Universidad de las Ciencias Informáticas
Vertex, Entornos Interactivos 3D, Facultad 5

Generador de plantillas para VirtualLabSDK

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

Autor: Arnaldo Enrique Peñaranda Prim

Tutores: Ing. José Andrés Suárez Alberteris

MSc. Orlay García Ducongé

La Habana, 2016

Lo que está bien puesto en el juicio será de seguro bien puesto en los labios
José Martí

Dedicatoria

A mi familia, y en especial a mis padres, hermana y abuelos por apoyarme con amor infinito en todos los retos que he afrontado en mi carrera.

A mi tía Nivia que ya no se encuentra entre nosotros.

A todas aquellas personas que me quieren y han estado conmigo en todo momento.

Agradecimientos

Con la realización de este trabajo culmina una etapa de mi vida y a la vez, se cumple uno de los sueños que tanto he esperado. Por esta razón quisiera agradecer a todas aquellas personas que de una forma u otra, han influido en la realización del mismo. Agradecerles a mis padres por el apoyo que me han dado todo este tiempo, en especial a mi madre por formarme y convertirme en la persona que hoy soy. A mi hermana y mi cuñado por ser mis segundos padres, los que siempre han estado ahí apoyándome en cada segundo. A mi hermano, cuñada, Lula y María que desde pequeño han contribuido a mi preparación y me han acogido con mucho cariño. A mis sobrinos que han sido un regalo de la vida. En especial reconocer a mi sobrino mayor por ser un ejemplo de valentía y de ganas de vivir inmensa, al vencer las difcultades de su enfermedad.

Agradecer también a mis abuelos, en especial mi abuelito que es mi escudero, el que más tiempo pasa conmigo y el que me ha soportado todo este tiempo. A la inmensa cantidad de primos que tengo, pero en especial a Jessica, Rachel y Ernestico. Qué decir de mis primos sobrinos: Meli, Anthony y mi chinita querida Meig.

Quisiera agradecerles a Adrián y a mis tíos Lissy y Ernesto por el apoyo que siempre me han dado. A mi novia y su familia por la acogida y la confianza que me han brindado. A mis amistades del barrio, en especial Abel, Roberto que siempre han estado ahí conmigo en las buenas y en las malas. A todos los profesores y amigos que he hecho en la UCI pero principalmente a los de mi apartamento; los amigos de guerra. A los amigos que por algún motivo ya no se encuentran en la Universidad pero que en algún momento fueron

artífices de este sueño. A mis tutores, en especial a Jose Andrés que me ayudó inmensamente y que siempre estuvo al tanto en la realización de este trabajo.

En fin, a todos aquellos que aportaron su granito de arena para que pudiera seguir en esta batalla enfrentando muchas dificultades en el camino.

En estas últimas líneas quisiera agradecer de forma muy especial a una persona que ya no se encuentra entre nosotros, a alguien que lamentablemente no pudo estar en este momento de felicidad, a mi tía Nivia. Realmente es una lástima que no esté, sé que se sentiría muy orgullosa de este momento. Quisiera decirte que te extraño mucho y desde aquí te mando todo el amor y el cariño del mundo.

Declaración de autoría

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

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

Arnaldo Enrique Peñaranda Prim
Autor

Ing. José Andrés Suárez Alberteris
Tutor

MSc. Orlay García Ducongé
Co-Tutor

Resumen

En la actualidad la humanidad vive un acelerado desarrollo tecnológico a nivel mundial. Con el paso de los años es más evidente el empleo de modernas tecnologías en casi todas las esferas de la vida; ya sea en la salud, la educación, la cultura o el deporte. Cuba no se encuentra ajena a algunos de estos adelantos tecnológicos y gran parte de ellos, se llevan a cabo en la Universidad de la Ciencias Informáticas (UCI). Entre los *software* que se desarrollan en esta Universidad se encuentran los relacionados con el tema de los laboratorios virtuales. Por lo que el presente trabajo muestra el desarrollo de una aplicación de escritorio que se encarga de facilitar el inicio del proceso de creación de un laboratorio virtual.

Para lograr este propósito, se utilizó la metodología *Extreme Programming* y como lenguaje de programación se empleó C++ sobre el entorno de desarrollo *Qt Creator*. Como lenguaje de marcado se empleó XML y la herramienta de modelado fue *Visual Paradigm for UML*. Se realizaron pruebas de aceptación para comprobar el buen funcionamiento del sistema y un experimento para comprobar su utilidad.

El resultado final obtenido fue una herramienta que logra minimizar los tiempos de inicio del proceso de desarrollo de un laboratorio virtual en el Centro de Entornos Interactivos 3D (VERTEX). La aplicación generará las plantillas que harán menos engorroso el proceso de configuración de proyectos relacionados con este tema y permitirá que se le puedan incorporar nuevas herramientas o tecnologías en un futuro.

Palabras clave: configuración, generador de plantillas, laboratorio virtual, proyectos.

Índice general

Índice de figuras

Índice de tablas

Introducción

Las Tecnologías de la Información y las Comunicaciones (TIC) han tenido un gran impacto a nivel mundial, contribuyendo a un aumento considerable del desarrollo de la humanidad. Su empleo se encuentra reflejado en casi todas las esferas de la vida diaria, ya sea en el ámbito social, económico, político, como el cultural. Una de las aplicaciones en la que las TIC juega un papel fundamental es en la esfera de la educación, donde determinadas plataformas permiten desarrollar herramientas de apoyo, dentro de las que se encuentran los laboratorios virtuales.

Los laboratorios virtuales constituyen un sistema de herramientas metodológicas, a partir de un espacio dotado de los medios necesarios para realizar investigaciones, experimentos y trabajos de carácter científico o técnico, producido por un sistema informático [1]. Representan una posible extensión de los verdaderos laboratorios y abren nuevas perspectivas que no podrían ser exploradas completamente dentro de un laboratorio tradicional.

El Centro de Entornos Interactivos 3D (VERTEX) de la Facultad No.5 de la Universidad de las Ciencias Informáticas (UCI) se centra en el desarrollo de aplicaciones de realidad virtual. El centro cuenta con un *framework*[1] llamado VirtualLabSDK que puede ser empleado para desarrollar laboratorios virtuales con el objetivo de apoyar el proceso de enseñanza-aprendizaje en Cuba.

VirtualLabSDK es un *framework* con una arquitectura basada en capas y componentes; presenta un enfoque multiplataforma[2] y representa una guía fundamental para la base del desarrollo de los laboratorios virtuales en VERTEX. Este *framework* define un conjunto de *plugins* a partir de un grupo de actividades, donde cada actividad tiene relacionada una lista de ejercicios [2].

[1] Estructura en la cual pueden ser desarrollados distintos tipos de proyectos de *software*. Normalmente incluye programas de ayuda, bibliotecas de código y lenguajes de programación.

[2] Término utilizado frecuentemente en informática para indicar la capacidad o características de poder funcionar o mantener una interoperabilidad de forma similar en diferentes sistemas operativos o plataformas.

Para iniciar el desarrollo de un Laboratorio Virtual se debe configurar una estructura de ficheros y carpetas bien definidas en cada uno de los niveles, a partir de las actividades y sus respectivos ejercicios. Inicialmente, se genera la carpeta principal en la que se agrupan los ficheros de configuración del nuevo proyecto y los de cada ejercicio, que se encuentran dentro de las carpetas generadas por cada actividad.

Una de las carpetas que también será generada es la carpeta *"media"* que agrupa todos los estilos e imágenes a utilizar. El nuevo laboratorio será ejecutado a partir de que se modifique un ejemplo que contiene los elementos básicos para el inicio del proceso de desarrollo de un laboratorio virtual.

Posteriormente el usuario modificará cada uno de los nombres de las carpetas, o creará nuevas carpetas en función de la cantidad de actividades con sus respectivos ejercicios. Cada ejercicio tiene relacionado un nombre que lo identifica y los *plugins* que utilizará de los disponibles en el *framework*; además el código fuente del ejercicio será diferente en dependencia de su tipo.

En los ficheros de configuración de cada ejercicio se debe especificar las herramientas a utilizar con sus diferentes formas de declaración en función del tipo de ejercicio. Se realiza la configuración del fichero XML *"exercises.cfg"* el cual contiene todos los nombres e imágenes de las actividades y ejercicios que se mostrarán en el diálogo de inicio al abrir la aplicación. Actualmente se modifica de forma manual toda una estructura de carpetas y ficheros bien definidas por un laboratorio ya existente, lo que hace que se convierta en un proceso complejo.

Teniendo en cuenta la situación antes planteada se define como **problema de la investigación**: ¿Cómo contribuir al proceso de creación y configuración de las aplicaciones desarrolladas con el *framework* VirtualLabSDK del centro VERTEX?

Por lo que se determinó como **objeto de estudio**: el proceso de creación y configuración automatizada de proyectos.

Para darle solución al problema planteado se define como **objetivo general**: desarrollar un generador de plantillas de proyecto para VirtualLabSDK que permita reducir el tiempo de creación de un laboratorio virtual y como **campo de acción**: el proceso de creación y configuración automatizada de proyectos sobre el *framework* VirtualLabSDK.

Para darle cumplimiento al objetivo se proponen las siguientes **tareas**:

1. Elaboración del marco teórico a partir del estudio de herramientas existentes encargadas de generar plantillas para la creación de nuevas aplicaciones.
2. Caracterización del *framework* VirtualLabSDK para acoplar correctamente el sistema desarrollado.

3. Selección de la metodología y las herramientas a utilizar para desarrollar el generador de plantillas.
4. Diseño de diagrama de clases, tablas de historias de usuarios y un prototipo no funcional del generador de plantillas.
5. Implementación del generador de plantillas a partir del diseño realizado.
6. Realización de las pruebas de aceptación a la solución implementada.

Para la realización de este trabajo se utilizaron los siguientes **métodos de investigación**:

- **Teóricos:**

  Histórico-Lógico: Empleado para profundizar acerca de soluciones similares con el estudio de alguna de las herramientas de configuración de proyectos existentes y basado en estos datos, complementar las características necesarias para la solución que se propone.

  Analítico-Sintético: Para la obtención y análisis de la información relacionada con herramientas de configuración de proyectos y el *framework* VirtualLabSDK.

  Modelación: Utilizado en la representación del conocimiento acumulado durante la investigación sobre el desarrollo de laboratorios virtuales en el centro VERTEX, así como el diseño de los diagramas y del prototipo[3] de un sistema funcional.

- **Empíricos:**

  Consulta de las fuentes de información: Empleado para la selección de la información necesaria en el Trabajo de Diploma del centro VERTEX relacionado con la arquitectura de *software* para los laboratorios virtuales.

  Consulta de especialistas: Para que las personas calificadas en el tema del desarrollo de laboratorios virtuales en el centro VERTEX evalúen la aplicabilidad y utilidad del generador de plantillas desarrollado.

[3] Modelo del ciclo de vida del *software*.

CAPÍTULO 1

Fundamentación Teórica

1.1. Introducción

El presente capítulo tiene como objetivo, sentar las bases y facilitar el entendimiento de capítulos posteriores, al abordar los elementos conceptuales relacionados con la investigación desarrollada de este trabajo. Partiendo de los conceptos más generales, se profundiza en la arquitectura de *software*, con un enfoque orientado a la arquitectura por capas y componentes empleada en el *framework* VirtualLabSDK y así lograr un mejor entendimiento de dicho *framework*. Además, se hace referencia a la identidad marcaria utilizada en la UCI y al análisis de algunas de las herramientas de configuración de proyectos existentes. Se realiza un estudio de herramientas y tecnologías como son el lenguaje de marcado XML y las etiquetas de reemplazo. Por último, se muestra un estudio sobre las diferentes herramientas y metodologías de desarrollo de *software* más utilizadas. El estudio de cada una de las metodologías existentes, brindará los elementos necesarios para una posterior selección de la metodología más conveniente a emplear en la propuesta de solución.

1.2. Laboratorio Virtual

Un laboratorio virtual es un sistema computacional que se encarga de dar la sensación de estar en un mundo real a partir del empleo de animaciones, imágenes, sonidos, que simulan y visualizan todo el ambiente de un laboratorio tradicional [3]. Sin embargo, no se considera que el laboratorio virtual vaya a suplantar a los verdaderos laboratorios o competir con ellos, simplemente son una posible extensión de los laboratorios tradicionales [1].

El empleo de un laboratorio virtual en Cuba sería de gran ayuda para la preparación de los estudiantes y profesores en el proceso de enseñanza-aprendizaje, ya que constituye una alternativa ante la falta de recursos, permitiendo la reducción de costo de materiales de enseñanza y mejorando la formación de los estudiantes, ya que ponen en práctica los conocimientos adquiridos en clases. Por eso, la UCI cuenta con proyectos relacionados con el desarrollo de laboratorios virtuales, y a continuación,

se muestra un ejemplo de un laboratorio virtual desarrollado en esta Universidad:

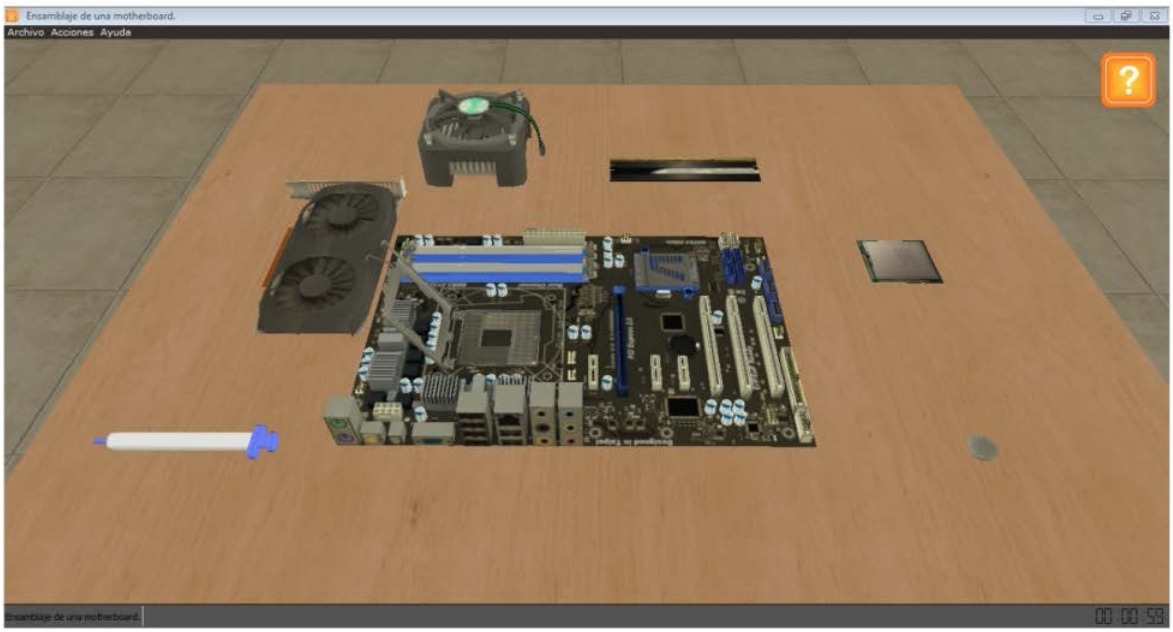


Figura 1.1. Ensamblaje de la placa madre de una computadora

El laboratorio que se muestra en la Figura 1.1 es utilizado como una herramienta de apoyo a la asignatura Arquitectura de Computadoras de la carrera Ingeniería en Ciencias Informáticas de la UCI. En la imagen se ilustra una actividad del laboratorio que representa el ensamblaje de una placa madre. Esta actividad tiene como objetivo mostra el ensamblado correcto de una tarjeta madre con sus diferentes componentes, siendo una de las fases en el ensamblaje de un computador. El propósito que persigue este laboratorio es que el estudiante desarrolle las habilidades de identificación y colocación de los componentes de las computadoras.

1.2.1. Identidad Marcaria

Actualmente en la UCI existen cinco líneas de desarrollo: salud, educación, administración pública, empresa-industria, y telemática. Cada una de estas líneas tiene una estrategia propia de promoción, por lo que la UCI ofrece una identidad sombrilla, con el objetivo de reforzar dicha promoción de cara al mercado.

La identidad es el conjunto de rasgos o formas que definen una institución o persona en un tiempo y espacio determinado, procurando una diferenciación-identificación en el entorno. Su principal objetivo es establecerse en la memoria de su público. En este sentido una identidad se define por elementos que caracterizan un sujeto, institución, personalidad, esencia y que a la vez son factibles de representar mediante la conformación de una imagen. También implica los conceptos acerca de la calidad, la filosofía y seguridad presentes en cualquier institución, producto o servicio. Algunos de los recursos gráficos, ya sea marca

o logotipo, están relacionados estrechamente con una identidad. De esta forma una identidad marcaria es representada a partir de un identificador visual que está conformado por un signo no verbal. Esta representación visual busca ser fácilmente recordado y diferenciado, siendo diseñado en función de los atributos a comunicar [4].

En la UCI se emplea la circunferencia como forma básica para expresar limpieza y avance tecnológico, siendo este el principal elemento distintivo y unificador. Su identificador visual también se compone por un logotipo que constituye la representación del nombre de la marca y por un identificador que define la línea de producción a seguir [5].

Figura 1.2. Líneas de desarrollo.

En la Figura 1.2, se muestran los identificadores visuales que representan la identidad marcaria de cada una de las cinco líneas de desarrollo que existen actualmente en la UCI.

Figura 1.3. Líneas de desarrollo.

En la Figura 1.3, se muestra un ejemplo de interfaz gráfica de un laboratorio virtual, utilizando un estilo predefinido en función de su estrategia marcaria.

1.3. Arquitectura

A partir del importante avance en el desarrollo de *software* en el mundo, los problemas de diseño van más allá de los algoritmos y las estructuras de datos para su computación. Por lo que se ha demostrado la importancia del uso de una arquitectura de *software* funcional y estable, permitiendo agilizar los tiempos de desarrollo y aumentando la reutilización de componentes, además de proporcionar un grupo de funcionalidades para alcanzar objetivos de rendimientos razonables. La arquitectura ofrece una clara perspectiva del sistema completo, organiza su desarrollo, y lo hace evolucionar [6].

El *framework* VirtualLabSDK presenta un enfoque multiplataforma y su arquitectura está basada en capas y en componentes. El estilo basado en capas admite muy naturalmente optimizaciones. Además el estilo basado en componentes hace que se implemente una interfaz bien definida por cada componente para proveerle funcionalidad, permitiendo el desarrollo sin afectar otras partes del sistema. Esta combinación de estilos brinda facilidad de desarrollo y proporciona un amplio nivel de reutilización [2].

1.3.1. Descripción general de la arquitectura basada en capas de los laboratorios virtuales

El estilo basado en capas funciona de manera jerárquica tal que cada capa proporciona servicios a la capa inmediatamente superior y toma los componentes que le brinda la inmediatamente inferior [7]. Cada capa mantiene comunicación solo con su capa adyacente, mejorando el proceso de separación entre ellas de ser necesario. La arquitectura de VirtualLabSDK está compuesta por las siguientes capas:

Presentación: se encarga de la interacción con el usuario, y permite reutilizar elementos comunes a otros laboratorios.

Lógica: se encarga de ejecutar los requerimientos de cada laboratorio virtual.

Soporte: contiene un grupo de componentes de soporte que son utilizados por más de un laboratorio virtual.

Capa Transversal: esta capa se encuentra de forma transversal brindándole un grupo de funciones que pueden ser accedidas desde cualquier capa evitando realizar llamadas innecesarias a las capas adyacentes [2].

En la Figura 1.4 se muestra de forma más detallada cómo se organizan las diferentes capas de la arquitectura antes mencionada :

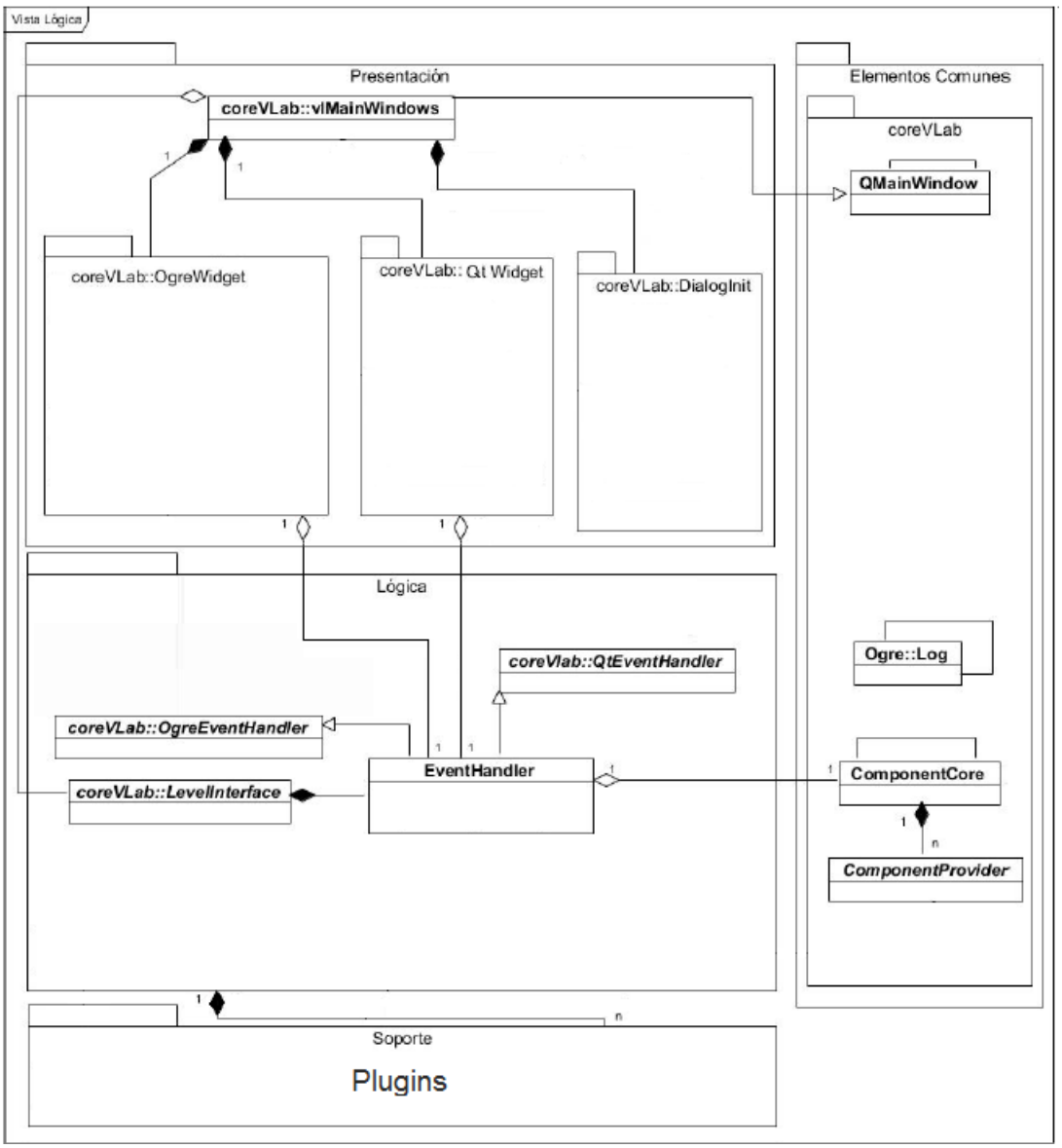


Figura 1.4. Organización de las capas [2].

Este estilo permite particionar un problema complejo en una secuencia de pasos incrementales además de proporcionar un amplio nivel de reutilización. Con el empleo de esta arquitectura también se pueden controlar y encapsular aplicaciones complejas.

1.3.2. Plugin

Un *plugin* es una pequeña aplicación informática que se utiliza para la integración de otras aplicaciones, habitualmente de mayor tamaño, aportándole alguna nueva función, y permite a desarrolladores interactuar con la aplicación para así aumentar la cantidad de funcionalidades que estos puedan realizar. Brinda la po-

sibilidad de separar el código fuente de la aplicación debido a cualquier incompatibilidad que exista con respecto a las licencias [8].

Los *plugins* no suelen funcionar independientes de la aplicación principal y dependen de los servicios prestados por esta, por el contrario un *software* puede funcionar sin la utilización obligatoria de los complementos, lo que posibilita al usuario actualizar y añadir los *plugins* de forma dinámica sin la necesidad de realizar alguna modificación en la aplicación principal [9].

Ventajas del *plugin* :

- Permiten a los desarrolladores externos colaborar con la aplicación principal extendiendo sus funciones.
- Reducen el tamaño de la aplicación.
- Permiten separar el código fuente de la aplicación a causa de la incompatibilidad de las licencias de *software* [9].

Desventajas del *plugin* :

- No se puede ejecutar sino está integrado a una herramienta.
- Se diseñan para funcionalidades específicas [9].

Teniendo en cuenta lo antes expuesto, se considera que los *plugins* son complementos con funcionalidades específicas que se adicionan a un determinado programa para aumentar sus funcionalidades, estos se pueden personalizar para adaptarlos al sistema al cual se integren. Su presencia es muy habitual en los navegadores *web*, en reproductores de música, así como en sistemas de gestión de contenidos. A partir de las características y ventajas que distinguen a los *plugins* se decidió integrarlos en VirtualLabSDK.

1.4. Herramientas para la configuración de proyectos

1. **CMake :**

Como necesidad para crear un potente entorno de compilación, multiplataforma para proyectos en C++ de código abierto surge CMake. La herramienta ha sido diseñada con el objetivo de apoyar el desarrollo de *software* robustos [10].

CMake es un sistema multiplataforma de código abierto que utiliza ficheros de configuración independientes de la plataforma y del compilador para generar un determinado proyecto. Con CMake no sólo se

controla la creación del proyecto, las pruebas, y su empaquetado una vez refinado. CMake se utiliza para controlar los archivos de configuración y el proceso de compilación de *software*, utilizando una misma plataforma. Genera espacios de trabajo que se pueden emplear en cualquier entorno de compilación que sea seleccionado [10].

En cada proyecto se distribuyen de uno a varios ficheros de configuración colocados en cada subdirectorio o directorio de origen llamado *CMakeLists.txt*, en el que se describe el proyecto utilizando un lenguaje con su propia sintaxis. Con él se declara la dirección de los directorios de inclusión, los ficheros fuente, las bibliotecas de las que depende y los productos generados ya sean ejecutables o bibliotecas. Cada *CMakeLists.txt* consta de un grupo de comandos, que CMake ofrece de forma predefinida, aunque de ser necesario, el mismo usuario puede incorporar sus propios comandos.

Posteriormente el fichero es procesado por CMake y genera los archivos de compilación para el entorno que se desee seleccionar. CMake puede generar un entorno de compilación que compile el código fuente, así como crear bibliotecas, generar contenedores y construir ejecutables. Cada proyecto con CMake construye y soporta bibliotecas que pueden ser a su vez dinámica o estática. Una característica interesante de CMake es que genera un archivo de caché que está diseñado para ser utilizado con un editor gráfico. Por ejemplo, al ejecutarse CMake localiza los archivos, bibliotecas y ejecutables, recogiendo esta información en la memoria caché, lo que puede ser cambiado por el usuario antes de la generación de los ficheros [10].

CMake está diseñado para soportar jerarquías de directorios complejas y aplicaciones que dependen de varias bibliotecas. Por ejemplo, CMake apoya proyectos que consisten en múltiples juegos de herramientas como bibliotecas, donde cada conjunto de herramientas puede contener varios directorios, y la aplicación depende de las herramientas más el código adicional. CMake también puede manejar situaciones en las que los ejecutables deben ser construidos con el fin de generar el código, que luego se compila y se enlaza en una aplicación final. Debido a que CMake es de código abierto y tiene un diseño simple, se puede ampliar si es necesario, para apoyar nuevas características [10].

2. **QMake :**

QMake es una herramienta que utiliza el Entorno Integrado de Desarrollo (IDE, por sus siglas en inglés) *Qt Creator*, el cual proporciona un sistema orientado a la gestión de proyectos, a partir del proceso de construcción de aplicaciones, bibliotecas, y otros componentes. Este enfoque brinda un control sobre los archivos fuentes utilizados, y permite que dentro de un solo archivo *qmake* se amplíe la información de cada proyecto hacia un *Makefile*, el cual se encarga de ejecutar los comandos necesarios para compilar y enlazar [11].

Los proyectos se describen por el contenido del archivo del proyecto (.pro). QMake utiliza la información dentro de los archivos para generar *Makefiles* que contienen todos los comandos que se necesitan para

construir cada proyecto. Por lo general, los archivos de un proyecto contienen una lista de archivos fuentes y de encabezamiento, información de configuración, y los detalles específicos de la aplicación, como la lista de bibliotecas [11].

Los archivos de proyecto pueden contener un número de elementos diferentes, incluyendo comentarios, declaraciones de variables, funciones incorporadas, y algunas estructuras de control simples. En la mayoría de los proyectos sencillos, sólo es necesario declarar los archivos fuentes y de encabezamiento, que serán utilizados para construir el proyecto con algunas opciones básicas de configuración [11].

QMake permite crear archivos de proyecto más sofisticados para proyectos complejos. Además de que especifica las opciones de configuración para facilitar el proceso de construcción, esta herramienta utiliza el asistente de *Qt Creator* para crear el archivo del proyecto. El usuario solamente debe elegir la plantilla a utilizar en el proyecto y *Qt Creator* crea un archivo con los valores predeterminados que le permiten generar y ejecutar el proyecto, el cual puede ser modificado en función de las necesidades del propio usuario. QMake también puede ser utilizado para generar archivos de proyectos y en la creación de proyectos simples, el usuario sólo tendrá que ejecutar *qmake* en el directorio de nivel superior de su proyecto para generar un *Makefile* [11].

3. **Maven :**

Maven es una herramienta de gestión de proyectos multiplataforma que se basa en un fichero central, *pom.xml*, donde se define todo lo que necesita el proyecto. Define una forma estándar para construir los proyectos, es una manera fácil de publicar información del proyecto y una forma de compartir archivos **JAR**[1] (por sus siglas en inglés, *Java Archive*) a través de varios proyectos [12].

Es una herramienta que se puede utilizar para la construcción y la gestión de cualquier proyecto basado en *Java*, que facilita el trabajo del día a día de los desarrolladores, ayudando en la comprensión de cualquier proyecto [12].

Objetivos de Maven:

El principal objetivo de Maven es permitir a un desarrollador alcanzar el mayor esfuerzo de desarrollo posible en el menor período de tiempo. Para poder lograr su principal objetivo, Maven realiza las siguientes acciones [12]:

Hacer el proceso de construcción sencilla: Durante el uso de Maven no se elimina la necesidad de saber acerca de sus mecanismos subyacentes, por lo que proporciona de una forma detallada una gran cantidad de funcionalidades que facilitan el proceso de construcción.

[1] Tipo de archivo que permite ejecutar aplicaciones en lenguaje *Java*

Proporcionar un sistema de construcción uniforme: Maven a partir de la construcción de un proyecto emplea su modelo de objeto de proyecto (POM) y un conjunto de *plugins* que son compartidos por todos los proyectos. Una vez que logra familiarizarse con la construcción de un proyecto determinado, Maven automáticamente reconoce cómo serán construidos todos los proyectos. De esta manera se reduce considerablemente el tiempo a la hora de navegar por muchos proyectos.

Proporcionar información de calidad del proyecto: Maven proporciona una gran cantidad de información útil sobre el proyecto, mucha de esta información es tomada de su POM, y en parte generada a partir de fuentes de su proyecto, lo que será de mucha utilidad para los usuarios.

Permitir la migración transparente a nuevas características: Maven proporciona una manera fácil para los clientes para actualizar sus instalaciones y *plugins*.

Una de las utilidades de Maven es el manejo de las dependencias, ya que elimina la descarga manual de los JAR necesarios para un determinado proyecto y su copia manualmente en el *classpath*, siendo un proceso bastante tedioso para el usuario, por lo que con el empleo de esta herramienta solo se necesitaría definir en el archivo *pom.xml* las dependencias necesarias, para posteriormente Maven las descargue y las añada al *classpath* automáticamente [12].

1.5. XML

El Lenguaje de Marcado Extensible, *Extensible Markup Language* (XML) es un lenguaje desarrollado por el consorcio W3C *(World Wide Web Consortium)* que se basa en el *Standard Generalized Markup Language*, Lenguaje de Marcado Generalizado Estándar (SGML) [13]. El XML fue propuesto en 1996, y la primera especificación apareció en 1998, desde entonces su uso ha tenido un crecimiento acelerado.

XML es un metalenguaje, es decir, que puede ser empleado para definir otros lenguajes, como por ejemplo [13]:

- GML (*Geography Markup Language*, Lenguaje de Marcado Geográfico).
- MathML (*Mathematical Markup Language*, Lenguaje de Marcado Matemático).
- RSS (*Really Simple Syndication*, Sindicación Realmente Simple).
- SVG (*Scalable Vector Graphics*, Gráficos Vectoriales Escalables).
- XHTML (*Exttensible HyperText Markup Language*, Lenguaje de Marcado de Hipertexto Extensible).

Este lenguaje de marcado fue diseñado básicamente para estructurar información en un documento o en general, en cualquier fichero que contenga texto, como por ejemplo ficheros de configuración de un programa o una tabla de datos, así como para almacenar y para intercambiar los datos entre diferentes aplicaciones.

Se convirtió en un estándar, ya que es extensible al posibilitar crear otros lenguajes de marca y puede ser utilizado por cualquier aplicación independientemente de la plataforma [14].

También se usa XML en los servicios *web*, con el fin de poder intercambiar datos con el cliente. O en las hojas de cálculo, para guardar la información de manera que se pueda utilizar desde cualquier otra aplicación.

Como principales ventajas del XML se puede decir que [14]:

Es extensible: Después de diseñado y puesto en producción, es posible extender XML con la adición de nuevas etiquetas, de modo que se lo pueda continuar utilizando sin complicación alguna.

Sencillez: Si un tercero decide usar un documento creado en XML, es sencillo entender su estructura y procesarla.

Mejora la compatibilidad entre aplicaciones: Permite la comunicación entre aplicaciones de distintas plataformas, sin que importe el origen de los datos, es decir, se podría tener una aplicación en sistema operativo *Linux* con una base de datos *PostgreSQL* y comunicarla con otra aplicación en sistema operativo *Windows* y base de datos *MS-SQL Server*.

Transforma datos en información: Le añade un significado concreto y los asocia a un contexto, lo que le otorga flexibilidad para estructurar documentos.

A partir de lo expuesto anteriormente, se puede decir que el lenguaje de marcas XML es una tecnología que posibilita un buen manejo de la información. Este lenguaje es estándar, lo que permite compartir la información entre sistemas de diferentes plataformas de una manera fácil. Debido a las facilidades de compatibilidad con diferentes sistemas, el XML es muy utilizado en la actualidad para estructurar información, ya sea en algún documento o fichero.

1.5.1. DOM

El Modelo de Objetos del Documento, *Document Object Model* (DOM) es una interfaz de la plataforma y de lenguaje neutro, que permitirá a los programas y *scripts* acceder dinámicamente y actualizar el contenido, la estructura y el estilo de los documentos. El documento se puede procesar adicionalmente y los resultados del procesamiento se pueden incorporar de nuevo en la página presentada [15].

DOM proporciona un conjunto de bajo nivel de objetos y elementos a partir de un modelo estándar que pueden representar un documento estructurado, siendo capaz de representar cualquier documento HTML o XML. Un analizador de XML compatible con DOM toma los datos de un documento XML y los expone mediante un conjunto de objetos que se pueden programar [16].

Los elementos de un archivo XML en DOM son tratados de forma jerárquica a partir de la estructura de un árbol de nodos, donde cada objeto es un nodo, por lo que cada documento XML se carga totalmente en memoria con esa estructura. Los nodos tienen un conjunto de métodos y propiedades, así como características básicas, bien definidas, como por ejemplo [17]:

- Un nodo tiene un único nodo primario, que se encuentra directamente encima de él. El único nodo que no tiene un nodo primario es la raíz del documento, puesto que éste es el nodo de nivel superior y contiene el propio documento y fragmentos de documento.
- La mayor parte de los nodos pueden tener varios nodos secundarios, que son los que están situados inmediatamente debajo de ellos.

La forma de controlar los atributos es una característica de DOM. Los atributos no son nodos que forman parte de las relaciones entre los nodos primarios y secundarios y entre nodos relacionados. Los atributos se consideran una propiedad del nodo que representa el elemento y están formados por un par nombre-valor [17].

Una de las principales ventajas que presenta es su facilidad a la hora de acceder a los datos en función de la jerarquía de los elementos, así como modificar el contenido de los documentos e incluso crearlos desde cero. Pero su principal problema es en cuanto al costo de tiempo y memoria en el momento de construir el árbol. DOM resulta más útil para leer datos XML en la memoria y cambiar su estructura, agregar o quitar nodos, o modificar los datos mantenidos en un nodo como en el texto contenido en un elemento [17].

A partir de lo planteado anteriormente, DOM se basa en un modelo objetos bien definidos y estructurados. Se puede decir que a pesar de esto, DOM posibilita una mejor combinación y relación entre los objetos. No suele ser una buena solución a la hora de trabajar con grandes cantidades de información debido al tiempo y la cantidad de memoria que consume por su estructura jerárquica en forma de árbol. Aunque posibilita un buen manejo a la hora de acceder a los datos en un documento XML.

1.5.2. SAX

Simple API for XML (SAX) es una interfaz estándar basada en eventos para los analizadores XML. SAX es la API[2] simple para XML, ampliamente adoptada en *Java*, pero existen versiones para varios entornos de lenguaje de programación distintos de *Java* [18].

SAX recibe información de los documentos XML en un flujo continua, por lo que una vez leído no se puede volver atrás, este enfoque lo hace extremadamente eficiente, entregando los documentos XML de casi

[2]Interfaz proporcionada por una biblioteca o sistema para poder acceder a las funciones de esta, que no incluye información sobre la implementación de la biblioteca.

cualquier tamaño en un tiempo lineal y casi constante de la memoria [19]. Es fácil e intuitiva, muchos programadores de *Java* la utilizan, ya que se usa especialmente en situaciones en los que los archivos XML ya están en una forma que es estructuralmente similar a la que se desea obtener.

SAX opta por darle acceso a la información en el documento XML, no como un árbol de nodos, sino como una secuencia de eventos y crea su propio modelo de objetos haciéndolo más rápido. Un analizador (*parser*) SAX es una herramienta más versátil, más veloz y menos potente que un analizador (*parser*) DOM debido a que requiere menos código y menos memoria siendo muy útil si lo que se interesa es rescatar un fragmento de un documento o buscar sólo un elemento en particular [20].

El desarrollo de la API simple para XML (SAX) es una especificación ampliamente utilizada que describe cómo analizadores XML pueden transmitir información de manera eficiente a partir de documentos XML para aplicaciones de *software*. SAX se implementó originalmente en Java, pero ahora recibe el apoyo de casi todos los principales lenguajes de programación [18].

En sentido general a partir de lo planteado anteriormente, se puede decir que SAX es un modelo de eventos que permite recorrer secuencialmente un documento XML. SAX resulta muy efectivo si no se necesita realizar alguna transformación en memoria y si sólo se necesita analizar partes de algún documento. Esto lo convierte en un mecanismo eficiente en cuanto al tiempo y a la memoria empleada en el análisis.

1.6. Etiquetas de reemplazo

Una etiqueta de reemplazo se puede definir como una palabra clave o alguna marca que se utiliza principalmente en la gestión de texto, en las situaciones en la que el texto o parte del texto se asocia con algún elemento. La marca o etiqueta puede ser interpretada generalmente, para realizar alguna acción sobre el mismo texto marcado. Las etiquetas son definidas por el propio desarrollador que le permite crear una forma muy sencilla y flexible de acceso y reclasificación de un dato, con sólo modificarlas. De esta manera, cualquier etiqueta es localizada e identificada donde se puede reemplazar o modificar automáticamente [21]. Se puede decir que unas de las características fundamentales de las etiquetas de reemplazo son las siguientes:

- Las etiquetas no tienen un significado semántico único, por lo que cada etiqueta no puede hacer referencia a más de un dato con un significado semántico variado. Lo que puede conducir a conexiones inadecuadas entre datos sin relación, producto de un inadecuado etiquetado.
- El identificador o nombre de una etiqueta es personal, aunque principalmente se utilizan identificadores que hacen referencia al dato que se quiera conectar.

- Al ser libre la elección del nombre o identificador de la etiqueta, los desarrolladores pueden introducir en el sistema etiquetas sin sentido o que sólo tengan sentido para ellos mismos.
- Varias etiquetas pueden ser utilizadas indistintamente para describir un mismo recurso, por lo que el desarrollador es el encargado de darle un significado a cada una, que posibilite las conexiones entre los datos en función de sus intereses [21].

1.7. Metodología de desarrollo

Muchos son los criterios dados por diferentes expertos acerca del concepto de metodología. Según *Piattini*, la metodología es un conjunto de procedimientos, técnicas, herramientas, y un soporte documental que ayuda a los desarrolladores a realizar nuevos *software* [22].

De manera general la metodología de *software* es una guía que ayuda a trazar la ruta a seguir y cómo actuar en cada una de las etapas del ciclo de desarrollo de *software*. A partir de aplicar una metodología, el proyecto se podrá dividir en varias etapas, las tareas a realizar, las herramientas que serán utilizadas, así como la forma en que será gestionado el proyecto, logrando una mejor eficiencia y eficacia durante todo el proceso de desarrollo, a fin de garantizar un *software* con la máxima calidad posible. En la actualidad son varias las metodologías existentes en el mundo, cada una con características similares y diferentes, que hacen que sean empleadas en diferentes proyectos a partir de sus necesidades, por la que se dividen en dos grandes grupos: robustas y ágiles.

Las metodologías robustas realizan un control y planificación del proyecto durante todo el proceso de desarrollo, haciendo énfasis en la definición de roles, actividades y herramientas a utilizar, lo que exige de una documentación detallada. Por lo que son muy efectivos en proyectos de gran envergadura en función de los recursos y del tiempo a emplear [23]. Algunas de las metodologías robustas son:

- Proceso Unificado de Desarrollo o *Rational Unified Process* (RUP).
- *Microsoft Solutions Framework* (MSF).

La metodología ágil se enfoca más en la obtención de un producto funcional que en la documentación, por lo que provee respuestas rápidas y se adapta fácilmente al cambio. Se puede decir que están orientadas fundamentalmente a proyectos de menor volumen, aunque no renuncia a las prácticas para garantizar la calidad del producto [23]. Algunas de las metodologías ágiles son:

- *Extreme Programming* (XP).
- SCRUM.

1.8. Consideraciones parciales

Con la conclusión del capítulo, se han expuesto los términos más importantes relacionados con el tema de investigación, referenciándose los conceptos de laboratorio virtual, *plugin* e identidad marcaria; éste último enfocado en la identidad marcaria definida en la UCI. También se realizó una investigación y análisis de diferentes herramientas de configuración de proyectos. La selección de las herramientas CMake, QMake y Maven, permitió identificar un grupo de características que ayudarán a la solución del problema. En el capítulo se hizo un estudio de la tecnología XML y de las etiquetas de reemplazo, además del estudio de las diferentes metodologías de *software* ya sean ágiles o robustas. Con el estudio de estas metodologías se tendrá un punto de partida en la selecćon de la metodología a emplear en la solución propuesta. Una vez concluido el capítulo se tienen los elementos necesarios para comenzar a realizar la propuesta de solución al problema de investigación.

CAPÍTULO 2

Características y diseño del Sistema

2.1. Introducción

En el presente capítulo se describe la solución propuesta, se seleccionan las tecnologías y herramientas a utilizar en el desarrollo del trabajo a partir de las descripciones del capítulo anterior. Se selecciona el ambiente de desarrollo y se confecciona el modelo de dominio. Además, se definen cada una de las diferentes fases aplicadas por la metodología de *software* seleccionada para el desarrollo de la solución. También se muestran las historias de usuarios realizadas por el cliente, definiendo a su vez, el total de iteraciones y elaborándose el plan de entrega para cada versión. Al mismo tiempo, el capítulo hará un enfoque en el diseño del sistema, se describirán las clases a través de las tarjetas CRC (Clase-Responsabilidad-Colaborador) , se abordarán los patrones de diseño y se confeccionará el diagrama de clases en lenguaje UML.

2.2. Selección del ambiente de desarrollo

Para darle cumplimiento al objetivo de la investigación a partir de las descripciones de la fundamentación teórica realizada en el capítulo anterior. Se propone realizar una aplicación capaz de generar las plantillas para iniciar el proceso de desarrollo de los laboratorios virtuales. Para poder llevar a cabo esta tarea, se ha seleccionado un conjunto de herramientas y tecnologías que por sus características serán empleadas.

- Para el proceso de desarrollo de la aplicación será empleada la metodología ligera XP. Esta metodología se encuentra basada en la comunicación fluida entre todos los participantes, la realimentación continua entre el cliente y el equipo de desarrollo, y la reutilización del código fuente [24]. Es utilizada para proyectos de corto plazo, donde se cuenta con un pequeño equipo de trabajo y donde se requiere de un corto tiempo de entrega del producto. Consiste en una programación rápida, donde una de sus principales particularidades es que el cliente se integra en el equipo, lo que se convierte en una pieza fundamental para llegar al éxito final [25]. Se puede decir que brinda un enfoque muy efectivo, ya que se adapta a la perfección en proyectos donde sus requisitos son muy cambiantes, y cuando se necesita

reducir el tiempo de desarrollo del *software*, manteniendo siempre su alta calidad [24], por lo que se adaptaría muy bien en la realización de este trabajo.

- Como herramienta CASE (*Computer Aided Software Engineering*, Ingeniería de Software Asistida por Computadora) se selecciona *Visual Paradigm for UML 8.0*, pues soporta el ciclo completo de desarrollo de *software* y ofrece un conjunto de herramientas necesarias para la realización de las diferentes tareas del proceso, tales como: captura de requisitos, planificación, y modelado de clases. Permite la realización del modelado en diferentes idiomas y produce documentación en varios formatos, ya sea en formato HTML, PDF u otros. Como una de las características más importantes se encuentra que es multiplataforma, con licencia gratuita y comercial, que permite realizar una ingeniería directa e inversa, generación de código, así como control de versiones [26].

- Para la implementación de la aplicación se utilizará como lenguaje de programación C++, el cual es un lenguaje versátil, potente y general. Su éxito entre los programadores profesionales le ha llevado a ocupar los primeros puestos como herramienta de desarrollo de aplicaciones [27]. Además es el lenguaje utilizado para el desarrollo de los laboratorios virtuales en el centro.

- Para el diseño de las interfaces gráficas será empleado *Qt Designer*, potente herramienta que lo caracteriza por su facilidad de uso y está orientado al desarrollo de aplicaciones a través del *framework* Qt *(Quasar Technologies)*. Está provisto de herramientas que permiten la creación de formas, etiquetas, botones y otros elementos propios de las interfaces. Permite además, la modificación de las propiedades de cada elemento utilizado y una pre-visualización del diseño creado [28].

- Como IDE se empleará *Qt Creator*, el cual está orientado al desarrollo de aplicaciones a través del *framework* Qt. Posee un editor de código con resaltado de sintaxis, soporte para varios lenguajes de programación, un editor de formas integrado, herramientas para la administración de proyectos, ayuda integrada, depurador, así como completamiento de código [27]. Además de que es el IDE encargado de cargar las plantillas que serán generadas por la aplicación final.

- Como lenguaje de marcas se empleará XML, el cual es una tecnología sencilla que soporta una amplia variedad de aplicaciones, es fácil escribir programas que procesan los documentos XML, es legible para los usuarios, el diseño de XML es formal y conciso, los documentos XML son fáciles de crear [14].

2.3. Propuesta de solución

A partir del análisis realizado y la problemática existente, se propone implementar una aplicación que le permita al desarrollador generar la plantilla base para el inicio de creación de un laboratorio virtual. La solución permitirá agregar todo un conjunto de actividades con sus respectivos ejercicios, así como un grupo de herramientas de gran utilidad para la creación del laboratorio, se podrán seleccionar los *plugins* y el tipo

de ejercicio, ya sea el motor de render[1] Ogre, o la biblioteca de interfaz gráfica Qt. Además de que permitirá seleccionar los diferentes módulos de Qt y el estilo a emplear. Todo esto se realizará de forma dinámica, facilitando siempre que se puedan incorporar otras herramientas en un futuro. Finalmente, la aplicación deberá generar una estructura de carpetas y ficheros bien definidos en cada uno de los niveles, brindando la posibilidad también, de crear o cargar algún fichero de configuración de plantillas existentes. A continuación en la Figura 2.1, se muestra un prototipo de la solución:

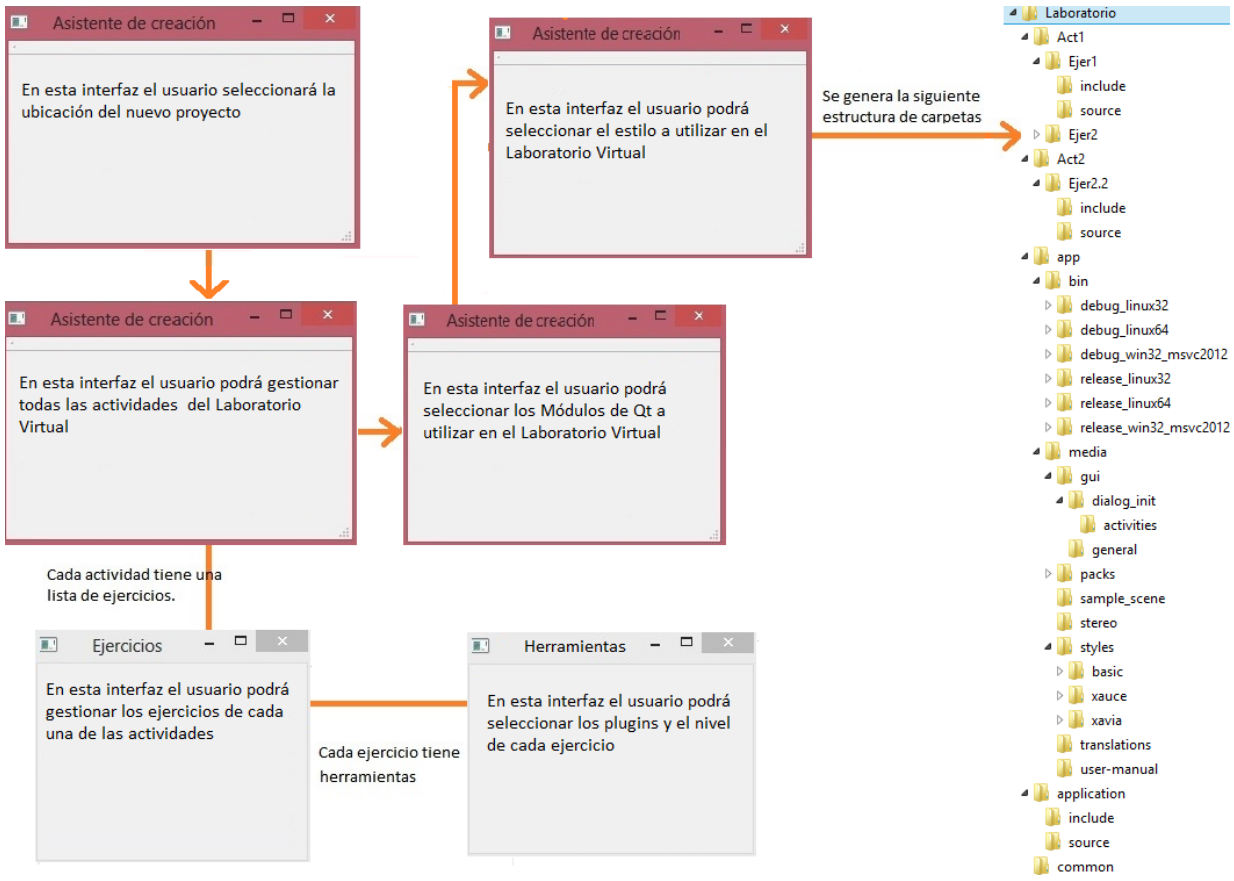


Figura 2.1. Prototipo del *software*.

Para el desarrollo de la solución propuesta serán tomados en cuenta algunos elementos de las herramientas de configuración estudiadas en el capítulo anterior. Estos aspectos serán de utilidad en la implementación de la solución; por lo que fueron seleccionados los siguientes elementos por herramientas:

[1]Biblioteca de clases que facilita al desarrollador el proceso de traducción de la geometría controlada por atributos gráficos, al medio de la imagen.

CMAKE

- Utilizar fichero de configuración independiente de plataforma y de compilador para generar el proyecto.
- A partir del fichero de configuración generar un proyecto.

QMAKE

- Generar archivos de proyectos.
- Especificar opciones de configuración para facilitar el proceso de construcción del proyecto.

MAVEN

- Utilizar fichero xml que contenga toda la información del proyecto.
- Utilizar una manera fácil para que el usuario pueda actualizar o incorporar nuevos *plugins* al proyecto.
- Definir una forma estándar para construir los proyectos.

2.4. Modelo de dominio

Un modelo de dominio se encarga de ilustrar una representación visual de clases conceptuales significativas en un determinado dominio. Por lo que una clase conceptual representa una idea, cosa u objeto. Este modelo se representa mediante un conjunto de diagramas de clases en los que no serán definidas ninguna operación. En sentido general, un modelo de dominio va a mostrar una vista parcial, o abstracción sin enfocarse en los detalles sin interés [29]. En la siguiente Figura 2.2 se muestra el modelo de dominio de la aplicación:

Desarrollador: persona que utiliza la aplicación para crear la plantilla.

Asistente de creación: elemento que se encarga de gestionar todo lo relacionado con la creación de plantillas.

Módulo: objeto que contiene los módulos de Qt necesarios a utilizar.

Actividad: objeto que contiene todas las actividades que forman parte del laboratorio virtual.

Ejercicio: objeto que contiene todos los ejercicios que forman parte del laboratorio virtual para cada actividad.

Plugin: objeto que contiene los plugins necesarios empleados por cada ejercicio.

Estructura de archivos y carpetas: elemento que se encarga de estructurar de forma bien definida un conjunto de archivos y carpetas.

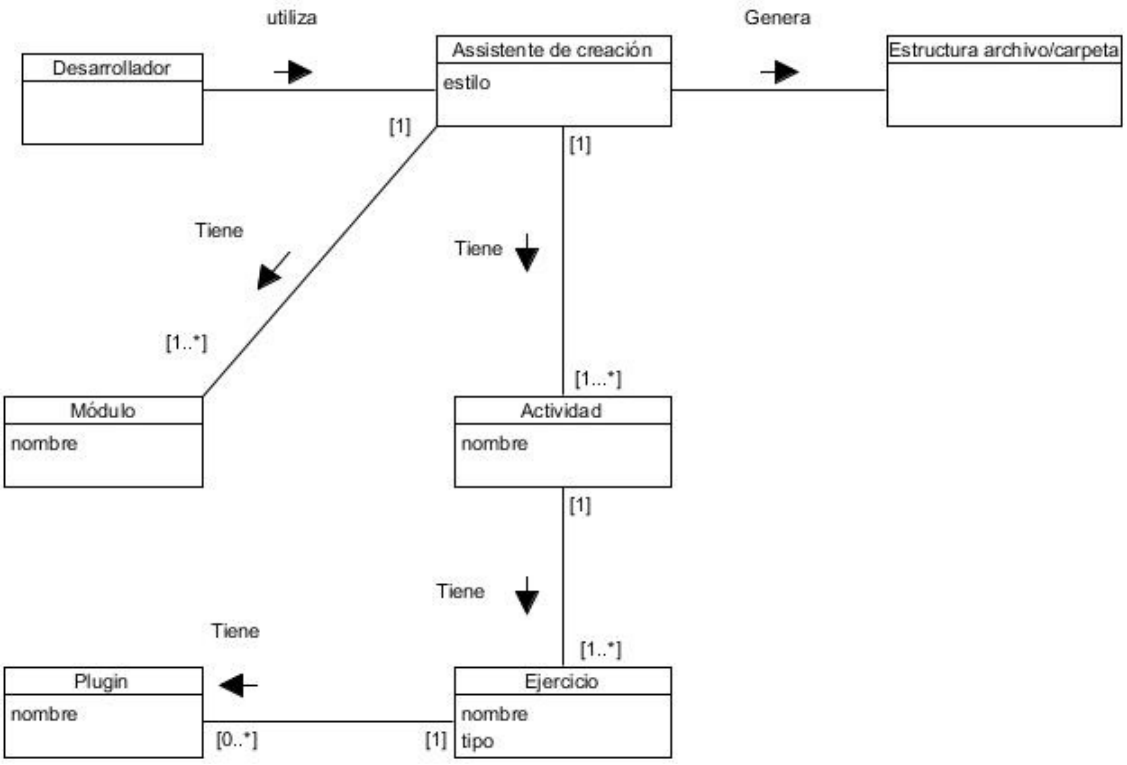


Figura 2.2. Modelo de dominio.

2.5. Personal relacionado con el sistema

La aplicación va destinada a los desarrolladores de los laboratorios virtuales, los cuales podrán utilizarla durante el inicio del proceso de desarrollo del laboratorio que desean desarrollar.

Personas	Justificación
Desarrollador	Utiliza la aplicación para generar plantillas de configuración.

2.6. Fase de exploración

El ciclo de vida del proyecto se inicia con la fase de exploración, los clientes plantean a grandes rasgos, las historias de usuario que son de interés para realizar una primera liberación del producto. Por su parte, el equipo de desarrollo se familiariza con las herramientas, tecnologías y prácticas que serán empleadas en el desarrollo del *software*. Se prueba la tecnología, se explora diversas posibilidades de conformar la arquitectura, además de que se realizarán las estimaciones correspondientes [24].

2.7. Historias de usuario

Las historias de usuario (HU) son una técnica utilizada en XP para especificar los requisitos del *software*. Tienen una descripción breve de las características que el sistema debería presentar. Cada HU debe ser entendible por los clientes y los desarrolladores, posible de probar, de valor para el cliente y lo suficientemente pequeña de forma tal que los desarrolladores puedan implementar media docena de HU en una iteración. El trabajo con ellas tiende a ser muy dinámico y flexible, ya que pueden ser cambiadas o modificadas en cualquier instancia [30]. A continuación, se muestran las HU:

Tabla 2.1. Historia de usuario # 1

Historia de usuario	
Número: 1	**Nombre:** Seleccionar ubicación del proyecto.
Usuario: Desarrollador	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Bajo
Puntos estimados: 1	**Iteración asignada:** 1
Programador responsable: Arnaldo E. Peñaranda Prim	
Descripción: • Permitir al usuario seleccionar la dirección donde se encontrará el proyecto a crear.	
Observaciones: En caso de que el usuario quiera continuar y no haya insertado la ubicación del proyecto saldrá un mensaje de alerta.	

Tabla 2.2. Historia de usuario # 2

Historia de usuario	
Número: 2	**Nombre:** Administrar actividades.
Usuario: Desarrollador	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Media
Puntos estimados: 1	**Iteración asignada:** 1
Programador responsable: Arnaldo E. Peñaranda Prim	
Descripción: El usuario podrá tener un mecanismo para: • adicionar actividades del laboratorio virtual • editar actividades del laboratorio virtual • eliminar actividades del laboratorio virtual	
Observaciones: En caso de que el usuario quiera continuar y no haya agregado una actividad, con al menos un ejercicio, saldrá un mensaje de alerta.	

Tabla 2.3. Historia de usuario # 3

Historia de usuario	
Número: 3	**Nombre:** Administrar ejercicios.
Usuario: Desarrollador	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Alta
Puntos estimados: 1	**Iteración asignada:** 1
Programador responsable: Arnaldo E. Peñaranda Prim	
Descripción: El usuario podrá tener un mecanismo para: • adicionar ejercicios a cada actividad. • editar los ejercicios de cada actividad. • eliminar los ejercicios de cada actividad. • cada ejercicio permitirá asignarle un grupo de plugins. • cada ejercicio brinda la posibilidad de seleccionar su nivel correspondiente.	
Observaciones:	

Tabla 2.4. Historia de usuario # 4

Historia de usuario	
Número: 4	**Nombre:** Seleccionar módulos.
Usuario: Desarrollador	
Prioridad en negocio: Media	**Riesgo en desarrollo:** Baja
Puntos estimados: 1	**Iteración asignada:** 2
Programador responsable: Arnaldo E. Peñaranda Prim	
Descripción: El usuario podrá tener un mecanismo para: • seleccionar los módulos de Qt a utilizar en el laboratorio virtual.	
Observaciones:	

Tabla 2.5. Historia de usuario # 5

Historia de usuario	
Número: 5	**Nombre:** Seleccionar estilos.
Usuario: Desarrollador	
Prioridad en negocio: Media	**Riesgo en desarrollo:** Baja
Puntos estimados: 1	**Iteración asignada:** 2
Programador responsable: Arnaldo E. Peñaranda Prim	

Continúa en la próxima página

Tabla 2.5. Continuación de la página anterior

Descripción: El usuario podrá tener un mecanismo para: • seleccionar el estilo de identidad marcaria a utilizar en el laboratorio virtual.
Observaciones:

Tabla 2.6. Historia de usuario # 6

Historia de usuario	
Número: 6	**Nombre:** Generar estructura de carpetas y ficheros.
Usuario: Desarrollador	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Media
Puntos estimados: 1	**Iteración asignada:** 3
Programador responsable: Arnaldo E. Peñaranda Prim	
Descripción: • Se genera la estructura de archivos y carpetas.	
Observaciones:	

Tabla 2.7. Historia de usuario # 7

Historia de usuario	
Número: 7	**Nombre:** Configurar el contenido de los ficheros.
Usuario: Desarrollador	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Alta
Puntos estimados: 2	**Iteración asignada:** 3
Programador responsable: Arnaldo E. Peñaranda Prim	
Descripción: • Lograr configurar el contenido de los ficheros del proyecto en función de la información obtenida sobre el proyecto creado por el usuario.	
Observaciones:	

Tabla 2.8. Historia de usuario # 8

Historia de usuario	
Número: 8	**Nombre:** Guardar proyecto.
Usuario: Desarrollador	
Prioridad en negocio: Baja	**Riesgo en desarrollo:** Alta
Puntos estimados: 1	**Iteración asignada:** 4
Programador responsable: Arnaldo E. Peñaranda Prim	
Descripción: • Brindar la posibilidad de que el usuario guarde la configuración del proyecto en un fichero XML.	
Observaciones:	

Tabla 2.9. Historia de usuario # 9

Historia de usuario	
Número: 9	**Nombre:** Cargar proyecto.
Usuario: Desarrollador	
Prioridad en negocio: Baja	**Riesgo en desarrollo:** Alta
Puntos estimados: 1	**Iteración asignada:** 4
Programador responsable: Arnaldo E. Peñaranda Prim	
Descripción: • Brindar la posibilidad de que el usuario cargue algún fichero de configuración de un proyecto existente.	
Observaciones:	

2.8. Fase de planificación

La planificación es una fase corta donde el cliente estableció la prioridad de cada historia de usuario y en función de estas prioridades el desarrollador podrá realizar una estimación del esfuerzo necesario para realizar cada una de ellas [23]. Al realizar la planificación se podrá llegar a un acuerdo entre clientes y desarrolladores sobre la posible fecha de entrega de cada grupo de funcionalidades. Para realizar la estimación se toma como unidad de medida el punto. Un punto se considera como una semana (6 días) ideal de trabajo donde los miembros de los equipos de desarrollo trabajan el tiempo planeado, sin ningún tipo de interrupción. Generalmente, las HU necesitan de una a tres semanas de desarrollo [31].

2.9. Estimación de esfuerzos por historias de usuario

Tabla 2.10. Estimación de esfuerzo por historia de usuario

Iteración	Historias de usuario		Puntos estimados (semanas)
1	1	Seleccionar ubicación del proyecto	1
	2	Administrar actividades.	1
	3	Administrar ejercicios.	1
2	4	Seleccionar módulos.	1
	5	Seleccionar estilos.	1
3	6	Generar estructura de carpetas y ficheros	1
	7	Configurar el contenido de los ficheros.	2
4	8	Guardar proyecto.	1
	9	Cargar proyecto.	1
Total			**10.0**

2.10. Plan de iteraciones

Luego de identificar y describir las HU y estimar el esfuerzo propuesto para la realización de cada una de ellas, se realizará la confección del plan de iteraciones. El cual tiene como objetivo definir las HU que serán implementadas en cada una de estas iteraciones. Para conformar el plan, se planificaron un total de 4 iteraciones, las que se describen a continuación, de manera más detallada:

Iteración 1: se implementarán un grupo de HU de importancia alta para el desarrollo del sistema en las cuales se incluyen la 1,2 y 3. Terminada la iteración, el sistema permitirá darle la ubicación del proyecto que se quiera crear y se podrá realizar el proceso de administración de las actividades para el laboratorio virtual. Posteriormente se realizarán las pruebas pertinentes a cada una de las funcionalidades.

Iteración 2: se implementarán las HU de importancia media para el desarrollo del sistema en las cuales se incluyen la 4 y 5. Terminada la iteración, el sistema permitirá seleccionar los componentes necesarios para crear un laboratorio virtual, como son los módulos y el estilo. Posteriormente se realizarán las pruebas pertinentes a cada una de las funcionalidades.

Iteración 3: se implementarán las restantes HU de importancia alta para el desarrollo del sistema en las cuales se incluyen la 6 y 7. Terminada la iteración, el sistema permitirá generar de forma bien definida la estructura de carpetas y archivos necesarios para crear un proyecto. Posteriormente se realizarán las pruebas pertinentes a cada una de las funcionalidades.

Iteración 4: se implementarán las HU de importancia baja para el desarrollo del sistema en las cuales se incluyen la 8 y 9. Terminada la iteración, el sistema permitirá que se guarden y se carguen la configuración de cada proyecto a partir de un fichero XML. Posteriormente se realizarán las pruebas pertinentes a cada una de las funcionalidades.

Tabla 2.11. Plan de duración de las iteraciones

Iteración		Historias de usuario	Duración (semanas)
1	1	Seleccionar ubicación del proyecto	3
	2	Administrar actividades.	
	3	Administrar ejercicios.	
2	4	Seleccionar módulos.	2.0
	5	Seleccionar estilos.	
3	6	Generar estructura de carpetas y ficheros	3.0
	7	Configurar el contenido de los ficheros.	
4	8	Guardar proyecto.	2.0
	9	Cargar proyecto.	
Total			**10.0**

2.11. Plan de entregas

En función del plan de iteraciones expuesto con anterioridad, se procede a realizar el plan de entregas, el cual tiene como objetivo definir las diferentes entregas que se irán realizando, con un orden especificado. A continuación se muestar una tabla con el plan de entregas:

Historias de Usuarios	Final 1era iteración 3ra semana de enero	Final 2da iteración 1ra semana de febrero	Final 3era iteración 4ta semana de febrero	Final 4ta iteración 2da semana de marzo
Seleccionar ubicación del proyecto.	V1.0			
Administrar actividades.	V1.0			
Administrar ejercicios.	V1.0			
Seleccionar módulos.		V1.1		
Seleccionar estilos.		V1.1		
Generar estructura de carpetas y ficheros.			V1.2	
Configurar el contenido de los ficheros.			V1.2	
Guardar proyecto.				V1.3
Cargar proyecto.				V1.3

2.12. Tarjetas CRC

Las tarjetas CRC son fichas, una por cada clase, en las que se escriben brevemente las responsabilidades de la clase e indican las colaboraciones existentes con el conjunto de objetos, al fin de llevar a cabo ciertas responsabilidades. La información recopilada se puede enriquecer utilizando diagramas de clases y de interacción. Lo importante no son las tarjetas o los diagramas, sino tener presente la asignación de responsabilidades [29]. En sentido general estas tarjetas brindan un acercamiento a la identificación de clases y la información referente a los objetos desarrollados en el proyecto. Cada tarjeta identifica una clase, sus responsabilidades y relaciones con otras clases. A continuación se detallan las clases del negocio mediante las siguientes tarjetas CRC.

Tabla 2.12. Tarjeta CRC # 1

Tarjeta CRC	
Clase: Assistant	
Responsabilidad	**Colaboración**
Es la clase principal y se encarga de : • Controlar la interfaz de la aplicación. • Administrar el listado de actividades del laboratorio virtual • Administrar el listado de ejercicios para cada actividad. • Validar nombre del proyecto. • Validar la existencia de archivos. • Guardar la configuración del proyecto. • Cargar la configuración del proyecto. • Administrar el historial de los 5 proyectos utilizados más reciente. • Adicionar los módulos de Qt a utilizar en el laboratorio virtual. • Adicionar el estilo a utilizar en el laboratorio virtual. • Generar la estructura de carpetas. • Generar la estructura de ficheros. • Configurar el contenido de los ficheros.	Activity AddActivity AddExcercise

Tabla 2.13. Tarjeta CRC # 2

Tarjeta CRC	
Clase: Activity	
Responsabilidad	**Colaboración**
Es la clase que se encarga de : • Guarda los atributos, nombre, y listado de ejercicios. • Construye una actividad a partir de sus atributos.	Excercise

Tabla 2.14. Tarjeta CRC # 3

Tarjeta CRC	
Clase: Excercise	
Responsabilidad	**Colaboración**
Es la clase que se encarga de : • Guarda los atributos, nombre, tipo, y un listado de plugins para el ejercicio. • Construye un ejercicio a partir de sus atributos.	

Tabla 2.15. Tarjeta CRC # 4

Tarjeta CRC	
Clase: AddExcercise	
Responsabilidad	**Colaboración**
Es la clase que se encarga de : • Mostrar la interfaz para agregar el ejercicio. • Mostrar la interfaz para editar el ejercicio. • Adicionar el ejercicio. • Editar el ejercicio.	Excercise

Tabla 2.16. Tarjeta CRC # 5

Tarjeta CRC	
Clase: AddActivity	
Responsabilidad	**Colaboración**
Es la clase que se encarga de : • Mostrar la interfaz para agregar la actividad. • Mostrar la interfaz para editar la actividad. • Adicionar la actividad. • Editar la actividad.	- Activity

2.13. Diagrama de clases

El diagrama de clases es un mecanismo de representación a las especificaciones de las clases e interfaces del *software* [29]. Este diagrama no es necesario según la metodología XP, pero es una herramienta muy importante como base para el entendimiento y comprensión de la estructura del sistema. A continuación se muestra el diagrama de clases del diseño del sistema a desarrollar.

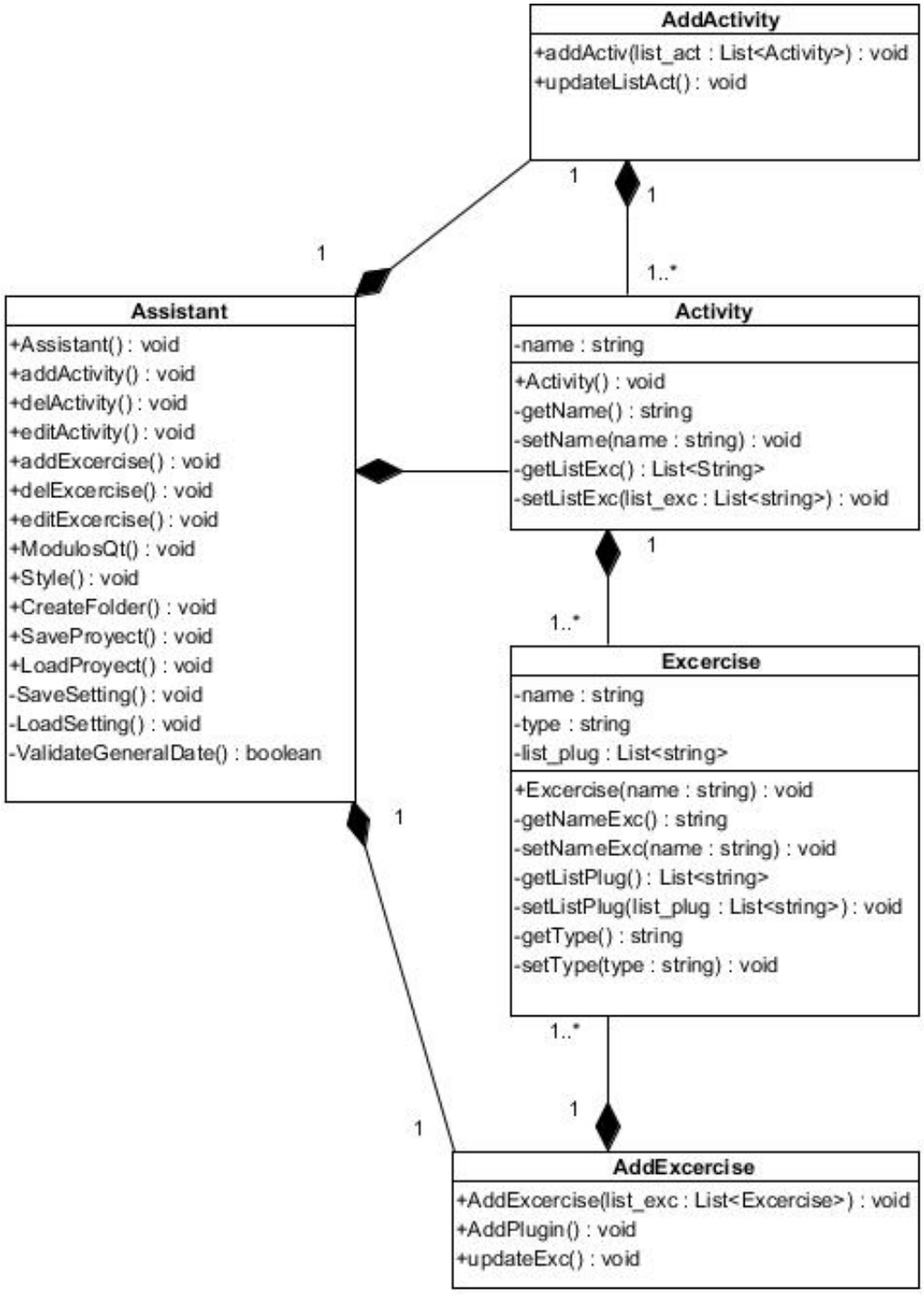


Figura 2.3. Diagrama de clases

2.14. Patrones de diseño

Los patrones de diseño no son más que una descripción detallada de las diferentes clases del sistema y de los objetos relacionados. Se encargan de resolver un problema de diseño general, en un determinado contexto [32]. Por lo que se puede decir, que un patrón de diseño es un conjunto de prácticas de óptimo diseño utilizado para abordar problemas recurrentes en la programación orientada a objetos. En la implementación de la aplicación se hará uso del siguiente patrón de diseño:

Patrones GRASP: Los Patrones Generales de Asignación de Responsabilidades de Software (GRASP, por sus siglas en inglés) describen los principios fundamentales de la asignación de responsabilidades a objetos, de forma tal, que se pueda diseñar *software* orientado a objetos en forma de patrones [33]. Estos patrones no introducen ideas novedosas, son la mera codificación de los principios básicos más usados. Los patrones GRASP están compuestos por: Experto, Creador, Bajo Acoplamiento, Alta Cohesión, Controlador. Estos principios básicos se tuvieron muy en cuenta en el momento de planificar el diseño de los componentes de la aplicación propuesta.

A partir del empleo del patrón GRASP se le asignaron responsabilidades a las clases. Basado en el diagrama de clases que se muestra en la Figura 2.3, las responsabilidades quedaron distribuidas de la siguiente manera:

- Experto: Las clases Activity y Excercise son las expertas en información. Cada clase se encarga de tener la información necesaria para cumplir determinadas responsabilidades. Estas clases tienen toda la información de los objetos actividad y ejercicio respectivamente.

- Creador: Las clases AddActivity y AddExcercise son las encargadas de crear instancias de las clases Activity y Excercise respectivamente. Por lo que son las responsables de crear una actividad y un ejercicio del laboratorio virtual.

- Bajo Acoplamiento: Existe un bajo acoplamiento de cada clase debido a la poca dependencia de ellas con otras clases.

- Alta Cohesión: Existe una colaboración entre las clases a partir de que cada una de ellas asume determinadas responsabilidades. Esto permite que se evite el exceso de responsabilidades de cada una de las clases. La clase Assistant realiza llamadas a AddActivity y AddExcercise siempre que se desee agregar una actividad o un ejercicio al laboratorio virtual. Mientras que AddActivity y AddExcercise realiza instancias de las clases Activity y Excercise respectivamente.

- Controlador: A la clase Assistant se le asigna la responsabilidad de las operaciones del sistema.

2.15. Consideraciones parciales

Con la conclusión del capítulo se describió la propuesta de solución, se seleccionaron las herramientas y tecnologías necesarias para el desarrollo de un *software* funcional. Se definieron los elementos necesarios referentes al análisis de la aplicación a desarrollar a partir del empleo de la metodología XP.

También se logró identificar 9 HU, divididas en 4 iteraciones una vez estimado el tiempo de duración de cada historia de usuario. Se realizó el cálculo del promedio de duración de las iteraciones a partir del grado de prioridad definida por el propio cliente y se realizó un plan de entregas que visualiza todas las diferentes versiones por la que transitará la aplicación hasta llegar a su versión final.

En el capítulo se ha realizado el diseño del *software*, donde se han descrito las responsabilidades que adquiere cada una de las clases que intervienen en el sistema a través de las tarjetas CRC y se ha confeccionado el diagrama de clases con las diferentes relaciones que existen entre ellas. Una vez terminado esta fase de análisis y diseño se comenzará la implementación de la solución en el capítulo siguiente.

CAPÍTULO 3

Implementación y prueba

3.1. Introducción

En el presente capítulo se realizan las fases de implementación y prueba del sistema. Como parte de la fase de implementación, se realiza la descripción de las tareas de ingeniería por cada historia de usuario a partir de las cuatro iteraciones planificadas. Al finalizar esta etapa se continuará con la fase de prueba, permitiendo que se realicen las pruebas de aceptación que permitan comprobar el correcto funcionamiento del sistema.

3.2. Fase de Implementación

En la etapa de implementación se realizarán las tareas relacionadas con la implementación de cada una de las HU correspondientes a cada iteración. Además se irá realizando una revisión del plan de iteraciones en caso de existir alguna posible modificación. A partir del desglose de cada historia de usuario, se le definieron un conjunto de tareas de programación o ingeniería que permitirán implementar exitosamente cada una de estas historias. A continuación se muestra de una forma más detallada las tareas a desarrollar para cada historia de usuario por iteración.

3.2.1. Iteración 1

Historias de Usuarios	Tareas por Historia de Usuario
Seleccionar ubicación del proyecto.	1. Permitir seleccionar la ubicación del nuevo proyecto.
Administrar actividades.	2. Permitir agregar una actividad al laboratorio. 3. Permitir eliminar una actividad del laboratorio. 4. Permitir editar una actividad del laboratorio.
Administrar ejercicios.	5. Permitir agregar un ejercicio a la actividad. 6. Permitir eliminar un ejercicio de la actividad. 7. Permitir editar un ejercicio de la actividad.

Tabla 3.1. Tarea de ingeniería # 1

Tarea	
Número de tarea: 1	**Número de Historia de usuario:** 1
Nombre de la tarea: Seleccionar la ubicación del nuevo proyecto.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 1
Fecha de inicio: 1 de enero de 2016	**Fecha de fin:** 7 de enero de 2016
Descripción: El usuario selecciona la ubicación del proyecto.	

Tabla 3.2. Tarea de ingeniería # 2

Tarea	
Número de tarea: 2	**Número de Historia de usuario:** 2
Nombre de la tarea: Agregar una actividad al laboratorio.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 0.5
Fecha de inicio: 8 de enero de 2016	**Fecha de fin:** 11 de enero de 2016
Descripción: El usuario agrega una actividad y se actualiza la lista de actividades.	

Tabla 3.3. Tarea de ingeniería # 3

Tarea	
Número de tarea: 3	**Número de Historia de usuario:** 2
Nombre de la tarea: Eliminar una actividad al laboratorio.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 0.25
Fecha de inicio: 12 de enero de 2016	**Fecha de fin:** 13 de enero de 2016
Descripción: El usuario podrá eliminar una actividad y se actualizará la lista de actividades.	

Tabla 3.4. Tarea de ingeniería # 4

Tarea	
Número de tarea: 4	**Número de Historia de usuario:** 2
Nombre de la tarea: Editar una actividad del laboratorio.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 0.25
Fecha de inicio: 14 de enero de 2016	**Fecha de fin:** 15 de enero de 2016
Descripción: El usuario podrá editar una actividad y se actualizará la lista de actividades.	

Tabla 3.5. Tarea de ingeniería # 5

Tarea	
Número de tarea: 5	**Número de Historia de usuario:** 3
Nombre de la tarea: Agregar un ejercicio a la actividad.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 0.5
Fecha de inicio: 16 de enero de 2016	**Fecha de fin:** 19 de enero de 2016
Descripción: El usuario podrá agregar un ejercicio a la actividad y se actualizará la lista de ejercicios de la actividad correspondiente.	

Tabla 3.6. Tarea de ingeniería # 6

Tarea	
Número de tarea: 6	**Número de Historia de usuario:** 3
Nombre de la tarea: Eliminar un ejercicio a la actividad.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 0.25
Fecha de inicio: 20 de enero de 2016	**Fecha de fin:** 21 de enero de 2016
Descripción: El usuario podrá eliminar un ejercicio a la actividad y se actualizará la lista de ejercicios de la actividad correspondiente.	

Tabla 3.7. Tarea de ingeniería # 7

Tarea	
Número de tarea: 7	**Número de Historia de usuario:** 3
Nombre de la tarea: Editar un ejercicio.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 0.25
Fecha de inicio: 22 de enero de 2016	**Fecha de fin:** 23 de enero de 2016
Descripción: El usuario podrá editar un ejercicio de la actividad y se actualizará la lista de ejercicios de la actividad correspondiente.	

3.2.2. Iteración 2

Historias de Usuarios	Tareas por Historia de Usuario
Seleccionar módulos.	8. Permitir seleccionar los módulos de Qt.
Seleccionar estilos.	9. Permitir seleccionar los estilos.

Tabla 3.8. Tarea de ingeniería # 8

Tarea	
Número de tarea: 8	**Número de Historia de usuario:** 4
Nombre de la tarea: Seleccionar los módulos de Qt.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 1
Fecha de inicio: 24 de enero de 2016	**Fecha de fin:** 30 de enero de 2016
Descripción: El usuario seleccionará los módulos de Qt a utilizar en el laboratorio virtual.	

Tabla 3.9. Tarea de ingeniería # 9

Tarea	
Número de tarea: 9	**Número de Historia de usuario:** 5
Nombre de la tarea: Seleccionar los estilos.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 1
Fecha de inicio: 1 de febrero de 2016	**Fecha de fin:** 7 de febrero de 2016
Descripción: El usuario seleccionará el estilo del laboratorio virtual.	

3.2.3. Iteración 3

Historias de Usuarios	Tareas por Historia de Usuario
Generar estructura de carpetas y ficheros.	10. Crear una estructura de carpetas. 11. Crear una estructura de ficheros.
Configurar el contenido de los ficheros.	12. Configurar el contenido de los ficheros.

Tabla 3.10. Tarea de ingeniería # 10

Tarea	
Número de tarea: 10	**Número de Historia de usuario:** 6
Nombre de la tarea: Crear una estructura de carpetas.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 0.5
Fecha de inicio: 8 de febrero de 2016	**Fecha de fin:** 11 de febrero de 2016
Descripción: Se genera la estructura de carpetas del proyecto.	

Tabla 3.11. Tarea de ingeniería # 11

Tarea	
Número de tarea: 11	**Número de Historia de usuario:** 6
Nombre de la tarea: Crear una estructura de ficheros.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 0.5
Fecha de inicio: 12 de febrero de 2016	**Fecha de fin:** 15 de febrero de 2016
Descripción: Se genera la estructura de ficheros del proyecto.	

Tabla 3.12. Tarea de ingeniería # 12

Tarea	
Número de tarea: 12	**Número de Historia de usuario:** 7
Nombre de la tarea: Configurar el contenido de los ficheros.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 2
Fecha de inicio: 16 de febrero de 2016	**Fecha de fin:** 28 de febrero de 2016
Descripción: Se configura el contenido de los ficheros del proyecto.	

3.2.4. Iteración 4

Historias de Usuarios	Tareas por Historia de Usuario
Guardar proyecto.	13. Guardar la configuración del proyecto.
Cargar proyecto.	14. Permitir cargar la configuración de algún proyecto.

Tabla 3.13. Tarea de ingeniería # 13

Tarea	
Número de tarea: 13	**Número de Historia de usuario:** 8
Nombre de la tarea: Guardar la configuración del proyecto.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 1
Fecha de inicio: 29 de febrero de 2016	**Fecha de fin:** 6 de marzo de 2016
Descripción: El usuario podrá guardar la configuración del proyecto en un fichero XML.	

Tabla 3.14. Tarea de ingeniería # 14

Tarea	
Número de tarea: 14	**Número de Historia de usuario:** 9
Nombre de la tarea: Cargar la configuración del proyecto.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 1
Fecha de inicio: 7 de marzo de 2016	**Fecha de fin:** 13 de marzo de 2016
Descripción: El usuario podrá cargar la configuración de algún proyecto existente.	

3.3. Descripción de la solución implementada

El generador de plantillas para VirtualLabSDK, se encarga de generar las carpetas y ficheros que utiliza la estructura de un proyecto. Además de que se encargará de configurar cada fichero de una forma dinámica. Esta configuración permitirá que el sistema sea flexible a cualquier cambio en cuanto al empleo de nuevas herramientas o tecnologías en el desarrollo de un laboratorio virtual. Para lograr alcanzar este dinamismo fue necesario el empleo de la tecnología XML y del uso de las etiquetas de reemplazo. A continuación se muestra un ejemplo más detallado.

3.3.1. Empleo del lenguaje de marcado XML

```
<plugin Name="pantool" Title="Panel" Folder="pan_tool" File_d="Plug_Pan_d" File_r="Plug_Pan">
<levels>
  <level Name="Ogre"/>
 </levels>
<depend>
 <component Name="readsceneinformation" Title="Lector de informacion de la escena"/>
 <component Name="loadscene" Title="Cargador de escenas"/>
</depend>
<include Name="PanelToolComponentProvider.h"/>
<class Name="PanelToolComponentProvider"/>
<var Value="0" Name="panel"/>
<methods>
 <method Type="public slots" Class="Level" Name="loadMesh" Return="void" Level="ogre">
  <arguments>
   <argument Type="QString" Name="name"/>
  </arguments>
  <implement>
   loadScene->getAllNodesFromLastScene();
  </implement>
 </method>
 <method Type="public slots" Class="Level" Name="loadMesh" Return="void" Level="qt">
  <arguments>
   <argument Type="QString" Name="name"/>
  </arguments>
  <implement>
  </implement>
 </method>
 <method Type="public" Class="Level" Name="ConfigurePanel" Return="void" Level="ogre">
  <implement>
   mainWindow->addDockWidget(Qt::RightDockWidgetArea,panel);
  </implement>
 </method>
</methods>
<codes>
 <code Class="Level" Method="createscene" Level="ogre">ConfigurePanel();</code>
</codes>
<initialise Value=" "/>
<connect>
 <sender Name="Sender">panel</sender>
 <signal Name="Signal">pressed(QString)</signal>
 <context Name="Context">this</context>
 <slots Name="Slots">loadMesh(QString)</slots>
</connect>
```

Figura 3.1. Ejemplo de un documento XML de un plugin.

Cada *plugin* existente en el proyecto tendrá asociado un fichero XML. Dicho fichero contendrá toda la información necesaria para que el *plugin* pueda ser configurado. En la Figura 3.1 se muestra un ejemplo de la estructura del fichero XML para un determinado *plugin*. Para lograr una mayor comprensión del mismo , se explicará la estructura del fichero por sus etiquetas principales.

plugin: Es la etiqueta raíz , contiene un grupo de atributos que lo identifican como nombre, el título que tendrá en la interfaz de selección de *plugins*, el nombre de la carpeta donde se encuentra ubicado en el *framework* de VirtualLabSDK , y el nombre de los ficheros empleados en *debug* y *release*.

levels: Contiene los niveles *(levels)* o tipos de ejercicios en los que dicho *plugin* puede ser empleado. De no poder ser utilizado en un nivel *(level)* seleccionado, el mismo se desmarcará y se bloqueará, impidiendo ser seleccionado por el usuario. En la siguiente Figura 3.2 se muestra un ejemplo más detallado.

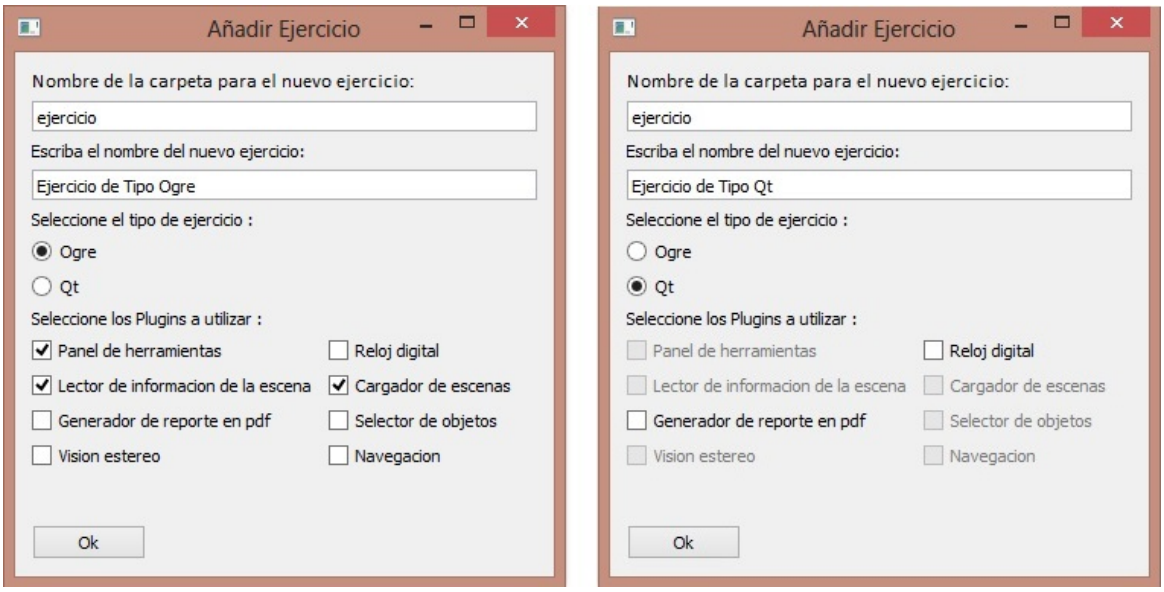


Figura 3.2. Ejemplo para agregar ejercicios de diferente tipo o nivel *(level)*

depend: Debido a que pueden existir *plugins* que dependan de otros. La etiqueta contiene un grupo de etiquetas el cual cada una de ellas representa cada *plugin* del que depende. De esta manera al seleccionar dicho *plugin*, se marcaran de forma automática los *plugins* de los cual depende. En la primera imagen, de izquierda a derecha de la Figura 3.2, se puede observar como al marcar el *plugin* del Panel de herramientas se seleccionará los otros dos *plugins* de los cual él depende.

include: Contiene el nombre del *plugin* para poder ser incluido en el proyecto.

class: Contiene el nombre de la clase del *plugin.*

var: Contiene el nombre y el valor de la variable con que se inicializará en el constructor del proyecto.

methods: Contiene una lista de todos los métodos del propio *plugin*. Cada método es representado por una etiqueta que lo identifica y por atributos que definen la estructura del método correspondiente. Para

lograr esta estructura se le incorporaron etiquetas que representan los argumentos *(arguments)* y la implementación *(implement)* del método.

codes: Contiene una lista de fragmentos de código *(code)* con el nombre, la clase y el nivel al que pertenece dicho código.

initialise: Contiene el valor de inicialización del componente.

connect: Contiene otras etiquetas que le brindarán la estructura de la conección a utilizar al inicializar el componente.

El usuario podrá modificar o incorporar un nuevo *plugin* al proyecto con solo modificar o crear un nuevo fichero XML para el nuevo *plugin*.

```
<modulos>
 <modulo Name="sql" Title="SQL library"/>
 <modulo Name="qml" Title=" QML library"/>
 <modulo Name="network" Title="Network library"/>
 <modulo Name="quick" Title="Quick library"/>
 <modulo Name="multimedia" Title="Multimedia library"/>
 <modulo Name="multimediawidgets" Title="Multimedia Widgets library"/>
 <modulo Name="webkit" Title="Web Kit library"/>
 <modulo Name="webkitwidgets" Title="Web Kit Widgets library"/>
 <modulo Name="testlib" Title="Test library"/>
 <modulo Name="gui" Title="GUI library"/>
 <modulo Name="core" Title="Core library"/>
 <modulo Name="widgets" Title="Widgets Library "/>
</modulos>
```

Figura 3.3. Documento XML que contiene los módulos de Qt.

Como parte de lograr una buena adaptabilidad del *software* a la hora de incorporar nuevos módulos de Qt, se decidió emplear un documento XML. Como se muestra en la Figura 3.3, el documento quedó estructurado de la siguiente manera:

modulos: Es la etiqueta raíz que contiene la lista de módulos que forman parte del proyecto. Cada módulo es definido por un nombre y por el título que tendrá en la interfaz de selección de los módulos. El usuario podrá incorporar un nuevo módulo al proyecto con solo incorporar una etiqueta en la lista de módulos.

El empleo del lenguaje de marcado XML no solo fue empleado para lograr una mejor adaptabilidad del sistema en cuanto a posibles cambios en el futuro. En base a la idea tomada por Maven de utilizar un fichero XML para la configuración del proyecto. Fue necesario crear un fichero que contenga toda la información básica del proyecto, permitiendo que esta configuración pueda ser guardada o cargada por el usuario. En la Figura 3.4 se muestra cómo quedaría conformado el fichero de configuración, con la estructura siguiente:

laboratorio: Es la etiqueta raíz , contiene un grupo de atributos que lo identifican como el nombre del proyecto, la ruta donde se encuentra ubicado y el estilo a utilizar.

actividades: Un proyecto contiene un grupo de actividades. Por lo que esta etiqueta contendrá todas las actividades con los ejercicios correspondientes a cada actividad. Además, cada ejercicio tendrá asociado los *plugins* que serán empleados en dicho ejercicio.

```
<laboratorio Name="Laboratorio" href="C:/Users/Arnaldo/Proyecto" Style="xauce">
 <actividades>
  <actividad Name="activ_1" Title="Actividad número 1">
   <ejercicios>
    <ejercicio Name="ejerc_1" Title="Ejercicio" Tipo="ogre">
     <plugins>
      <plugin Name="digitalClock" Title="Reloj digital"/>
      <plugin Name="loadscene" Title="Cargador de escenas"/>
      <plugin Name="meshSelector" Title="Selector de objetos"/>
      <plugin Name="navigation" Title="Navegacion"/>
      <plugin Name="paneltool" Title="Panel de herramientas"/>
      <plugin Name="reportpdf" Title="Generador de reporte en pdf"/>
      <plugin Name="stereoVision" Title="Vision estereo"/>
     </plugins>
    </ejercicio>
   </ejercicios>
  </actividad>
 </actividades>
 <modulos>
  <modulo Name="webkitwidgets" Title="Web Kit Widgets library"/>
  <modulo Name="testlib" Title="Test library"/>
  <modulo Name="gui" Title="GUI library"/>
  <modulo Name="core" Title="Core library"/>
  <modulo Name="widgets" Title="Widgets Library "/>
 </modulos>
</laboratorio>
```

Figura 3.4. Fichero de configuración del proyecto.

3.3.2. Empleo de las etiquetas de reemplazo

Para modificar el código fuente de cada uno de los ficheros asociados al laboratorio virtual, fueron empleadas las etiquetas de reemplazo. Las etiquetas se estructuraron con un nombre que lo identifique, cerrado entre corchetes. El nombre de la etiqueta constituye un texto que identifica de una forma clara y sencilla, el elemento que posteriormente reemplazará a la etiqueta. De no existir ningún elemento correspondiente a la etiqueta en cuestión, esta será reemplazada por un caracter vacío. De este modo se evitará que aparezcan las etiquetas en el código fuente del laboratorio virtual. A continuación, se mostrarán algunos ejemplos más detallados de su utilización en la solución del problema.

```
2   #include <coreVLab.h>
3
4   [INCLUDES]
5
6   #include <QEvent>
7   #include <QWidget>
8
9   class Level : public coreVLab::QtWidgetEventHandler
10  {
11      Q_OBJECT
12
13  private:
14
15      [PLUGINS]
16
17  public slots:
18
19      [Level::public slots]
20
21  public:
22      Level();
23      ~Level(void);
24
25      void InitialiseComponents();
26
27      [Level::public]
28
29      void createscene();
30  };
```

Figura 3.5. Ejemplo de código fuente de un fichero (.h) asociado al laboratorio virtual.

En la Figura 3.5 se muestra un ejemplo del empleo de las etiquetas de reemplazo en el código fuente de un fichero (.h) asociado a un proyecto. Para lograr una mayor comprensión, se explicará la forma en que fueron aplicadas cada etiqueta.

***INCLUDES* :** Es la etiqueta que será sustituida por la estructura de las inclusiones de cada uno de los *plugins* empleados en el proyecto.

***Level::public slots* :** El nombre de la etiqueta está compuesta por el nombre de la clase seguido de cuatro puntos con el tipo de método a continuación. De esta forma, se podrá identificar exactamente que la etiqueta será sustituida por la estructura de declaración de aquellos métodos que sean de la clase y del tipo que ella define.

***Level::public* :** Cumple la misma función de la etiqueta antes mencionada. Esta etiqueta será sustituida por la estructura de declaración de los métodos que sean de la clase y del tipo que ella define.

```
#include "ExampleLevel.h"

    [Level::Methods]

Level::Level(): coreVLab::OgreWidgetEventHandler()
{
    [PLUGINS]

    mGuiMgr = 0;
    loadingBar->show();
}

void Level::InitialiseComponents()
{
    coreVLab::ComponentCore::getSingleton()->loadComponents();

    [Components]
}

void Level::onKeyPress(QKeyEvent *event)
{
    [Level::onKeyPress]

    if(event->key() == Qt::Key_Escape)
    {
        ((coreVLab::vlMainWindows*)mainWindow)->stopLevel();
    }

    event->ignore();
}

void Level::onMousePress(QMouseEvent *event)
{
    mGuiMgr->injectMouseDown(event);

    [Level::onMousePress]
}
```

Figura 3.6. Ejemplo de código fuente de un fichero (.cpp) asociado al laboratorio virtual.

En la Figura 3.6 se muestra un ejemplo del empleo de las etiquetas de reemplazo en el código fuente de un fichero (.cpp) asociado a un proyecto. Su uso sigue el mismo principio de lo mencionado anteriormente sobre la Figura 3.5. A continuación se definirán cada una de las etiquetas representadas en la Figura 3.6.

***Level::Methods* :** Es la etiqueta que será sustituida por todos los métodos de cada uno de los *plugins* empleados en el proyecto.

***PLUGINS* :** Es la etiqueta que será sustituida por el nombre de la variable y el valor de cada *plugin* empleado en el proyecto.

***Components* :** Es la etiqueta que será sustituida por la estructura de inicialización de componentes para cada *plugin*.

***Level::onKeyPress* :** El nombre de la etiqueta está compuesta por el nombre de la clase seguido de cuatro puntos con el nombre del método a continuación. De esta forma se podrá identificar exactamente que la etiqueta será sustituida por el fragmento de código definida por la clase y el nombre del método que se representa. Esto permitirá que cualquier *plugins* que contenga el fragmento correspondiente al método de la clase, podrá ser añadido.

***Level::onMousePress* :** Cumple la misma función de la etiqueta antes mencionada. La etiqueta será sustituida por el fragmento de código de la clase y método que ella define.

Como parte de lograr cierto dinamismo en cuanto a la incorporación de nuevos métodos y a su vez nuevas etiquetas al fichero asociado al laboratorio virtual. Fue necesario el empleo de una expresión regular que permita identificar todas aquellas etiquetas que sean incorporadas y que cumplan con la siguiente estructura definida para los métodos:

[*Level::*nombre del método]**.

Una expresión regular representa una secuencia de caracteres que forman un patrón de búsqueda. Es muy factible su uso para reconocer cadenas de textos. Por lo que fue necesario su empleo para reconocer las etiquetas de reemplazo en los ficheros de configuración. A continuación se muestar la expresión regular utilizada para las etiquetas de reemplazo correspondiente a los métodos del código fuente de los ficheros asociados al proyecto.

ExpReg=$(\backslash[Level :: \backslash w + [\backslash s + \backslash w+] * \backslash])$

$[\ ]$: Agrupar secuencia de caracteres.
$\backslash[$: Reconocer caracter especial corchete abierto.
$\backslash]$: Reconocer caracter especial corchete cerrado.
Level :: : Reconocer la cadena **Level::** .
$\backslash w$: Reconocer cualquier caracter alfanumérico.
$\backslash s$: Reconocer un espacio en blanco.
$+$: Reconocer cualquier combinación de caracteres desde uno a mucho.
$*$: Reconocer cualquier combinación de caracteres desde cero a mucho.

En las siguientes figuras se ilustrarán cómo quedarían los ejemplos de las Figuras 3.5 y 3.6 después de reemplazadas las etiquetas.

```
#include <DigitalClockComponentProvider.h>
#include <LoadSceneComponentProvider.h>
#include <MeshSelectorComponentProvider.h>

class Level : public coreVLab::QtWidgetEventHandler
{
    Q_OBJECT

private:

   DigitalClockComponentProvider* digitalClock;
   LoadSceneComponentProvider* loadScene;
   MeshSelectorComponentProvider* meshSelect;

public slots:

     void loadMesh(QString name);

public:
    Level();
    ~Level(void);

    void InitialiseComponents();

    void ConfigurePanel();

    void createscene();
};
```

Figura 3.7. Sustitución de las etiquetas de reemplazo en el ejemplo de la Figura 3.5.

```
void Level::loadMesh(QString name)
{
 loadScene->LoadScene(name,false,false,false);
}
void Level::ConfigurePanel()
{
 QList <QString> temp=read->readForInformation("../media/sample_sc","sp");
 QListIterator <QString> listItr(temp);
}
Level::Level(): coreVLab::OgreWidgetEventHandler()
{
    meshSelect = 0;
    panel = 0;
    mGuiMgr = 0;
    loadingBar->show();
}
void Level::InitialiseComponents()
{
 coreVLab::ComponentCore::getSingleton()->loadComponents();
 meshSelect= static_cast<MeshSelectorComponentProvider*>;
 if(meshSelect == 0)
 {
Ogre::LogManager::getSingletonPtr()->logMessage("No se encuentra meshSel");
  exit(0);
 }
 meshSelect->initialise(mSceneMgr);
 panel= static_cast<PanelToolComponentProvider*>;
 if(panel == 0)
 {
  Ogre::LogManager::getSingletonPtr()->logMessage("No se encuentra panel");
   exit(0);
 }
 panel->initialise( );
  connect(panel,SIGNAL(pressed(QString)),this,SLOT(loadMesh(QString)));
}
void Level::onMousePress(QMouseEvent *event)
{
     mGuiMgr->injectMouseDown(event);
     meshSelect->selectMesh( mCamera,event->x(),event->y() );
}
```

Figura 3.8. Sustitución de las etiquetas de reemplazo en el ejemplo de la Figura 3.6.

3.3.3. Descripción de la solución en la selección de estilos

A partir de que actualmente el desarrollo de proyectos relacionados con los laboratorios virtuales en la UCI se realiza sobre las líneas de desarrollo de salud y educación. Cada línea de desarrollo tiene asociado su propio estilo de identidad marcaria. Para desarrollar un laboratorio virtual hay que tener presente este estilo de diseño en función de la línea sobre la que se desea desarrollar el proyecto. Como parte de que el sistema permita la incorporación de nuevos estilos, se decidió realizar la selección de los mismos de una

forma dinámica. En las siguientes figuras se ilustra cómo se describe el proceso de selección de estilos.

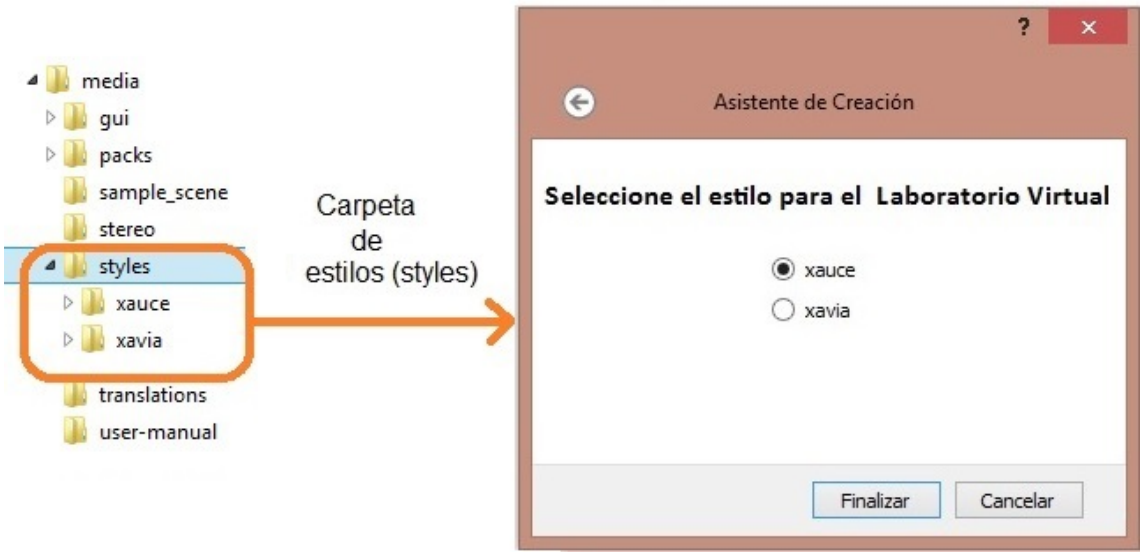


Figura 3.9. Estilos xauce y xavia existentes en la selección.

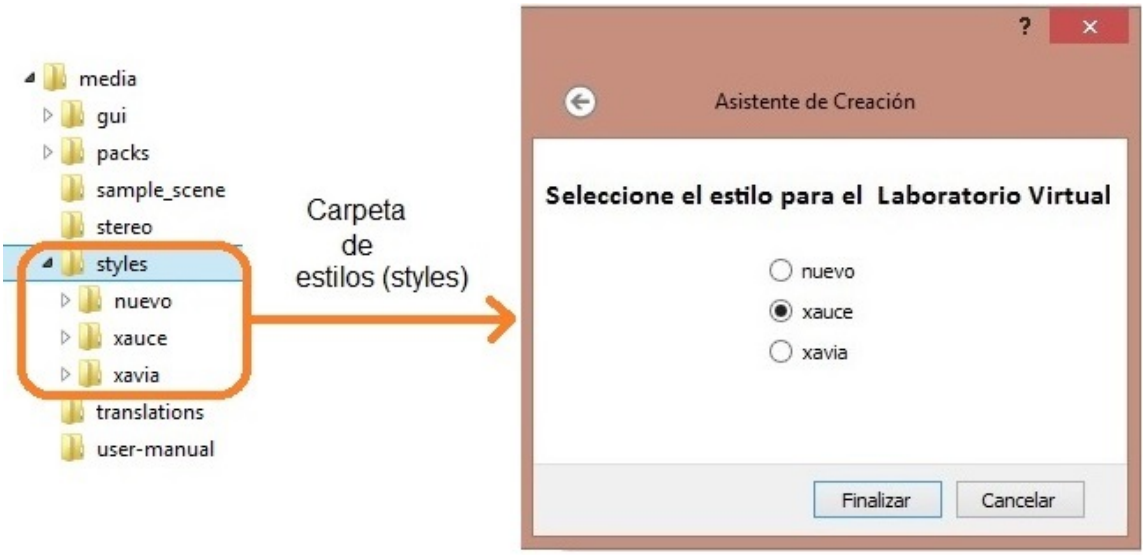


Figura 3.10. Incorporación de un nuevo estilo a la selección.

Cómo se muestra en las dos figuras anteriores se decidió tener una carpeta predefinida para los estilos. En esta carpeta estarán ubicadas todas las carpetas de estilos que puedan ser utilizadas. De incorporarse uno nuevo como se ilustra en la Figura 3.10, solo se tendría que agregar la carpeta del nuevo estilo a la carpeta donde se encuentran todos los estilos del proyecto. De este modo se actualizaría la interfaz de selección de estilos del asistente de creación de proyectos. En la Figura 3.9 se muestran los estilos que actualmente existen en la UCI para el desarrollo de laboratorios virtuales. Mientras que en la Figura 3.10 se muestra la

interfaz de selección de estilos después de haber agregado uno nuevo.

3.3.4. Validación de la solución

El generador de plantillas se encarga de generar un nuevo proyecto a partir de un conjunto de datos introducidos por el propio usuario. La herramienta brinda la posibilidad de asignarle una dirección y un nombre al laboratorio que se desee crear. También permite adicionar una lista de actividades en el que a cada actividad se le podrá asignar un grupo de ejercicios. A cada ejercicio el usuario tendrá la posibilidad de seleccionarle su tipo y el grupo de plugins que necesite utilizar. Con el empleo del sistema también se facilitará la selección de módulos y del estilo de diseño a emplear en el nuevo laboratorio. La aplicación brinda la facilidad de que la configuración de cada proyecto generado pueda ser guardado en un fichero XML con el objetivo de que dicho proyecto, pueda ser modificado en un futuro. El generador permite mostrar y seleccionar de forma rápida la configuración de los cuatro proyectos más recientes que han sido generados. Para que se puedan incorporar nuevas herramientas y componentes la aplicación se adapta facilmente al cambio, logrando una mayor usabilidad del mismo. Con el objetivo de medir aproximadamente la utilidad de la aplicación desarrollada se realizó un experimento interno en el centro VERTEX, donde la situación escogida fue la siguiente:

El experimento consistirá en crear un laboratorio virtual de forma manual. Para realizar la tarea se deberá modificar un laboratorio ya existente que contiene 1 actividad con 1 ejercicio. Se modificará solamente el nombre de la actividad y del ejercicio. No serán incorporados *plugins*, módulos, tipo de ejercicio y estilo debido a que se creará un nuevo proyecto con los mismos elementos del que será modificado. Con la realización del experimento se quiere ver la utilidad del sistema a partir de un proyecto simple, que no requiera de muchos cambios. El experimento fue realizado en 4 ocasiones y el resultado arrojado fue el siguiente:

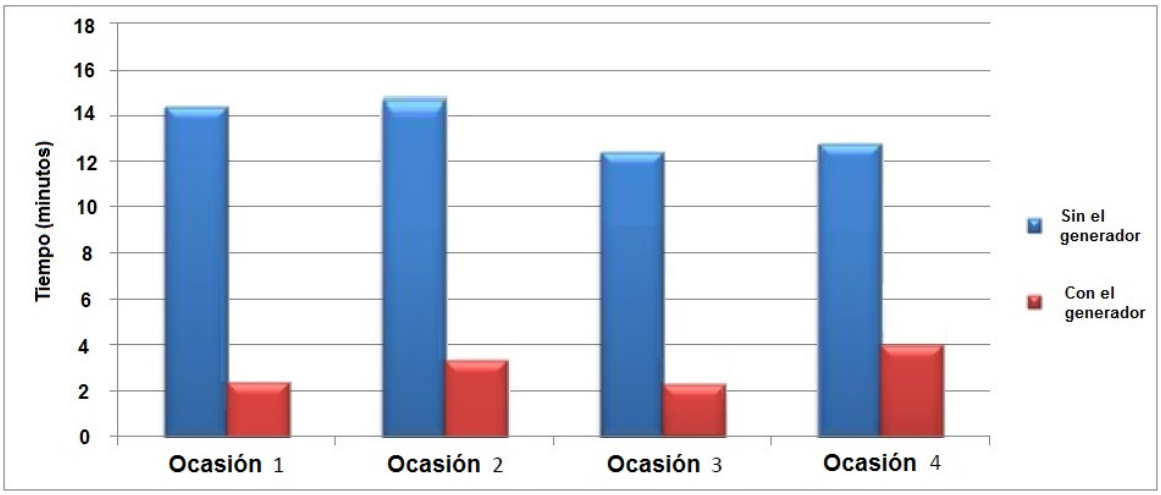


Figura 3.11. Representación del resultado del experimento.

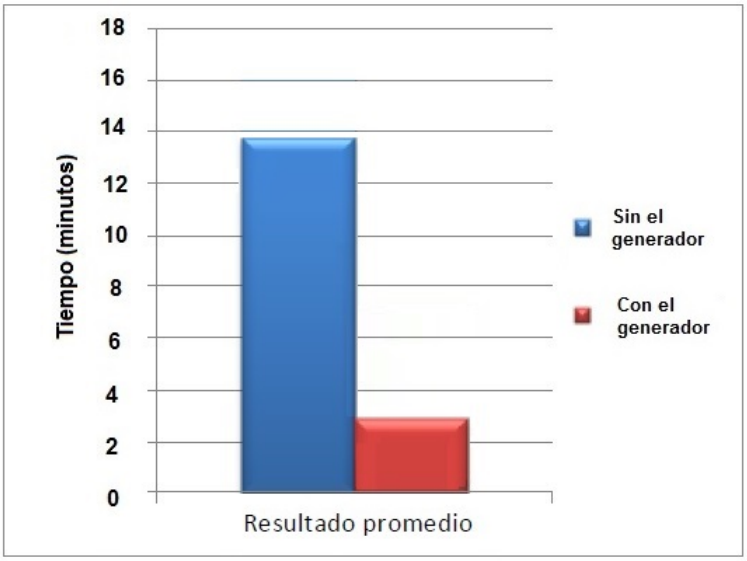


Figura 3.12. Representación del tiempo promedio.

En la Figura 3.12 se puede observar el resultado promedio de los tiempos de creación de un proyecto de forma manual y con el empleo del generador de plantillas. Luego de concluido el experimento se puede observar que el tiempo promedio sin emplear el generador es de 12,63 minutos y empleando el generador es de solo 3 minutos lo que demuestra una disminución de un 77 % de los tiempos de desarrollo de un laboratorio virtual. A partir del experimento con un ejemplo sencillo se puede concluir que la disminución del tiempo podría ser mayor a medida que el proyecto sea más complejo en cuanto a cantidad de actividades, ejercicios y *plugins*.

3.4. Pruebas

Son un conjunto de actividades planificadas con anticipación para la detección de errores en un *software*. El *software* se prueba para descubrir errores cometidos al realizar su diseño e implementación [34]. La metodología XP le da gran importancia a la fase de pruebas, posibilitando que se realicen dichas pruebas constantemente. Esto es una gran ventaja, ya que permite que se reduzca el número de errores no detectados, aumentando de este modo la calidad del sistema. Otra de la ventaja que brinda, es la seguridad en cuanto a evitar efectos colaterales no deseados. La metodología XP divide las pruebas del sistema en dos grupos: pruebas unitarias, encargadas de verificar el código y diseñada por los programadores y pruebas de aceptación o pruebas funcionales destinadas a evaluar si al final de cada iteración se consiguió la funcionalidad requerida por el cliente [35].

3.4.1. Pruebas de aceptación

Las pruebas de aceptación tienen mayor importancia que las pruebas unitarias, dado que significan la satisfacción del cliente con el producto desarrollado. Estas pruebas permiten comprobar el cumplimiento de las funcionalidades requeridas por el propio cliente, por lo que sería el encargado de diseñar dichas pruebas que a la vez puedan ser ejecutadas por la máquina [36]. El diseño de las mismas se basa en la evaluación de las funcionalidades de cada historia de usuario. Esta evaluación debe comprobar que el resultado sea el esperado. En caso de no ser el resultado que se espera, se realizará una corrección de las mismas y se tomarán las medidas pertinentes para obtener los resultados que se esperan. A continuación se detallan las pruebas de aceptación de la aplicación desarrollada:

Tabla 3.15. Prueba de aceptación # 1

Caso de prueba de aceptación	
Código: HU1_P1	**Historia de usuario:** 1
Nombre: Seleccionar una ruta válida para el proyecto.	
Descripción: Según la ruta seleccionada por el usuario, obtener la ubicación del proyecto.	
Condiciones de ejecución: • La ruta seleccionada por el usuario debe ser válida. • El proyecto requiere de un nombre • El nombre del proyecto no debe tener espacios.	
Pasos de ejecución: El usuario introduce un nombre al proyecto y selecciona su ubicación.	
Resultados esperados: El usuario selecciona correctamente la ubicación del proyecto.	

Tabla 3.16. Prueba de aceptación # 2

Caso de prueba de aceptación	
Código: HU2_P2	**Historia de usuario:** 2
Nombre: Administrar las actividades del proyecto.	
Descripción: El usuario agrega,elimina y edita las actividades del proyecto	
Condiciones de ejecución: • El nombre de la carpeta de la actividad no debe tener espacios.	
Pasos de ejecución: El usuario agrega una actividad, edita sus datos y a continuación la elimina	
Resultados esperados: El usuario administra correctamente la actividad.	

Tabla 3.17. Prueba de aceptación # 3

Caso de prueba de aceptación	
Código: HU3_P3	**Historia de usuario:** 3
Nombre: Administrar los ejercicios del proyecto.	
Descripción: El usuario agrega,elimina y edita el ejercicio correspondiente a la actividad del proyecto	
Condiciones de ejecución: • El nombre de la carpeta del ejercicio no debe tener espacios. • Debe existir al menos una actividad en el proyecto.	
Pasos de ejecución: El usuario agrega un ejercicio, edita sus datos y a continuación lo elimina	
Resultados esperados: El usuario administra correctamente el ejercicio.	

Tabla 3.18. Prueba de aceptación # 4

Caso de prueba de aceptación	
Código: HU4_P4	**Historia de usuario:** 4
Nombre: Seleccionar los módulos para el proyecto.	
Descripción: El usuario podrá seleccionar los módulos en la interfaz de selección de módulos de Qt.	
Condiciones de ejecución: • Debe existir el fichero XML con la información de los módulos.	
Pasos de ejecución: El usuario selecciona todos los módulos en la interfaz de selección de módulos de Qt.	
Resultados esperados: El usuario selecciona correctamente los módulos existente en el fichero XML de los módulos del proyecto.	

Tabla 3.19. Prueba de aceptación # 5

Caso de prueba de aceptación	
Código: HU5_P5	**Historia de usuario:** 5
Nombre: Seleccionar el estilo para el proyecto.	
Descripción: El usuario podrá seleccionar los estilos en la interfaz de selección de estilos.	
Condiciones de ejecución: • Debe existir la carpeta con la información de cada uno de los estilos.	
Pasos de ejecución: El usuario selecciona el estilo en la interfaz de selección de estilos.	

Continúa en la próxima página

Tabla 3.19. Continuación de la página anterior

Resultados esperados: El usuario selecciona correctamente el estilo de los existentes en la carpeta de los estilos.

Tabla 3.20. Prueba de aceptación # 6

Caso de prueba de aceptación	
Código: HU6_P6	**Historia de usuario:** 6
Nombre: Genera toda la estructura de carpetas y ficheros del proyecto.	
Descripción: El usuario seleccionará los elementos necesario para que la aplicación genere la estructura de carpetas y ficheros.	
Condiciones de ejecución: • Debe seleccionarse la ubicación del proyecto. • Debe existir al menos una actividad con al menos un ejercicio.	
Pasos de ejecución: El usuario introduce y selecciona todos los elementos a tener en cuenta en el proyecto.	
Resultados esperados: A partir de los elementos seleccionados por el usuario se genera correctamente la estructura de carpetas y ficheros.	

Tabla 3.21. Prueba de aceptación # 7

Caso de prueba de aceptación	
Código: HU7_P7	**Historia de usuario:** 7
Nombre: Configurar el contenido de los ficheros del proyecto.	
Descripción: El usuario seleccionará los elementos necesario para que la aplicación modifique el contenido de los ficheros.	
Condiciones de ejecución: • Debe seleccionarse la ubicación del proyecto. • Debe existir al menos dos actividades con al menos dos ejercicios. • Cada ejercicio debe ser de tipo diferente. • Debe seleccionarse todos los *plugins* al menos de un ejercicio. • Debe seleccionarse todos los módulos de Qt.	
Pasos de ejecución: El usuario introduce y selecciona todos los elementos a tener en cuenta en el proyecto.	
Resultados esperados: A partir de los elementos seleccionados por el usuario se configura correctamente el contenido de los ficheros.	

Tabla 3.22. Prueba de aceptación # 8

Caso de prueba de aceptación	
Código: HU8_P8	**Historia de usuario:** 8
Nombre: Guardar toda la configuración del proyecto.	
Descripción: El usuario guardará la configuración del proyecto en un archivo XML.	
Condiciones de ejecución: • Debe seleccionarse la ubicación del archivo de configuración.	
Pasos de ejecución: El usuario guarda la configuración en un archivo XML, de todos los elementos tenidos en cuenta para crear el proyecto.	
Resultados esperados: Se guardó correctamente toda la configuración del proyecto en el archivo XML.	

Tabla 3.23. Prueba de aceptación # 9

Caso de prueba de aceptación	
Código: HU9_P9	**Historia de usuario:** 9
Nombre: Cargar toda la configuración de un proyecto existente.	
Descripción: El usuario cargará un archivo XML con toda la configuración de un proyecto.	
Condiciones de ejecución: • Debe cargar un archivo XML con la configuración de un proyecto existente.	
Pasos de ejecución: El usuario carga un archivo XML con la configuración de un proyecto existente.	
Resultados esperados: Se cargó correctamente toda la configuración del proyecto.	

3.4.2. Consideraciones parciales

Con la asignación de las tareas de implementación se logró detallar mejor las HU y así alcanzar una mayor organización en el desarrollo del sistema. Después de una buena distribución de las tareas, se pudo llevar a cabo el empleo de métodos, herramientas y tecnologías para la realización de una aplicación lo más dinámica posible. A continuación, se ejecutó una fase de pruebas, en la cual se realizaron 9 pruebas de aceptación a partir de las HU divididas en 4 iteraciones. Estas pruebas arrojaron diversas no conformidades que posteriormente fueron corregidas ayudando a mejorar la aplicación final.

Finalmente el producto desarrollado podrá ser utilizado en el proceso de creación de laboratorios virtuales sobre el *framework* VirtualLabSDK.

Conclusiones

Con la culminación de la presente investigación, se logró dar cumplimiento a los objetivos trazados, por lo que se arribaron a las siguientes conclusiones:

1. El generador de plantillas desarrollado, permitió minimizar considerablemente los tiempos de inicio de desarrollo de un laboratorio virtual, sobre la base de VirtualLabSDK.

2. El empleo del lenguaje de marcado XML y de las etiquetas de reemplazo, facilitaron el dinamismo de la aplicación. Este dinamismo permitirá que el propio sistema se adapte fácilmente al cambio, en cuanto a la incorporación de nuevas herramientas o tecnologías en el desarrollo de laboratorios virtuales.

3. El generador de plantillas convertirá el proceso de inicio de creación de un laboratorio virtual, en un proceso menos engorroso. Por lo que dicha herramienta generará todo el contenido y la estructura de un proyecto sin necesidad de realizar el proceso de creación de un laboratorio virtual de forma manual.

Recomendaciones

Como resultado en la realización de la presente investigación se recomiendan los siguientes aspectos a tener en cuenta para un futuro perfeccionamiento del sistema:

1. Agregar al asistente una página que permita seleccionar herramientas de VirtualLabSDK como el editor de tareas y de materiales.
2. Incorporar un mecanismo que agilice el proceso de administrar el archivo XML correspondiente a los módulos y al de cada *plugin*.

Referencias

1. **UNED**. [En línea] http://lab.dia.uned.es/rlab/contenido/labvirtual.html?page=3
2. **Merzeau, Martínez, Alejandro David**. *Arquitectura de software para los Laboratorios Virtuales*. Trabajo de diploma para optar por el título de ingeniero en ciencias informáticas. Ciudad de La Habana 2012.
3. **Rosado, L and Herreros, J R**. *Nuevas aportaciones didácticas de los laboratorios virtuales y remotos en la enseñanza de la Física*, 2009.
4. **Rodriguez, Aguilar, Jorge Luis**. *Diseño Diseñar Diseñando*, 2011.
5. Manual de Normas Gráficas. [En línea] http://iux.prod.uci.cu/.
6. **Jacobson, Ivar, Booch, Grady and Rumbaugh, James**. *El Proceso Unificado de Desarrollo de Software*. Madrid 2000.
7. **Garlan, David and Shaw, Mary**. *An introduction to software architecture*. Technical report, Carnegie Mellon University. School of Computer Science, 1994.
8. **Slideshare**. [En línea] http://www.slideshare.net/guacaneme/cris006
9. **Deitel, Harvey M. and Deitel, Paul J**. *Cómo programar en C/C++ y Java*. Pearson Educación, 2004.
10. **CMake**.Sitio Oficial. [En línea] http://www.cmake.org
11. Manual de Asisstant QtCreator, 2012.
12. **Maven**. Sitio Oficial. [En línea] http://maven.apache.org/index.html
13. **Abrirllave**. [En línea] http://www.abrirllave.com/xml
14. **Ri5**. [En línea] http://www.ri5.com.ar
15. **W3**. [En línea] http://www.w3.org/DOM/
16. **Coverpages**. [En línea] http://xml.coverpages.org/dom.html
17. **Microsoft**. [En línea] https://msdn.microsoft.com

18. **Saxproject**. [En línea] http://www.saxproject.org/

19. **Megginson**. [En línea] http://www.megginson.com/downloads/SAX/

20. **Developerlife**. [En línea] http://developerlife.com/saxvsdom/default.htm

21. **Unicode**. [En línea] http://www.unicode.org/reports

22. **Piattini, M.**. *Análisis y diseño detallado de aplicaciones informáticas de gestión*. Madrid,1996.

23. **Canós, José H. and Letelier , Patricio and Penadé, María del Carmen.** *Métodologías Ágiles en el Desarrollo de Software.*. Universidad Politécnica de Valencia, 2003.

24. **Letelier, Patricio.** *Extreme Programming (XP)*, 2004.

25. Metodologías De Desarrollo De Software. [En línea] http://www.informatizate.net

26. **Visual Paradigm International.** Sitio Oficial de Visual Paradigm. [En línea] 12 de diciembre de 2011. https://www.visual-paradigm.com/product/vpuml/.

27. **García de Jalón, Javier and Rodríguez, José Ignacio and Sarriegui, Alfonso and Brazález, José María.** *Aprenda C++ como si estuviera en primero*. Escuela Superior de Ingenieros Industriales de San Sebastián, 1998.

28. **Nokia Corporation**. Sitio Oficial de Qt. [En línea] http://qt.nokia.com/.

29. **Larman, Craig**. *UML y Patrones. Una introducción al análisis y diseño orientado a objetos y al proceso unificado*, 2003.

30. **Jeffries, R. and Anderson, A. and Hendrickson, C**. *Extreme Programming Installed*, 2001

31. **Beck, Kent**. *Extreme Programming Explained*, 1999.

32. **Gamma, Erick and Helm, Richard and Jonhson, Ralph and Vlissides Jonh**. *Patrones de Diseño*, 2012.

33. **Grosso, Andrés**. *Prácticas de Software: Patrones Grasp*. Disponible en: http://www.practicasdesoftware.com.ar/20 grasp/.

34. **Pressman, Roger**. *Ingenieria del Software 6ta Edicion*, 2004.

35. **Gutiérrez, Javier J. and Escalona, M. J. and Mejías, M. and Torres, J**. *Pruebas del sistema en Programación Extrema*, 2013.

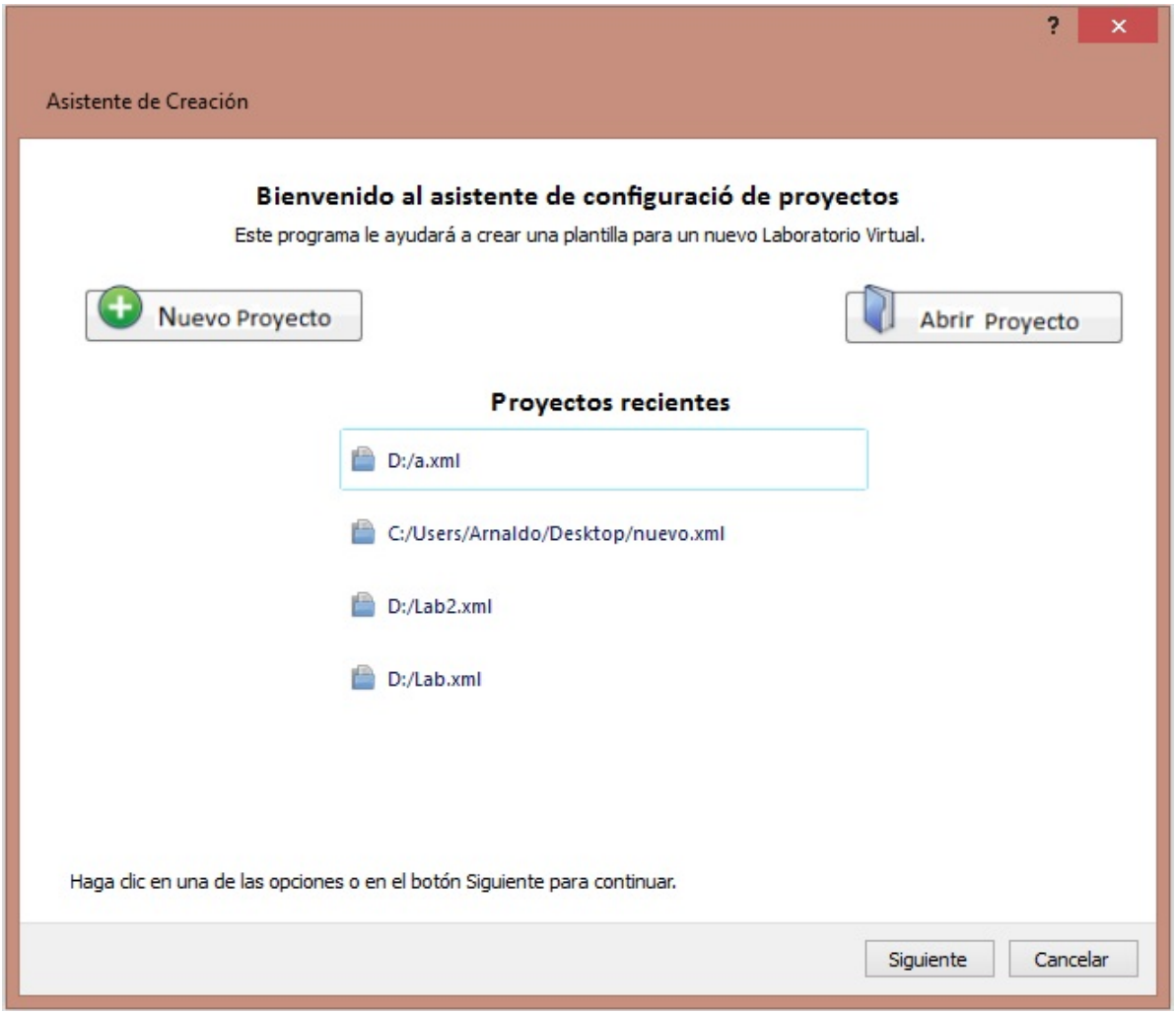
36. **Jurado, Carlos Blé**. *Diseño ágil con TDD*, 2010.

Apéndices

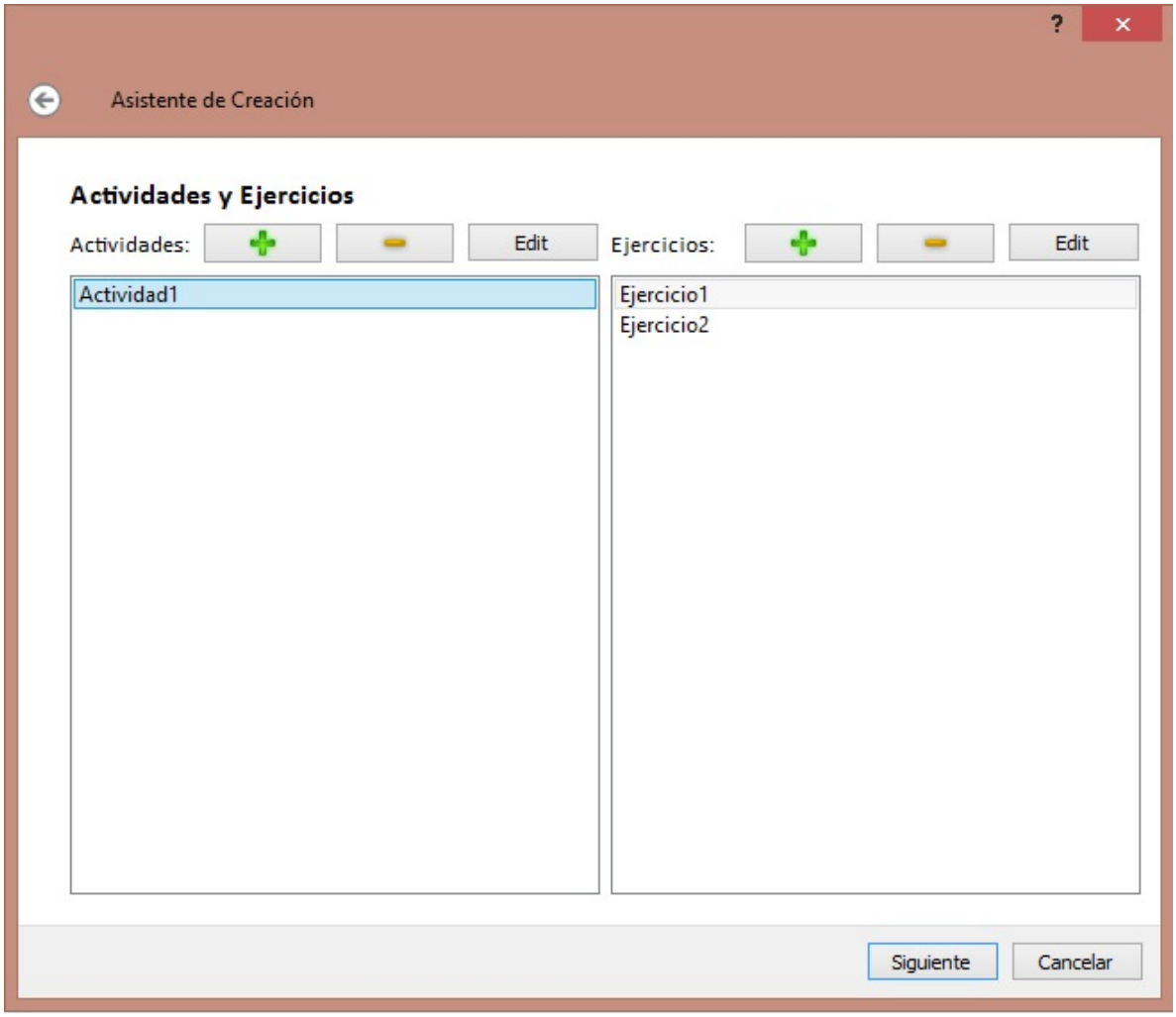
APÉNDICE A

Ejemplo de proyecto

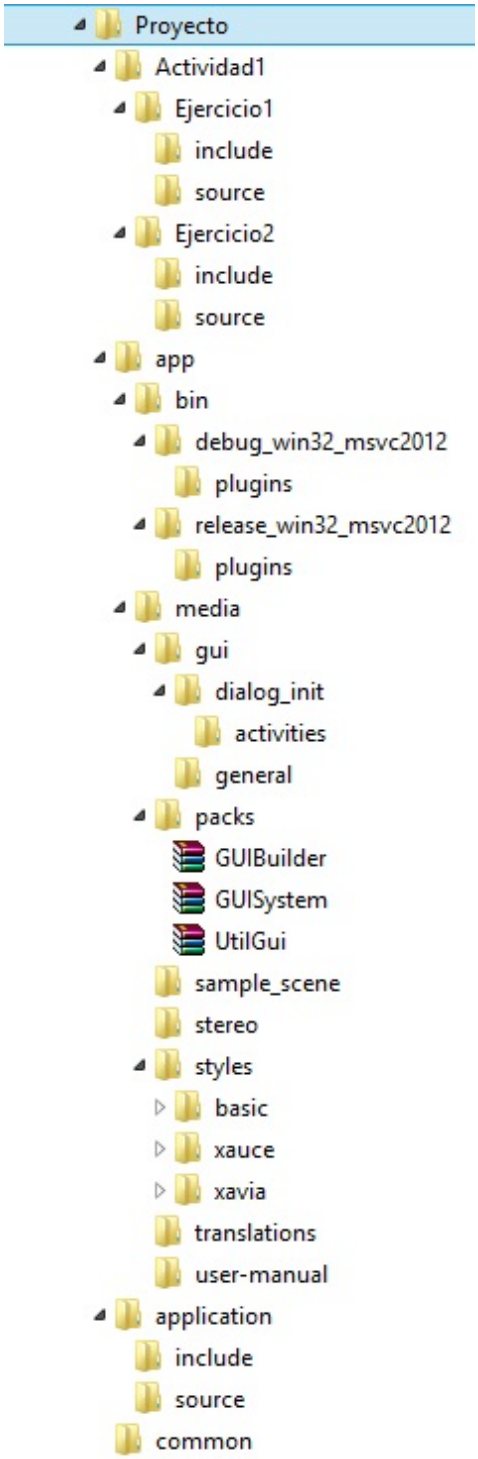
A.1. Interfaz de inicio con la lista de proyectos recientes.



A.2. Interfaz para gestionar las actividades y ejercicios del laboratorio.



A.3. Estructura de archivos y carpetas del proyecto



A.4. Interfaz que debe generarse al compilar el proyecto