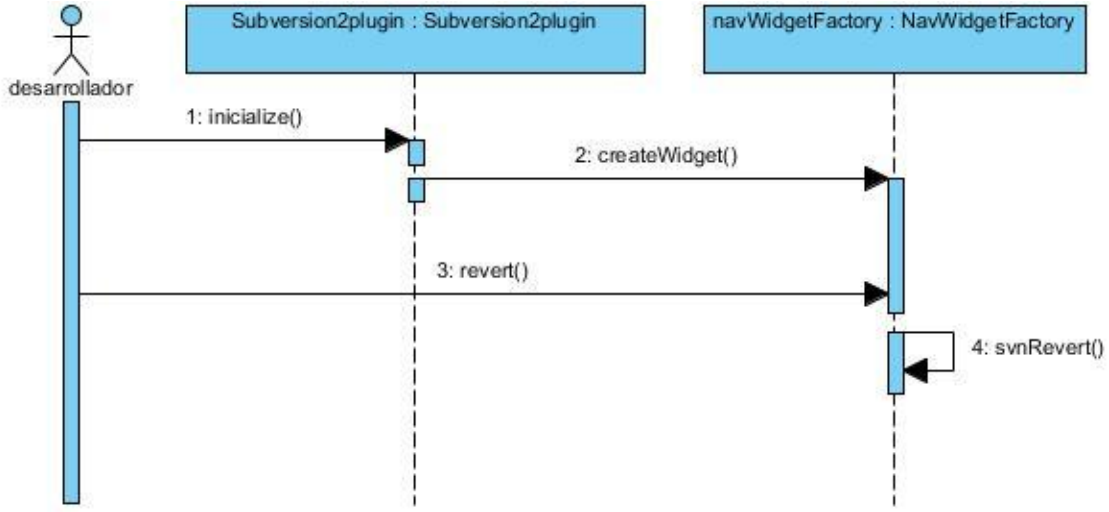

Fig. 18 Diagrama de secuencia del caso de uso revertir cambios.

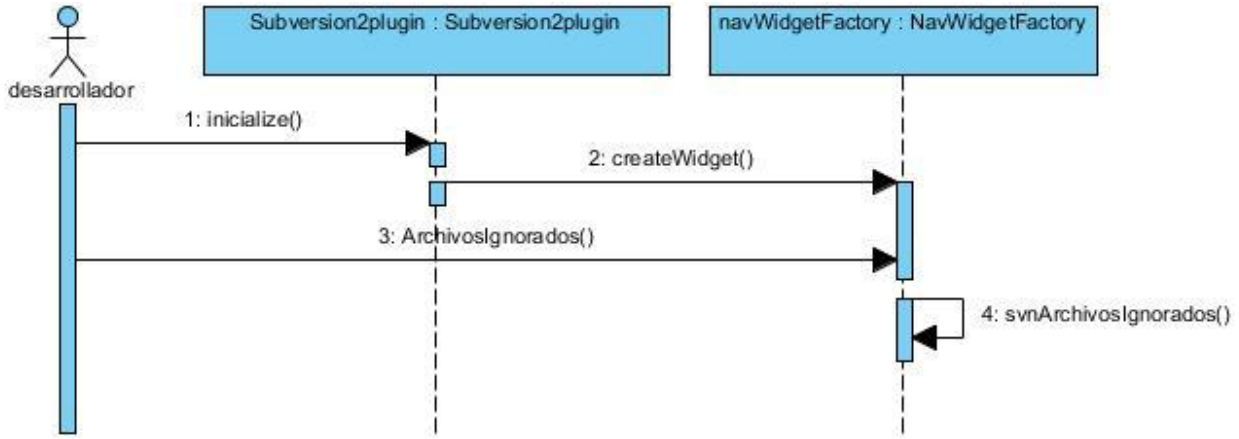

Fig. 19 Diagrama de secuencia del caso de uso ver archivos ignorados.

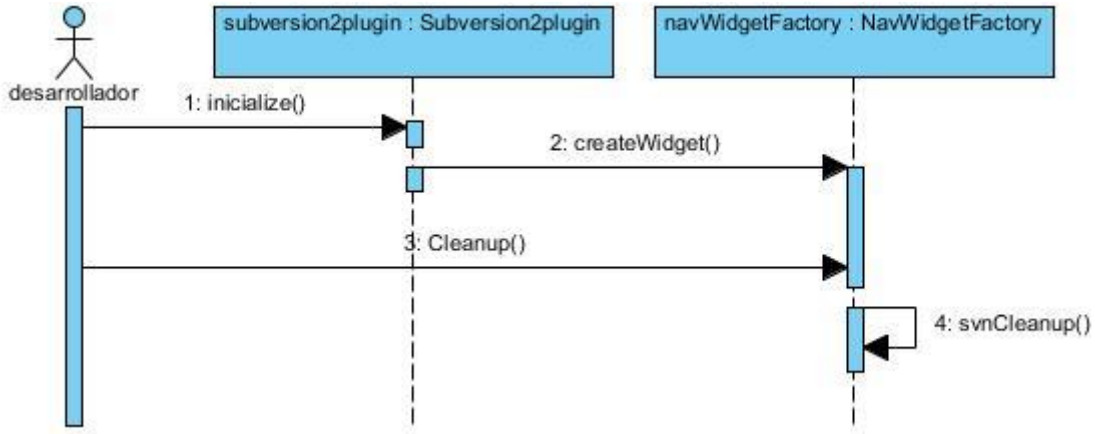

Fig. 20 Diagrama de secuencia del caso de uso limpiar copia de trabajo.

Capítulo 4 . Implementación y validación de los resultados

En este capítulo se abordan los temas relacionados con la implementación del sistema, basándose en el trabajo desarrollado en los capítulos anteriores. Esta sección centra su contenido en el diagrama de componente del sistema desarrollado. Posteriormente se valida la solución propuesta mediante el correcto funcionamiento de las principales funcionalidades de la aplicación.

4.1 Implementación

El modelo de implementación describe como los elementos del modelo de diseño y las clases, se implementan en términos de componentes, ficheros de código fuente, ejecutables, entre otros. Como resultado se obtiene un sistema ejecutable que incluye todas las funcionalidades propuestas en la captura de requisitos funcionales

4.1.1Diagrama de componentes

Cada componente define una interfaz que describe su funcionalidad y forma de empleo. El diagrama de componentes, permite conocer a los desarrolladores y clientes la estructura física que tiene el sistema y cómo se relacionan sus partes. A continuación se presenta el diagrama de componentes.

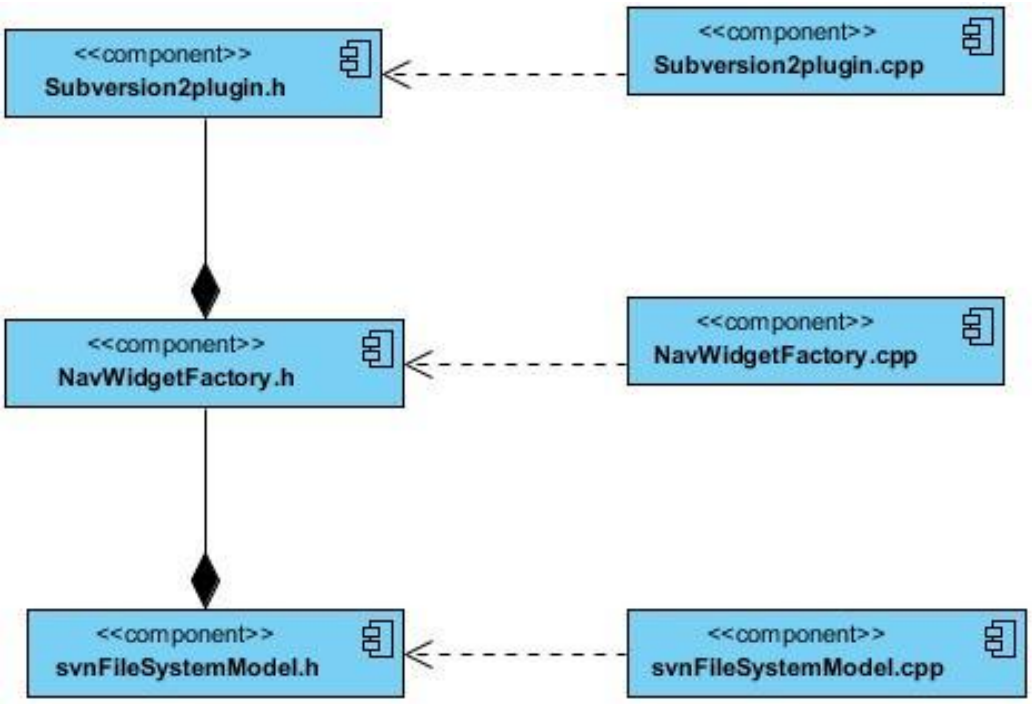

Fig. 21 Diagrama de componentes.

4.2 Validación de las funcionalidades

Esta sección muestra los resultados de las principales funcionalidades a tener en cuenta para medir el correcto funcionamiento del *plugin*, probando que funcionen correctamente las funcionalidades agregadas con las cuales no cuenta el *plugin* que tiene actualmente Qt Creator para la conexión con SVN. Todas las pruebas se realizaron sobre el mismo proyecto, utilizando como cliente externo el TortoiseSVN para probar las nuevas funcionalidades agregadas y se compara con el *plugin* actual de Qt Creator para las funcionalidades que son comunes pero que se adicionan al plugin propuesto para hacer uso de las mismas de manera rápida y fácil para el desarrollador por la nueva filosofía que aporta el *plugin* propuesto.

4.2.1 Funcionalidades a medir

Con el objetivo de que se haga uso del SVN desde Qt Creator, se tendrán en cuenta fundamentalmente que se represente de manera visual el estado de los ficheros dentro de la copia de trabajo y que el plugin resuelva el conflicto, aunque también se prueban las demás funcionalidades para su correcto funcionamiento.

4.2.2 Representación visual

Las siguientes imágenes muestran como queda representado de manera visual estado de los ficheros dentro de la copia de trabajo local para el caso que se existan ficheros modificados, utilizando el TortoiseSVN y el *plugin* propuesto.

En caso de archivos fuera de carpetas:

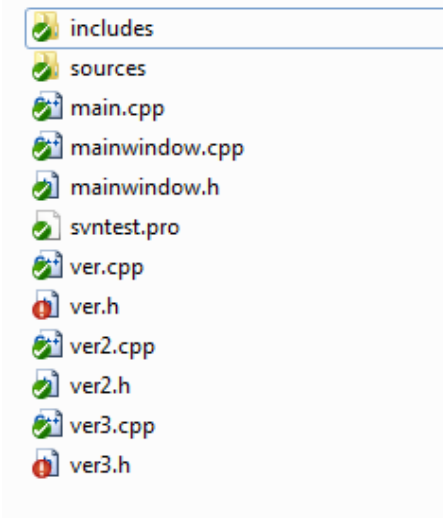

Fig. 22 Utilizando TortoiseSVN.

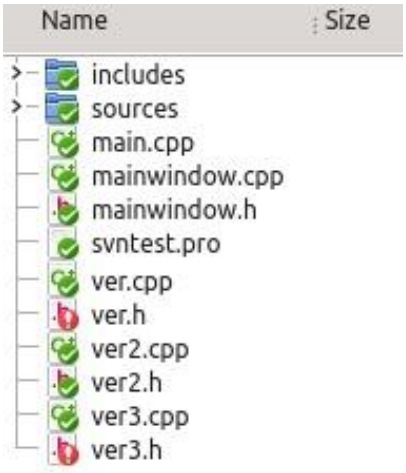

Fig. 23 Utilizando el *plugin* propuesto.

En caso de modificación de archivos dentro de carpetas:

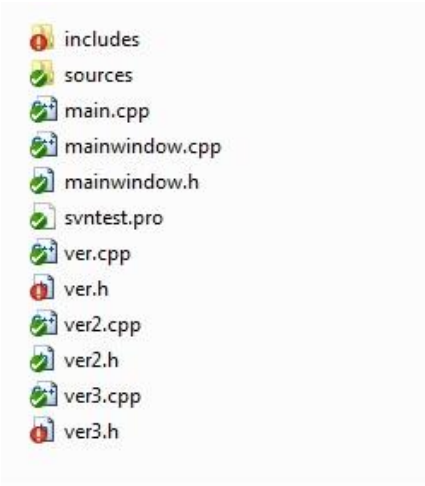

Fig. 24 Utilizando TortoiseSVN.

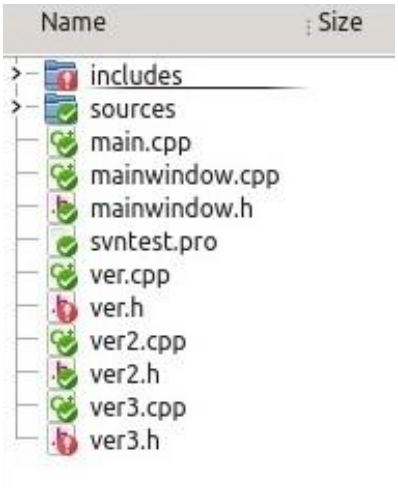

Fig. 25 Utilizando el *plugin* propuesto.

Puede notarse que en caso de que un archivo dentro de una carpeta se encuentra modificado entonces la carpeta al cual pertenece, también muestra el ícono que señala el estado como modificado (ícono en rojo). Para ambos casos los resultados son iguales por lo que el *plugin* propuesto resuelve el problema de la representación visual.

4.2.3 Resolver conflicto

Las siguientes imágenes reflejan la forma en que queda resuelto el conflicto tanto desde TortoiseSVN como desde el *plugin* propuesto, dándose una breve explicación y mostrando los pasos con imágenes.

En la Fig. 25 queda reflejados los siguientes aspectos concernientes al conflicto:

- Para cada fichero en conflicto, Subversion coloca tres ficheros extra en su copia de trabajo local.

- Subversion coloca marcas de conflicto (secuencias especiales de texto que delimitan los “lados” del conflicto) en el fichero para demostrar visualmente las áreas solapadas.

- Subversion imprime una C durante la actualización, y recuerda que el fichero está en un estado de conflicto.

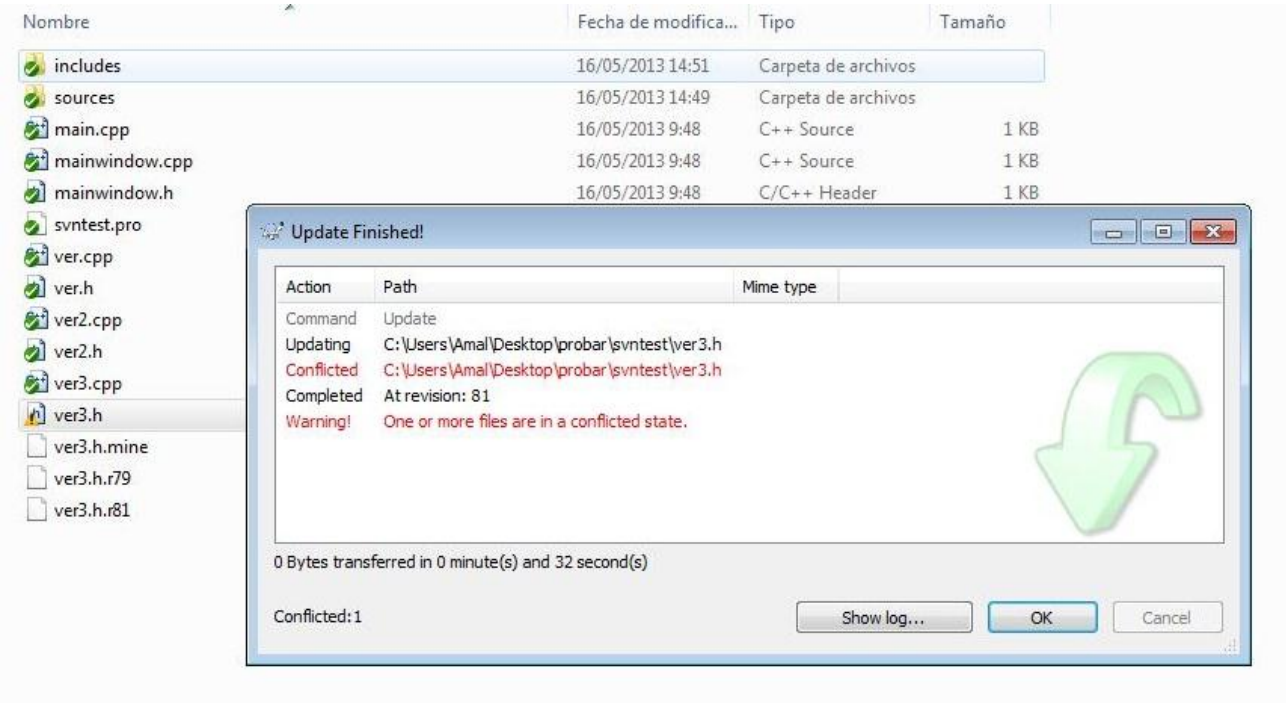

Fig. 26 Conflicto desde TortoiseSVN.

En la Fig. 27 se refleja lo anterior mostrado pero desde el *plugin* propuesto, donde igual que en el anterior se genera los ficheros extras para el archivo en conflicto y Subversion imprime en los mensajes, una C durante la actualización recordando que existe un conflicto.

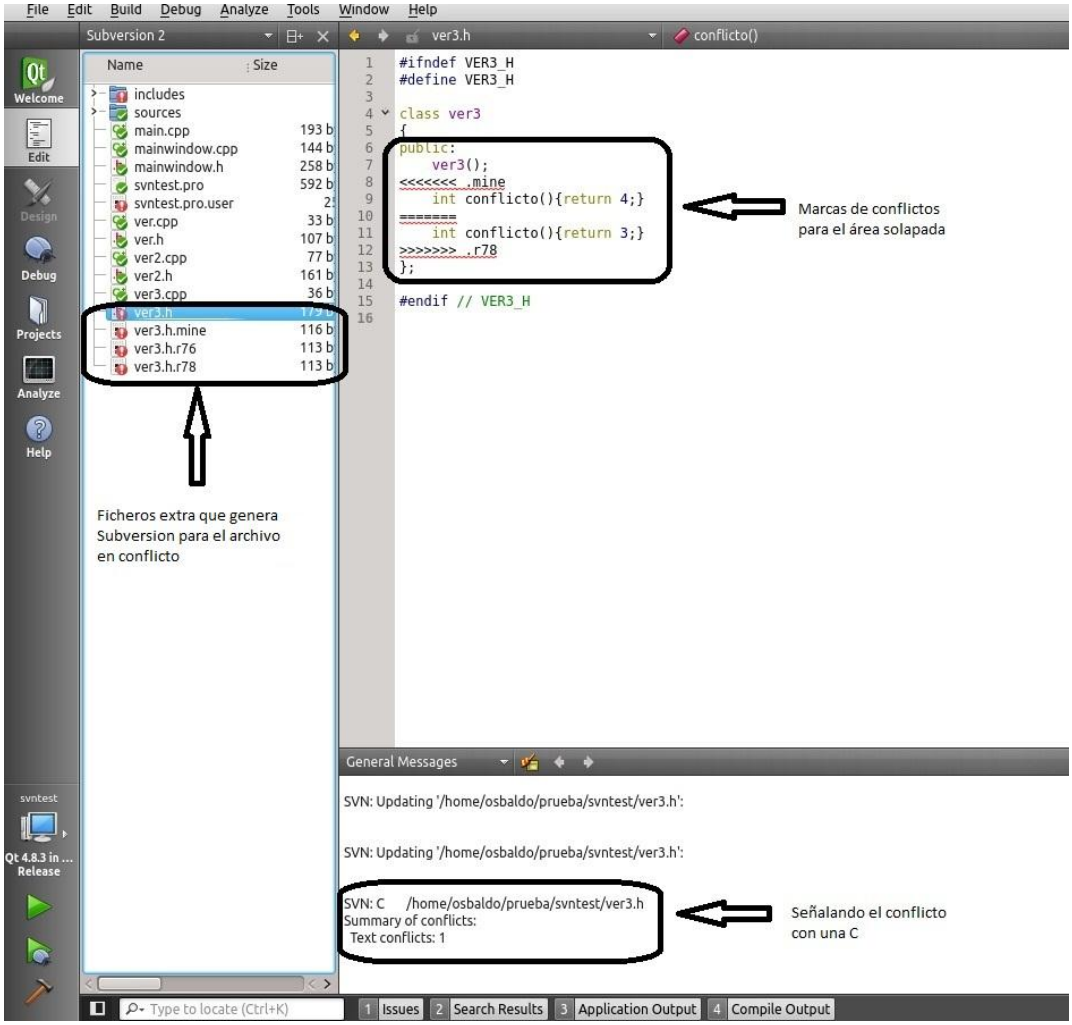

Fig. 27 Conflicto desde el *plugin* propuesto.

Luego de que se detecte un conflicto es tarea del desarrollador darle solución al mismo, el cual necesita hacer una de tres cosas:

- Fusionar el texto en conflicto “a mano” (examinando y editando las marcas de conflicto dentro del fichero).
- Copiar uno de los ficheros temporales sobre su fichero de trabajo.
- Ejecutar **svn revert** <**filename**> para eliminar todos sus cambios locales.

Una vez que se haya resuelto el conflicto de forma manual, es necesario dejar que Subversion lo sepa, para ello se selecciona la opción **resolved**. Esto borrará los tres ficheros temporales y Subversion no considerará por más tiempo que el fichero está en estado de conflicto. Las siguientes imágenes muestran cómo queda resuelto el conflicto para ambas aplicaciones, dejando claro al Subversion que ya no existe conflicto.

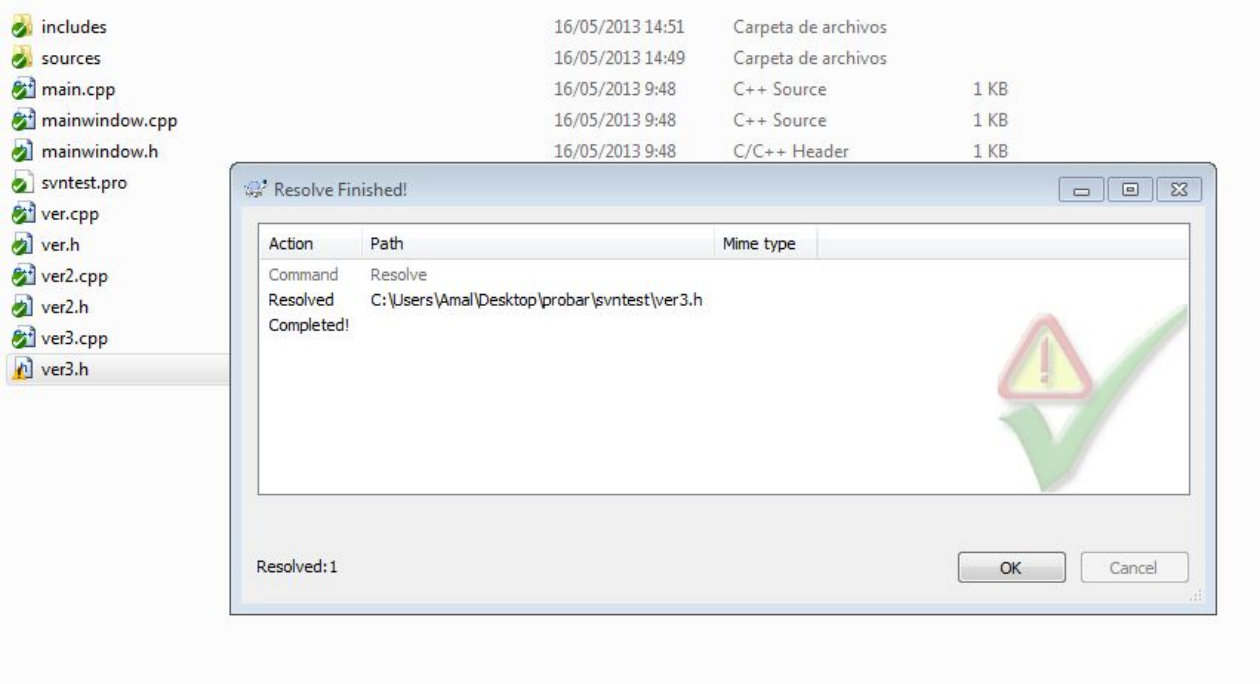

Fig. 28 Conflicto resuelto desde TortoiseSVN.

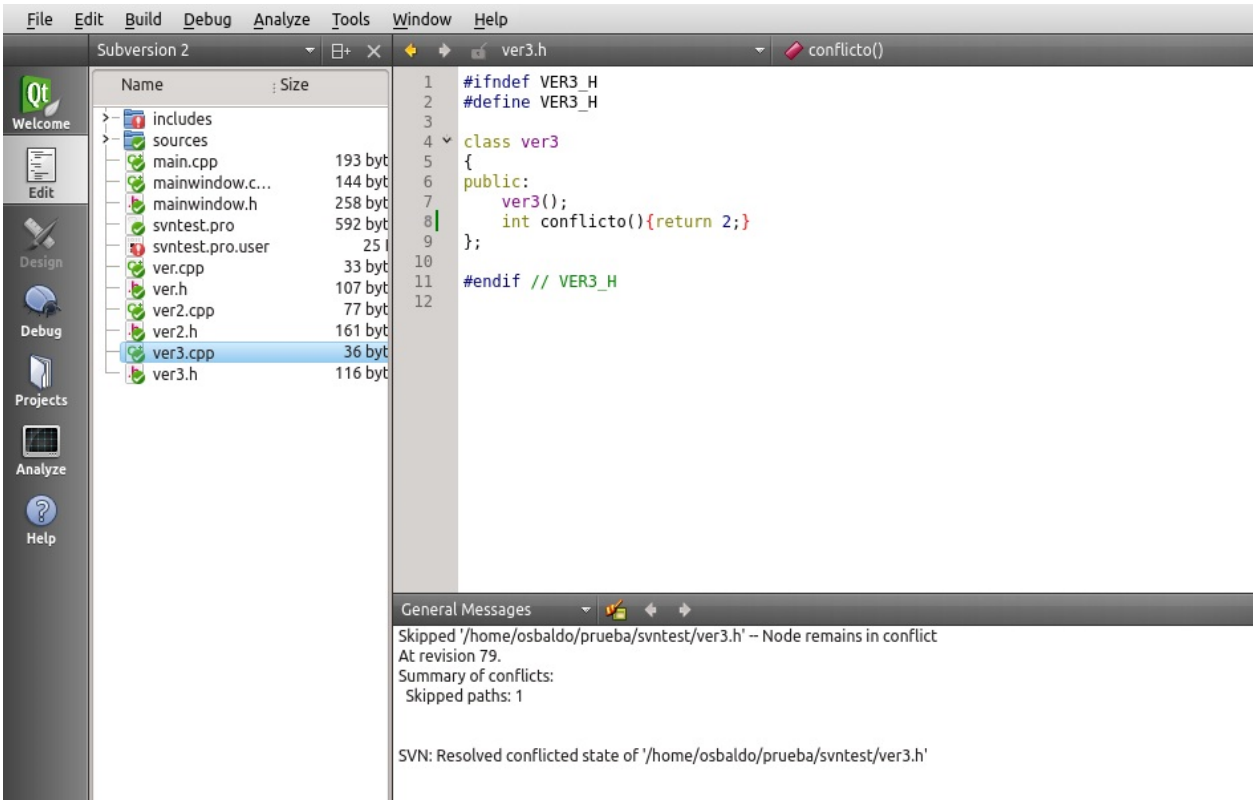

Fig. 29 Conflicto resuelto desde el *plugin* propuesto.

CONCLUSIONES

Con la realización de este trabajo se desarrolló un *plugin* que permite una mejor integración con el sistema de control de versiones Subversion, permitiendo utilizar el mismo desde Qt Creator sin hacer necesario el uso de herramientas externas.

- La incorporación de la retroalimentación visual ayuda a los desarrolladores a no confundirse y conozcan claramente los archivos modificados.
- La incorporación de las nuevas funcionalidades hace posible la utilización del *plugin* para conectarse con el SVN haciendo más fácil el trabajo de los desarrolladores.

RECOMENDACIONES

Al *plugin* propuesto en el presente trabajo se le pueden añadir algunas mejoras entre las cuales se encuentra:

- Adicionar un editor gráfico para resolver los conflictos, similar a los que presentan herramientas como TortoiseSVN y Eclipse.
- Permitir la edición de los archivos, desde el *plugin*, con un doble clic.

BIBLIOGRAFÍA

1. Observatorios Tecnológicos. [En línea] [Citado el: 9 de 12 de 2012.] http://recursostic.educacion.es/observatorio/web/fr/software/software-general/548-luis-garcia.

2. Centro Inca. [En línea] [Citado el: 10 de 12 de 2012.] http://mario010890.blogspot.es/1316118339/.

3. JAVAWORLD solutions for java developers. *JAVAWORLD solutions for java developers.* [En línea] 2006-2013. [Citado el: 2 de 3 de 2013.] http://www.javaworld.com/javaworld/jw-09-2007/jw-09-versioncontrol.html.

4. **Saldaña, Gabriel.** Gabriel Saldaña. [Online] [Cited: 5 4, 2013.] http://www.gabrielsaldana.org/platica_git.pdf.

5. **Jerez, Angel García.** AdictosAlTrabajo. *AdictosAlTrabajo.* [Online] 09 17, 2010. [Cited: 3 20, 2013.] http://www.adictosaltrabajo.com/tutoriales/tutoria.

6. **Donato, Gerardo Morgade.** *ClioBD. Sistema de control de versiones para bases de datos.* La Habana : s.n., 2009.

7. Usability.gov. *Usability.gov.* [Online] [Cited: 02 03, 2013.] http://www.usability.gov/methods/test_refine/implement.html.

8. **Ivar Jacobso, Grady Booch, James Rumbauch.** *El Proceso Unificado de Desarrollo de Software.* Madrid : ADDISON WESLEY.PEARSON EDUCACIÓN, S. A., 2000. ISBN 84-7829-036-2.

9. **López, José Luis.** Slideshare. *Slideshare.* [Online] 10 15, 2009. [Cited: 01 19, 2013.] http://www.slideshare.net/jlpino/control-de-versiones-y-subversion.

10. Hasheado.com. *Hasheado.com.* [Online] 02 16, 2010. [Cited: 01 20, 2013.] http://www.hasheado.com/usando-subversion-desde-la-linea-de-comandos.html.

11. **Ben Collins-Sussman, Brian W. Fitzpatrick, C. Michael Pilato.** *Control de versiones con Subversion.* 2002, 2003, 2004. 3324.

12. Qt Project. *Qt Project.* [Online] [Cited: 3 25, 2013.] http://qt-project.org/wiki/QtCreatorWhitepaper.

13. VCreate Logic. [Online] [Cited: 9 5, 2012.] http://www.vcreatelogic.com/downloads/files/Writing-Qt-Creator-Plugins.pdf.

14. Manuel Resinas de Reyna, Manuel J. Recena Soto. Introducción a los Sistemas de Control de Versiones. 2005.

GLOSARIO DE TÉRMINOS

B

BSD: La licencia BSD es la licencia de software otorgada principalmente para los sistemas BSD (Berkeley Software Distribution). Pertenece al grupo de licencias de software Libre. Esta licencia tiene menos restricciones en comparación con otras como la GPL estando muy cercana al dominio público. La licencia BSD al contrario que la GPL permite el uso del código fuente en software no libre.

D

DICOM: En Inglés (Digital Imaging and Communication in Medicine) es el estándar reconocido mundialmente para el intercambio de imágenes médicas, pensado para el manejo, almacenamiento, impresión y transmisión de imágenes médicas. Incluye la definición de un formato de fichero y de un protocolo de comunicación de red. El protocolo de comunicación es un protocolo de aplicación que usa TCP/IP para la comunicación entre sistemas. Los ficheros DICOM pueden intercambiarse entre dos entidades que tengan capacidad de recibir imágenes y datos de pacientes en formato DICOM.