Facultad 5 – Centro de Informática Industrial

Módulo para la Edición de Escenas de los Laboratorios Virtuales

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

AUTOR: **Omar Pino Rodríguez**

TUTOR: **Lic. Luis Gabriel Viciedo Caraballoso**

ASESOR: **Manuel Villanueva Betancourt**

La Habana, junio de 2011.

Año 53 de la Revolución.

DECLARACIÓN DE AUTORÍA

Declaro ser autor de la presente tesis y reconozco a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

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

_____	_____
Firma del Autor	Firma del Tutor
Omar Pino Rodríguez.	**Lic. Luis Gabriel Viciedo Caraballoso.**

DATOS DE CONTACTO

Nombre y apellidos: Luis Gabriel Viciedo Caraballoso.

Especialidad de graduación: Licenciatura en Educación, especialidad Física.

Categoría docente: Profesor auxiliar.

Categoría Científica: Licenciado.

Años de experiencia: 11 años.

Correo electrónico: viciedo@uci.cu.

AGRADECIMIENTOS

Hago llegar mis más sinceros agradecimientos a todas las personas que me ayudaron en la realización de este trabajo de diploma. Un agradecimiento especial a mi tutor Luis Gabriel Viciedo por su apoyo, al asesor Manuel Villanueva por atender en todo momento las dudas y propuestas surgidas. A mis amigos Alberto y Leiser por su gran ayuda. A todos los profesores que me han ayudado en general.

A mi madre por su cariño y amor todos estos años. A mi padre por su dedicación y apoyo. A mi novia Ladimairy por su amor y comprensión. A toda mi familia y amigos.

DEDICATORIA

A má y pá por ser los mejores padres del mundo.

A mis abuelos por confiar en mí.

Al amor de mi vida.

A mi gran familia.

RESUMEN

El gran avance de la informática en los últimos tiempos ha propiciado el surgimiento de sistemas de realidad virtual y por ende nuevas técnicas para su desarrollo. Lo que comenzó como parte de entrenamientos para centros de investigación, se ha convertido en una amplia gama de softwares, para el uso de profesionales, técnicos y estudiantes en todo el mundo. Ejemplo de ello son los programas de simulación para entrenamientos de tiro, en la industria militar, de vuelos aeronáuticos, y laboratorios virtuales utilizados para el aprendizaje.

La creación de este tipo de aplicaciones conlleva a un trabajo en su mayoría complejo y que requiere de mucho cuidado, ya que se pretende semejar la realidad. Las escenas dentro de este tipo de programas constituyen una parte fundamental, ya que simulan el entorno donde nos movemos y donde se interactúa con los demás objetos en el escenario. De este modo, existen estrategias para el manejo de esta información en el desarrollo de las aplicaciones. Es por ello que el presente trabajo aborda la implementación de un módulo para la edición de los archivos que controlan las características e información de las escenas utilizadas en la realización de laboratorios virtuales. Este módulo permitirá al programador manejar estos archivos de una manera fácil, restándole menos tiempo de implementación, con una correcta estructura y sintaxis, entre otras características de edición.

Para satisfacer el objetivo propuesto se realizó un estudio básico de los archivos scene y otras estructuras de documentos XML, conceptos asociados al dominio del problema. Se analizaron algunas técnicas que son usadas actualmente en la edición de documentos y algunos editores existentes en el mundo. Finalmente se describen los procesos que ocurren en el sistema y se implementan las interacciones entre éstos.

Palabras clave

Entornos virtuales, editores XML, environment, laboratorios virtuales, node, scene, XML.

Índice

Índice de tablas

Índice de figuras

INTRODUCCIÓN

En los últimos años la Realidad Virtual ha sido una de las ramas que más ha revolucionado, no solo a la informática, sino a otras esferas como la educación, la medicina, la industria militar, entre otras. La Realidad Virtual básicamente consiste en la simulación de las diferentes percepciones gráficas, auditivas y táctiles en un universo tridimensional ficticio creado por el ordenador. En la actualidad la realidad virtual se plasma en una gran cantidad de sistemas que permiten a los usuarios experimentar, de manera simulada, diferentes aspectos de la vida.

Dentro de esta gama de sistemas se encuentran los laboratorios virtuales o automatizados que apoyan la investigación y el aprendizaje de profesionales y técnicos en todo el mundo. Los laboratorios virtuales son sistemas computacionales que simulan el ambiente de un laboratorio real o tradicional y aportan las mismas funcionalidades que estos, con la única diferencia que el individuo y el laboratorio no coinciden en el espacio.

Las aplicaciones de estos laboratorios en la educación y la investigación son notables en estos tiempos, en Cuba también se ha logrado un desarrollo de los laboratorios virtuales y la Realidad Virtual. Empresas dedicadas a la simulación y algunas universidades se desarrollan en esta línea de trabajo. Específicamente, por su perfil de producción, en la Facultad 5 de la Universidad de las Ciencias Informáticas existen proyectos que laboran en esta índole, uno de ellos es el Proyecto de Laboratorios Virtuales (PROLAVI).

Para el desarrollo de estos laboratorios virtuales dentro del proyecto se han buscado muchas alternativas de acuerdo con los recursos y el factor tiempo disponible para la entrega de estos softwares siempre buscando una mejor calidad con una mayor facilidad de implementación. De esta necesidad de acudir a estrategias de desarrollo que faciliten el trabajo de los desarrolladores surge la idea de implementar un modulo para la edición de las escenas de los laboratorios virtuales atendiendo a la **situación problémica** siguiente que presenta en la actualidad el proyecto PROLAVI.

Las características de las escenas de los laboratorios virtuales son guardadas en un archivo que contiene los aspectos más significativos, como la posición de los elementos dentro de la escena, entre otras. En la actualidad estos archivos son creados completamente desde cero, mediante la utilización de una biblioteca de análisis de documentos que utiliza el lenguaje XML.

El hecho de crear estos archivos en su totalidad se hace un trabajo engorroso para los desarrolladores del proyecto, restando mucho tiempo de desarrollo y estando a expensas de cometer errores, puesto que no se tiene un control total del formato y la estructura que debe seguir este tipo de archivos, entre otros errores de edición.

Luego, se formula el siguiente **Problema de Investigación:** ¿Cómo crear y modificar los archivos que manejan las características de las escenas, para disminuir el tiempo de implementación y contribuir a la calidad de la edición?

Lo que determina el siguiente **Objeto de estudio:** Proceso de edición de documentos.

Para darle solución al problema planteado se propone como **Objetivo General:** Implementar un módulo para la edición de los archivos que controlan las características de las escenas de los laboratorios virtuales.

Lo que precisa como **Campo de acción:** Proceso de edición de archivos scene.

Como **Idea a defender,** se propone: La implementación de un módulo para la edición de los archivos scene, permitirá un mejor manejo de las escenas de los laboratorios virtuales y una reducción del coste en tiempo de implementación.

Para darle cumplimiento al objetivo se proponen las siguientes**: Tareas de investigación:**

- Elaboración del marco teórico a través del estudio del estado del arte de la extensión de archivos scene y de los diferentes editores de documentos existentes con sus características.
- Selección de la metodología de desarrollo de software.
- Selección de herramientas y tecnologías para el desarrollo de la aplicación.
- Análisis y diseño del mecanismo a utilizar para la edición de archivos scene en los laboratorios virtuales.
- Implementación del modulo de edición de archivos scene con la metodología, lenguajes y herramientas seleccionadas.
- Valoración de los resultados obtenidos.

Métodos Científicos de Investigación:

Teóricos:

Analítico-Sintético: Para la consulta de la documentación existente acerca de la extensión de archivos scene, del lenguaje XML, de los diversos editores de documentos existentes, y el estudio de los conceptos empleados en el tema en cuestión.

Análisis Histórico-Lógico: Para tener información sobre las tendencias actuales de los editores de documentos existentes.

Inductivo – Deductivo: Se hace uso de este método porque se centrará la investigación en el uso del archivo scene, buscando la factibilidad que el mismo ofrece en su uso en los laboratorios virtuales.

Empíricos:

Consulta de fuentes de información: Método empírico en el cual se recogen las consultas hechas a las distintas fuentes de información necesarias en el desarrollo del trabajo, desde las diversas bibliografías hasta el asesor de metodología, el tutor, entre otros.

Resultados esperados:

Se espera que el siguiente trabajo sea utilizado por los desarrolladores del proyecto PROLAVI y contribuya a aumentar la calidad y las funcionalidades de los laboratorios virtuales que se desarrollan con una mayor facilidad y en menor coste de tiempo.

Estructura de cada Capítulo:

- **Capítulo 1**. Fundamentación teórica. Recopilación de la teoría asociada al dominio del problema de los laboratorios virtuales, del lenguaje XML, de la extensión de archivos scene, de los diferentes editores de documentos existentes y la descripción de las metodologías y herramientas que pudieran ser empleadas.
- **Capítulo 2.** Descripción de la solución propuesta. Selección de la metodología de desarrollo, lenguajes y herramientas a utilizar en el desarrollo del trabajo. Definición de los requisitos funcionales y no funcionales; modelo de domino; actores y casos de uso del sistema. Así como la identificación y descripción de los patrones de diseño a utilizar.

- **Capítulo 3**. Diseño arquitectónico del sistema. Descripción del estándar de codificación utilizado. Realización del diagrama de clases del sistema con sus respectivas descripciones, diagramas de colaboración y de componentes de la vista de código fuente y binaria.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

Introducción

En este capítulo se realiza la fundamentación teórica del propósito general de este trabajo dándole cumplimiento al problema científico planteado. Se abordan un conjunto de referencias, citas bibliográficas y valoraciones que enriquecen el contenido del mismo. En un primer epígrafe se conforma el estado del arte de los entornos virtuales de aprendizaje con sus diferentes ejemplos de actualidad, para aterrizar en los laboratorios virtuales como modalidad de estos escenarios de aprendizaje. Más adelante se abordan las características de la extensión de archivos (scene) y del lenguaje XML en general. Luego se hace una caracterización de varios editores XML existentes en el mundo en la actualidad y se culmina el capítulo con una caracterización de las diferentes herramientas y metodologías que se pudieran optar para la realización de este trabajo.

1.1- Entornos virtuales de aprendizaje

Los sistemas de aprendizaje actuales en su mayoría se nutren de los nuevos sistemas mediados por las tecnologías de la información y las comunicaciones que constantemente añaden nuevas posibilidades de innovación en las formas de aprendizaje. De esta nueva forma de organización de la enseñanza mediante la combinación de herramientas telemáticas y multimedias surge lo que hoy conocemos como Entornos Virtuales de Aprendizaje.

Se entiende por entorno virtual de aprendizaje el espacio donde interactúan las nuevas tecnologías como son: la televisión, el internet, los sistemas satelitales, las multimedias, entre otras, favoreciendo el aprendizaje, la apropiación de conocimientos y los procesos pedagógicos. Estos están conformados por el espacio, el estudiante, el profesor asesor, los contenidos educativos, la evaluación y los medios de información y comunicaciones.

Los ambientes de aprendizaje no se circunscriben a la educación formal, ni tampoco a una modalidad educativa particular, se trata de aquellos espacios en donde se crean las condiciones para que el individuo se apropie de nuevos conocimientos, de nuevas experiencias, de nuevos elementos que le generen procesos de análisis, reflexión y apropiación. Llamémosle virtuales en el sentido que no se llevan a cabo en un lugar predeterminado y que el elemento distancia está presente. (Avila, 1999)

La UNESCO (1998) en su informe mundial de la educación, señala que los entornos de aprendizaje virtuales constituyen una forma totalmente nueva de tecnología educativa y ofrece una compleja serie de oportunidades y tareas a las instituciones de enseñanza de todo el mundo, el entorno de aprendizaje virtual lo define como un programa informático interactivo de carácter pedagógico que posee una capacidad de comunicación integrada, es decir, que está asociado a nuevas tecnologías. (Calderon, 2006)

1.1.1- Tipos de entornos virtuales de aprendizaje

Actualmente, la aplicación del concepto nuevas tecnologías en la enseñanza al ámbito de la realización de prácticas ha dado lugar a la aparición de diferentes modalidades de entornos de experimentación. A continuación diferentes modalidades de los entornos de aprendizaje:

Aula Virtual

Aula virtual dentro del entorno de aprendizaje, consta de una plataforma o software a través del cual el ordenador permite la facilidad de dictar las actividades en clases, de igual forma permitiendo el desarrollo de las actividades de enseñanza y aprendizaje habituales que requerimos para obtener una buena educación. A través de ese entorno el alumno puede acceder y desarrollar una serie de acciones que son las propias de un proceso de enseñanza presencial tales como conversar, leer documentos, realizar ejercicios, formular preguntas al docente, trabajar en equipo, etc. Todo ello de forma simulada sin que medie una interacción física entre docentes. Por ende en el mercado existen actualmente numerosas aplicaciones que permiten la capacidad de crear cursos a distancia simulando aulas virtuales como, por ejemplo, WebCT, Catedr@, eCollege, Moodle, Dokeos, Claroline, Manhattan Virtual Classroom, Learning Space, e-ducativa, etc. (Marchena, 2008)

Play Learning

Es un sistema de formación, desarrollado en el año 2008 por IFO (Instituto de Formación Online), que combina la comodidad y eficacia del e-learning con el carácter lúdico de los videojuegos, con el objetivo de hacer al usuario más ameno y eficaz el proceso de aprendizaje. Con esta metodología, el usuario avanza en sus lecciones bajo la apariencia de un videojuego modular en el que las diferentes lecciones se traducen en las fases del juego. En cada competencia, el usuario tiene que superar diferentes retos que le permiten ganar puntos en cada una de las habilidades y superar, los retos marcados por el coach virtual o tutor virtual.

Metodológicamente el sistema permite la simulación de entornos casi imposibles de otra forma, tanto a nivel de localizaciones (por ejemplo una central nuclear) como de situaciones (por ejemplo, conductas conflictivas y su resolución mediante habilidades).

Virtual Classroom

Es una herramienta desarrollada por CAE (Computer Aided Elearning) que combina un sistema de videoconferencia con herramientas que permiten la perfecta comunicación entre tutor y alumno para el aprendizaje a distancia.

1.1.2- Los laboratorios virtuales

Los entornos virtuales de aprendizaje se han venido intensificando en cuanto al crecimiento de las tecnologías y las nuevas tendencias que van adoptando las mismas, como ya vimos anteriormente en los diferentes ejemplos de entornos virtuales de aprendizaje. Uno de estos sistemas de apoyo al aprendizaje y uno de los más interesantes son los *e-laboratories* o laboratorios virtuales.

Se conoce como laboratorio virtual a un sistema de cómputo que aproxima la realidad de un laboratorio normal o tradicional como suele llamársele. En los laboratorios virtuales se llevan a cabo los mismos procedimientos que en los laboratorios tradicionales, en dependencia de los objetivos que se persigan, visualizándose los instrumentos, fenómenos y actividades mediante objetos simulados, imágenes interactivas, videos, etc. (Herreros, 2009)

Ventajas e inconvenientes de los laboratorios virtuales

Bajo estas políticas es posible modelar entornos que no están disponibles para los estudiantes pues constituyen riesgos o un alto valor económico y a su vez no puede sustituir la enseñanza que brinda una experiencia práctica en un laboratorio tradicional, y estas constituyen, entre otras, algunas de las ventajas e inconvenientes de los laboratorios virtuales que se destacan a continuación:

- Los estudiantes aprenden probando, sin miedo a provocar algún tipo de accidente y permitiéndoles realizar varias veces la misma práctica.
- Reducen el coste de construcción y mantenimiento que conlleva un laboratorio tradicional y el costo de los recursos utilizados en las diferentes prácticas.
- El hecho de que el laboratorio no sea físico permite a un mayor número de estudiantes realizar prácticas en ellos.
- Permite un conjunto de funcionalidades que se pueden alterar por el propio estudiante y de esta manera permite en ocasiones una vista más detallada del problema.

Pero estas ventajas vienen acompañadas por algunas inconvenientes:

- Si no existe algún tipo de guía para las actividades a realizar puede darse el caso que el estudiante solo se comporte como un espectador y pierda la experiencia práctica que conlleva realizar los diferentes ejercicios.
- Los laboratorios reales brindan una experiencia práctica mucho más enriquecedora, muchas veces emparejadas por la presencia de un profesor tutor que se ve un tanto frenada en los laboratorios virtuales.
- También se dice que el estudiante puede perder un tanto la vista de la realidad al estar interactuando constantemente con objetos no reales.

1.2- Extensible Markup Language (XML)

XML es un lenguaje de marcas que ofrece un formato para la descripción de datos estructurados, el cual conserva todas las propiedades importantes SGML. XML es un metalenguaje, dado que con él podemos definir nuestro propio lenguaje de presentación y se centra en la información en sí misma. La particularidad más importante del XML es que no posee etiquetas prefijadas con anterioridad, ya que es el propio diseñador el que las crea a su antojo, dependiendo del contenido del documento. XML fue desarrollado por un grupo de trabajo bajo los auspicios del consorcio World Wide Web (W3C) a partir de 1996.

1.2.1- Características de XML

XML es un formato basado en texto, específicamente diseñado para almacenar y transmitir datos. Un documento XML se compone de elementos XML, cada uno de los cuales consta de una etiqueta de inicio, de una etiqueta de fin y de los datos comprendidos entre ambas etiquetas. Un documento XML contiene texto anotado por etiquetas. XML admite un conjunto ilimitado de etiquetas, no para indicar el aspecto que debe tener algo, sino lo que significa.

Por ejemplo: un elemento XML puede estar etiquetado como precio, número de pedido o nombre. El autor del documento es quien decide que tipo de datos va a utilizar y que etiquetas son las más adecuadas.

<reporte-clima>

<fecha>Mayo 13, 2011</fecha>

<hora>09:00</hora>

<area>

<departamento>IMNC</ departamento >

<ciudad>La Habana</ciudad>

<pais>Cuba</pais>

</area>

<medidas>

<cielo>parcialmente nublado </cielo>

<temperatura>26</temperatura>

<viento>

<direccion>NO</direccion>

<velocidad>16</velocidad>

</viento>

<h-indice>51</h-indice>

<humedad>87</humedad>

<visibilidad>10</visibilidad>

<uv-indice>1</uv-indice>

</medidas>

</reporte-clima>

XML se basa en una tecnología desarrollada a partir de estándares probados y optimizada para la Web. La iniciativa XML consta de un conjunto de estándares relacionados entre sí:

- XML (Extensible Markup Language). Es una recomendación, que significa que el estándar es estable y que los desarrolladores de Web y de herramientas pueden adoptarlo plenamente.
- Namespaces. En XML es una recomendación que describe la sintaxis y la compatibilidad de los espacios de nombres para los intérpretes de XML.
- DOM (Document Object Model). Es una recomendación que ofrece un estándar para el acceso mediante programación a los datos estructurados (a través de scripts), de modo que

los desarrolladores puedan interactuar de forma coherente con los datos basados en XML y computarlos.

1.3- Archivos de formato SCENE

El formato de archivo (scene) es uno de los formatos más usados en la actualidad por las funcionalidades que brinda. Posee una sintaxis basada en XML, representa la forma más sencilla de rellenar una escena 3D y facilidad en el manejo de objetos 3D y de la escena y sus propiedades en general, muy utilizada en el desarrollo de aplicaciones en OGRE. Un archivo scene muy simple puede lucir así:

```
<scene>
    <environment>
        <colourAmbient r="0.5" g="0.5" b="0.5" />
    </environment>
     <nodes>
        <node>
            <entity meshFile="sample.mesh" />
        </node>
    </nodes>
</scene>
```

Esto establece el ambiente mundo de la iluminación a RGB (0.5, 0.5, 0.5), que es un gris oscuro. También carga un modelo de ejemplo *sample.mesh* en el coeficiente por defecto de las coordenadas de (0, 0, 0).

Tabla 1 Estructura general de un archivo scene

1 Common Structures	3 externals
1.1 Colour	4 nodes
1.2 Vector	4.1 node
1.3 Quaternion	4.1.1 position

2 environment section	4.1.2 scale
2.1 colourAmbient	4.1.3 orientation/rotation
2.2 colourBackground	4.2 entity
2.3 fog	4.3 light
2.4 shadows	4.3.1 lightAttenuation
2.5 skyBox	4.3.2 lightRange
2.6 skyDome	4.4 camera
2.7 skyPlane	4.4.1 clipping
2.8 title	4.4.2 renderTarget
2.9 gravity	4.5 billboardSet
2.10 extents	4.5.1 billboard
2.11 physicsGroup	4.6 particleSystem
2.11.1 collide	4.7 shape
2.11.2 dontCollide	4.7.1 size
2.12 renderTexture	4.7.2 radius
2.13 perRenderGroup	4.7.3 length
	4.7.4 distance
	4.7.5 normal

Un archivo scene presenta una estructura amplia que puede llegar a tener cualquiera de las propiedades que se enumeran a continuación, siempre cumpliendo con la estructura básica que debe poseer (como se ve en el ejemplo anterior):

A continuación una descripción de algunas de las propiedades que se pueden ver en la tabla, para una mejor comprensión:

1 Common Structures

1.1 Colour

Una etiqueta con los atributos "r", "g", "b" y opcionalmente "a". A estos se le asignan valores de punto flotante entre 0.0 y 1.0.

1.2 Vector

Una etiqueta con "x", "y", "z" atributos. Estos son números de punto flotante, sin límite de alcance.

1.3 Quaternion

Una etiqueta con:

"w", "x", "y", "z"

"qw", "qx", "qy", "qz"

"axisx", "axisy", "axisz"

"anglex", "anglada", "anglez", "ángulo"

Estas etiquetas representan una orientación. Representando los valores *Quaternion* o los ángulos de *Euler*. Estos ángulos se especifican en radianes.

2 environment section

2.1 colourAmbient

Establece la iluminación ambiental de la escena.

2.2 colourBackground

Establece el color de la limpieza de la ventana.

2.3 fog

Establece el modo de utilizar la niebla.

4.1 node

Son los objetos que se encuentran en la escena. Tienen un nombre único o id que los diferencia. Pueden contener otros nodos.

4.1.1 position

Posición en relación con el nodo principal. Tipo: *Vector.*

4.1.2 scale

Escala, en relación con el nodo principal. Tipo: *Vector*.

4.1.3 orientation/rotation

Orientación, en relación con el nodo principal. Tipo: *Quaternion*.

1.4- Editor XML

Un editor XML es un editor de lenguaje de marcas con muchas funcionalidades que facilitan la edición de un archivo XML. Esto puede hacerse utilizando un editor común de texto plano, con todo el código visible, pero los editores de XML han añadido varias funcionalidades que facilitan mucho este trabajo, como los menús y los botones para las tareas que son comunes en la edición de XML, con base en los datos suministrados con la definición de tipo de documento o el árbol XML.

También hay editores gráficos XML que ocultan el código fuente y presentan el contenido al usuario en un formato más agradable y fácil de usar. Esto es útil para situaciones donde las personas que no dominan el código XML presentan la necesidad de introducir información en documentos basados en XML tales como hojas de tiempo, informes de gastos, etc. E incluso si el usuario está familiarizado con XML, el uso de estos editores, que se ocupan de los detalles de la sintaxis, es a menudo más rápido y más conveniente.

Existen 3 tipos de editores XML:

1.4.1- Editores textuales

Los editores de texto XML o editores textuales generalmente proporcionan características que tratan de trabajar con las etiquetas de los elementos presentes en el documento. El resaltado de sintaxis es una norma básica de cualquier editor de XML, es decir, que el texto posee elementos de colores diferentes del texto normal.

La ventaja de los editores de texto es que se representan exactamente la información que se almacena en el archivo XML. Es la mejor manera de controlar el formato del archivo para realizar operaciones de bajo nivel (como una búsqueda y sustitución en los nombres de elementos) y para editar archivos XML sin ningún tipo de esquema o el archivo de configuración.

1.4.2- Editores gráficos

Los editores gráficos están basados en interfaces gráficas de usuario y su aplicación puede ser más fácil para algunas personas que el uso de editores de texto. Su mayor funcionalidad radica en no exigir un conocimiento de la sintaxis XML.

Este tipo de editores son generalmente más útiles para los lenguajes XML de datos. Los editores de documentos o de texto tienden a ser bastante libres en su estructura, y esto tiende a desafiar la naturaleza rígida de muchas estructuras de archivos XML existentes.

1.4.3- Editores WYSIWYG

Estos editores son llamados WYSIWYG como parte de una abreviatura de "lo que ves es lo que obtienes" o sea cuando tratan de mostrar la última prestación que va a tener el documento y trabajar en esa base.

En los editores WYSIWYG la gente puede editar archivos directamente con las marcas representadas por algún tipo de visualización gráfica en lugar de al descubierto el código XML. A menudo, los editores WYSIWYG intentan emular el resultado final de alguna transformación o la aplicación de estilos CSS. Esta emulación puede o no ser posible, en función de la transformación del XML en el resultado final.

El uso ingenuo de un editor WYSIWYG puede conducir a la creación de documentos que no tienen la semántica intrínseca del lenguaje XML. Esto se produce si el usuario se centra en tratar de lograr una presentación visual con el editor, en lugar de usar el WYSIWYG para hacer más fácil la edición del documento.

1.4.4- Ejemplos de editores XML

XML Notepad:

Es un editor de texto diseñado por la compañía Microsoft en el 2007, XML Notepad presenta una muy fácil interfaz para la creación y manejo de documentos XML. Es compatible solo con el sistema operativo Windows en versiones superiores a Windows XP.

Dentro de sus características se incluyen:

- Vista de árbol sincronizado con el nodo de texto Vista para la edición rápida de nombres de nodos y los valores.
- Búsqueda incremental (Ctrl + I) en ambos puntos de vista de árbol y texto, con el fin de escribir que se desplaza a nodos coincidentes.
- Cortar / copiar / pegar con el apoyo del espacio de nombres completo.
- Arrastrar y soltar para la manipulación fácil del árbol, incluso a través de diferentes instancias de XML Notepad y del sistema de archivos.
- Infinito deshacer / rehacer para todas las operaciones de edición.
- Fuentes configurables y colores a través del diálogo de opciones.
- Instantánea validación de esquemas XML, mientras que se edita con errores y advertencias se muestran en la ventana de lista de tareas.
- Soporte para los editores de tipos de datos personalizados para la fecha, dateTime y el tiempo y otros tipos como el color.
- HTML visor para el procesamiento de instrucciones de procesamiento de hoja de estilo XML.

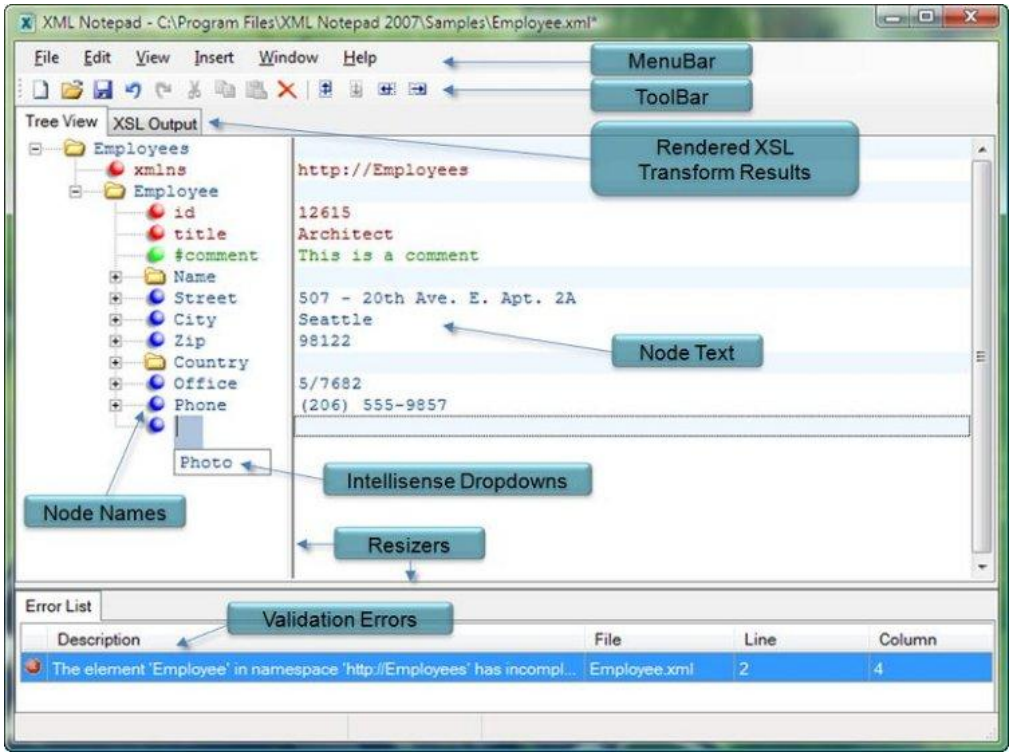


Figura 1 Editor XML Notepad (CodePlex, 2006 - 2011)

EditiX

EditiX es un potente editor de XML, presenta un editor visual de esquemas, XQuery Editor y depurador de XSLT para Windows, Linux y Mac OS X diseñado para ayudar a los autores y los programadores de aplicaciones web y tomar ventaja de las últimas tecnologías relacionadas con XML. EditiX proporciona a los usuarios una amplia gama de funcionalidades de XML dentro de un IDE refinado que lo guía en su manejo. EditiX presenta ubicación de XPath en tiempo real y detección de errores de sintaxis.

EditiX soporta múltiples plantillas y gestión de proyectos. El usuario puede aplicar XSLT, XQuery o cualquier transformación y se muestra el resultado del fin específico. EditiX incluye plantillas predeterminadas con XML, DTD, XHTML, XSLT, XSD, RelaxNG XML, SVG, MathML y FOR XML.

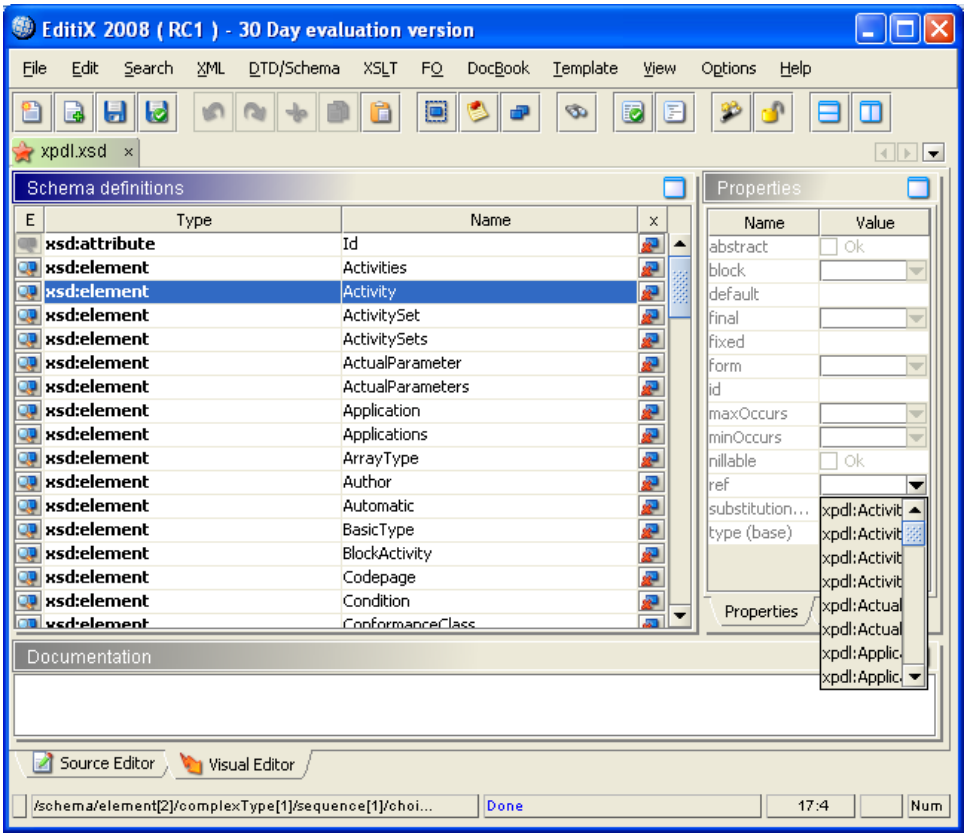


Figura 2 Editor EditiX XML (JapiSoft, 2004 - 2011)

<Oxygen/> XML:

<oXygen/> es uno de los mejores editores de XML disponibles en la actualidad, con un gran número de usuarios, desde principiantes hasta expertos en XML. Es la única herramienta XML que soporta todos los lenguajes de esquema XML hasta el momento. La implementación de XSLT y XQuery se ha mejorado con depuradores de gran alcance y perfiles de rendimiento. Se puede utilizar <oXygen/> Editor XML para trabajar con todas las tecnologías basadas en XML como bases de datos XML, tuberías XProc, y servicios web. <oXygen/> XML Editor es una aplicación multiplataforma disponible en los principales sistemas operativos (Windows, Mac OS X, Linux, Solaris) y se puede utilizar ya sea independiente o como un plugin de Eclipse. Hasta el momento el software presenta licencia de software propietario.

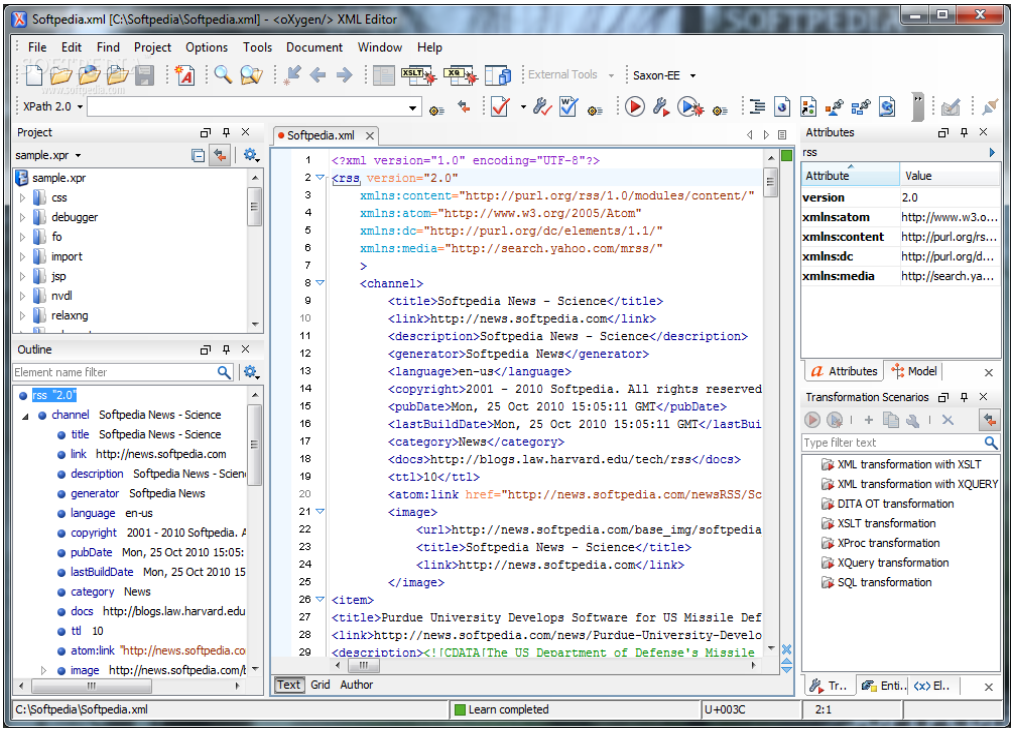


Figura 3 Editor Oxygen XML (Oxygen, 2002 - 2011)

1.5- Metodologías de desarrollo de software

Las Metodologías de Desarrollo de Software surgen ante la necesidad de utilizar una serie de procedimientos, técnicas, herramientas y soporte documental a la hora de desarrollar un producto software. Dichas metodologías pretenden guiar a los desarrolladores al crear un nuevo software, pero los requisitos de un software a otro son tan variados y cambiantes, que ha dado lugar a que exista una gran variedad de metodologías para la creación del software. Se podrían clasificar en dos grandes grupos:

- *Las metodologías orientadas al control de los procesos*, estableciendo rigurosamente las actividades a desarrollar, herramientas a utilizar y notaciones que se usarán. Estas metodologías son llamadas Metodologías Pesadas.
- *Las metodologías orientadas a la interactuación con el cliente y el desarrollo incremental del software*, mostrando versiones parcialmente funcionales del software al cliente en intervalos cortos de tiempo, para que pueda evaluar y sugerir cambios en el producto según se va desarrollando. Estas son llamadas Metodologías ligeras/ágiles. (José Canós, 2006)

1.5.1- Diferencias entre las metodologías

Metodologías Ágiles	Metodologías Tradicionales
Basadas en heurísticas provenientes de prácticas de producción de código	Basadas en normas provenientes de estándares seguidos por el entorno de desarrollo
Especialmente preparados para cambios durante el proyecto	Cierta resistencia a los cambios
Impuestas internamente (por el equipo)	Impuestas externamente
Proceso menos controlado, con pocos principios	Proceso mucho más controlado, con numerosas políticas/normas
No existe contrato tradicional o al menos es bastante flexible	Existe un contrato prefijado
El cliente es parte del equipo de desarrollo	El cliente interactúa con el equipo de desarrollo mediante reuniones
Grupos pequeños (<10 integrantes) y trabajando en el mismo sitio	Grupos grandes y posiblemente distribuidos
Pocos artefactos	Más artefactos
Pocos roles	Más roles
Menos énfasis en la arquitectura del software	La arquitectura del software es esencial y se expresa mediante modelos

Figura 4 Comparativa entre tipos de metodologías (Molpeceres, 2002)

1.5.2- RUP

El Proceso Racional Unificado (Rational Unified Process en inglés) es un proceso de desarrollo de software y junto con el Lenguaje Unificado de Modelado UML, constituye la metodología estándar más utilizada para el análisis, implementación y documentación de sistemas orientados a objetos. La metodología RUP no es un sistema con pasos firmemente establecidos, sino un conjunto de metodologías adaptables al contexto y necesidades de cada organización. RUP es uno de los procesos más generales de los existentes actualmente, ya que en realidad está pensado para adaptarse a cualquier proyecto, y no tan solo de software. (Molpeceres, 2002)

Se caracteriza por ser iterativo e incremental, estar centrado en la arquitectura y guiado por los casos de uso. Incluye artefactos (que son los productos tangibles del proceso como por ejemplo, el modelo de casos de uso, el código fuente, etc.) y roles (papel que desempeña una persona en un determinado momento, una persona puede desempeñar distintos roles a lo largo del proceso).

Un proyecto realizado siguiendo RUP se divide en cuatro fases:

1- Inicio (puesta en marcha).
2- Elaboración (definición, análisis, diseño).
3- Construcción (implementación).
4- Transición (fin del proyecto y puesta en producción).

En cada fase se ejecutarán una o varias iteraciones y dentro de cada una de ellas seguirá un modelo de cascada para los flujos de trabajo que requieren las nuevas actividades.

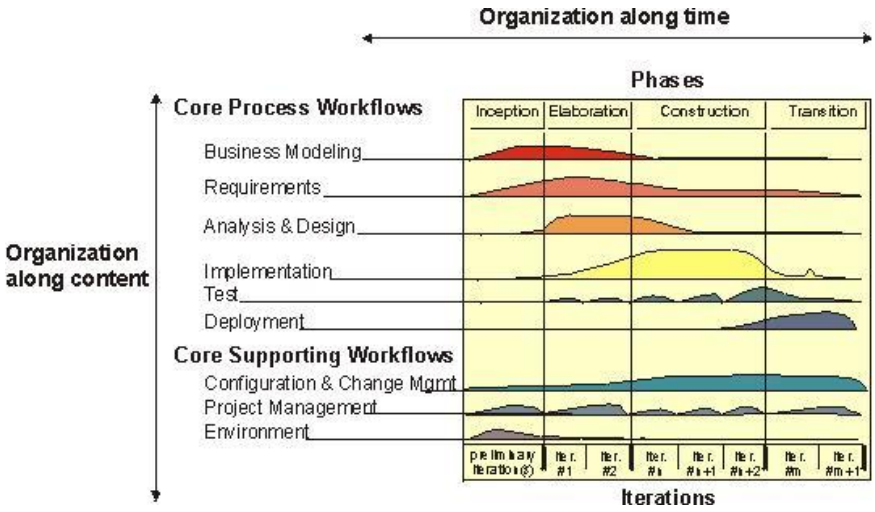


Figura 5 Vista general de RUP (Molpeceres, 2002)

Para una mejor comprensión de la imagen se definen a continuación las nueve actividades de RUP:

1- Modelado del negocio.
2- Análisis de requisitos.
3- Análisis y diseño.
4- Implementación.
5- Prueba.
6- Distribución.
7- Gestión de configuración y cambios.
8- Gestión del proyecto.
9- Gestión del entorno.

1.5.3- Programación extrema XP

La programación extrema o eXtreme Programming (XP) es un enfoque de la ingeniería de software formulado por Kent Beck, autor del primer libro sobre la materia, Extreme Programming Explained: Embrace Change (1999). Es el más destacado de los procesos ágiles de desarrollo de software. Al igual que éstos, la programación extrema se diferencia de las metodologías tradicionales principalmente en que pone más énfasis en la adaptabilidad que en la previsibilidad.

XP define 4 variables para el proyecto de software:

1- Coste.
2- Tiempo.
3- Calidad.
4- Alcance.

XP define Userstories como base del software a desarrollar. Estas historias las escribe el cliente y describen escenarios sobre el funcionamiento del software. A partir de las Userstories y de la arquitectura perseguida se crea un plan de entrega. (Escribano, 2002)

XP deriva una docena de Principios Básicos:
Realimentación rápida, Asumir la Simplicidad, El Cambio Incremental, Adherirse (Abrazar) al Cambio, Trabajo de Alta Calidad.

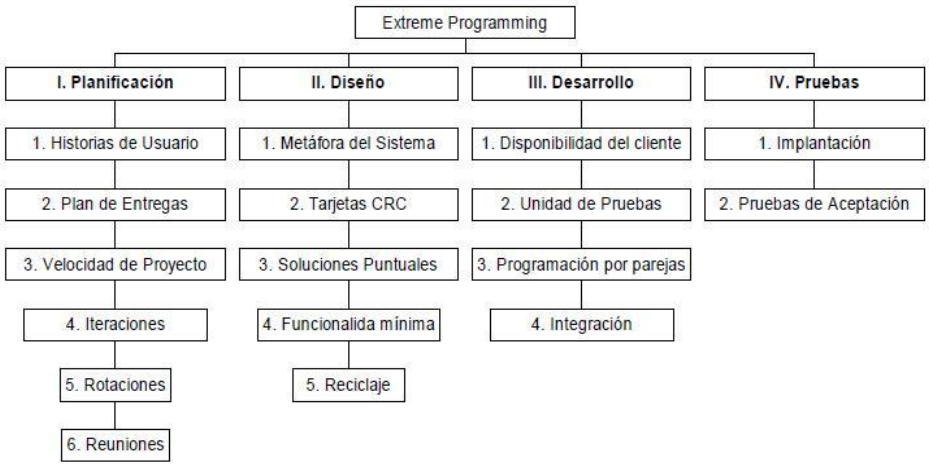


Figura 6 Ciclo de vida de XP (Escribano, 2002)

1.5.4- FDD

FDD es un proceso diseñado por Peter Coad, Erich Lefebvre y Jeff De Luca y se podría considerar a medio camino entre RUP y XP. Está pensado para proyectos con tiempo de desarrollo

relativamente cortos (menos de un año). No hace énfasis en la obtención de los requerimientos sino en cómo se realizan las fases de diseño y construcción. Se preocupa por la calidad, por lo que incluye un monitoreo constante del proyecto.

Ayuda a contrarrestar situaciones como el exceso en el presupuesto, fallas en el programa o el hecho de entregar menos de lo deseado. Propone tener etapas de cierre cada dos semanas. Se obtienen resultados periódicos y tangibles. (Mendoza, 2004)

Se basa en un proceso iterativo con iteraciones cortas que producen un software funcional que el cliente puede ver y monitorizar. El proceso consiste de cinco pasos secuenciales durante los cuales se diseña y se construye el sistema:

1- Desarrollo de un modelo global.
2- Construcción de una lista de funcionalidades.
3- Planeación por funcionalidad.
4- Diseño por funcionalidad.
5- Construcción por funcionalidad.

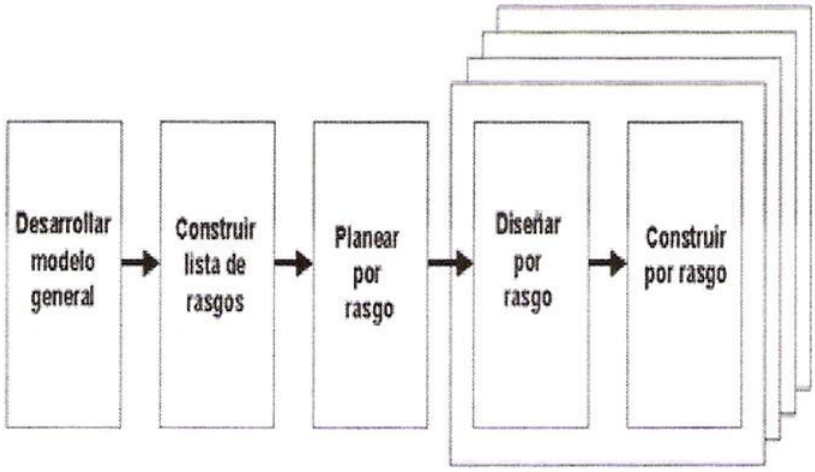


Figura 7 Ciclo de vida de FDD (Molpeceres, 2002)

1.6- Herramientas de ingeniería de software asistida por computadoras

Las herramientas CASE (Computer Aided Software Engineering, Ingeniería de Software Asistida por Computadora) son diversas aplicaciones informáticas destinadas a aumentar la productividad en el desarrollo de software reduciendo el coste de las mismas en términos de tiempo y de dinero. Estas

herramientas nos pueden ayudar en todos los aspectos del ciclo de vida de desarrollo del software en tareas como el proceso de realizar un diseño del proyecto, cálculo de costes, implementación de parte del código automáticamente con el diseño dado, compilación automática, documentación o detección de errores entre otras. (Kendall, 2006)

1.6.1- Visual Paradigm

Es una herramienta profesional que soporta el ciclo de vida completo del desarrollo de un software. Contribuye a la rápida elaboración de aplicaciones de calidad y con un menor costo. Permite modelar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación. (Orallo, 2005)

Entre sus características se encuentran:

- Ingeniería inversa: Código a modelo, código a diagrama.
- Editor de Detalles de Casos de Uso: Entorno todo en uno, para la especificación de los detalles de los casos de uso, incluyendo la especificación del modelo general y de las descripciones de los casos de uso.

Esta herramienta utiliza UML como lenguaje de modelado, mediante el cual es posible establecer una serie de requerimientos y estructuras necesarias para plasmar un sistema de software previo al proceso intensivo de escribir código.

1.6.2- Rational Rose

Es una herramienta de software para el Modelado Visual mediante UML de la familia Rational Software de IBM para el despliegue, diseño, construcción, pruebas y administración de proyectos en el proceso de desarrollo de software.

Rational Rose domina el mercado de herramientas para el análisis, modelamiento, diseño y construcción orientado a objetos. Rational Rose permite visualizar, entender, y refinar los requerimientos y arquitectura antes de enfrentar el código. Otras características de la herramienta son:

- Soporte para análisis de patrones ANSI C++, Rose J y Visual C++.
- Característica de control por separado de componentes modelo que permite una administración más granular y el uso de modelos.

- Soporte de ingeniería Forward y/o reversa para algunos de los conceptos más comunes.
- La generación de código Ada, ANSI C ++, C++, CORBA, Java y Visual Basic, con capacidad de sincronización modelo-código configurables.
- Capacidad de análisis de calidad de código.
- El Add-In para modelado Web provee visualización, modelado y las herramientas para desarrollar aplicaciones de Web.
- Modelado UML para trabajar en diseños de base de datos, con capacidad de representar la integración de los datos y los requerimientos de aplicación a través de diseños lógicos y físicos.
- Capacidad de crear definiciones de tipo de documento XML para el uso en la aplicación.
- Integración con otras herramientas de desarrollo de Rational.
- Publicación web y generación de informes para optimizar la comunicación dentro del equipo.

1.6.3- Umbrello

Umbrello es una herramienta libre para crear y editar diagramas UML, que ayuda en el proceso del desarrollo de software. Fue desarrollada por Paul Hensgen, y está diseñado principalmente para KDE, aunque funciona en otros entornos de escritorio.

Umbrello maneja gran parte de los diagramas estándar UML pudiendo crearlos, además de manualmente, importándolos a partir de código en C++, Java, Python, IDL, Pascal/Delphi, Ada, o también Perl (haciendo uso de una aplicación externa). Así mismo, permite crear un diagrama y generar el código automáticamente en los lenguajes antes citados, entre otros. El formato de fichero que utiliza está basado en XMI. También permite la distribución de los modelos exportándolos en los formatos DocBook y XHTML, lo que facilita los proyectos colaborativos donde los desarrolladores no tienen acceso directo a Umbrello o donde los modelos van a ser publicados vía web. (Enríquez, 2007)

En la actualidad, Umbrello permite la creación de los siguientes tipos de diagramas:

- Diagrama de casos de uso.
- Diagrama de componentes.
- Diagrama de despliegue.
- Diagrama de modelo entidad-relación.
- Diagrama de clases.
- Diagrama de secuencia.
- Diagrama de estados.
- Diagrama de actividades.
- Diagrama de colaboración.

Conclusiones del capítulo

En este capítulo se abordó el estado del arte de los escenarios de aprendizaje donde se pudo apreciar los diferentes tipos de escenarios de aprendizaje existentes y sus características. Se enfatizó en la definición de lenguaje XML y en el tipo de archivo (scene) y su importancia para el proyecto. Se realizó un análisis de los diferentes tipos de editores XML existentes y se tomaron como ejemplo tres de los más destacados en la actualidad. Al culminar el capítulo conocimos las diferentes herramientas y metodologías existentes que se podrían elegir para el desarrollo del trabajo.

CAPÍTULO 2: DESCRIPCIÓN DE LA SOLUCIÓN PROPUESTA

Introducción

En este capítulo se seleccionarán las diferentes herramientas, lenguajes y metodología a utilizar en el desarrollo de la investigación planteada. A demás se realizará el levantamiento de los requisitos funcionales y no funcionales, modelo de dominio, diagrama de casos de uso con sus descripciones y los patrones de diseño utilizados.

2.1- Selección de los lenguajes, metodología y herramientas a utilizar

2.1.1- Lenguajes de programación

C++:

Para la implementación de la aplicación se utilizará el paradigma de programación Orientado a Objetos y el lenguaje de programación de alto nivel C++, por las funcionalidades que brinda este lenguaje y porque es el lenguaje en que se tiene mayor experiencia por parte del equipo de desarrolladores del proyecto. C++ es un lenguaje de programación de propósito general, especialmente indicado para la programación de sistemas por su flexibilidad y su potencia.

XML:

Se hace uso del lenguaje de marcas XML para el trabajo con los datos, ya que es el lenguaje por el cual está constituida la sintaxis de los tipos de archivos scene usados en los laboratorios virtuales por las oportunidades que estos tipos de archivos y este lenguaje brindan. XML permite al programador y los soportes dedicar sus esfuerzos a las tareas importantes cuando trabaja con los datos, ya que algunas tareas tediosas como la validación de estos o el recorrido de las estructuras corre a cargo del lenguaje y está especificado por el estándar, de modo que el programador no tiene que preocuparse por ello.

2.1.2- Lenguaje de modelado

UML:

Para el modelado se escogió UML. El Lenguaje Unificado de Modelado por sus siglas en ingles, prescribe una notación estándar y semánticas esenciales para el modelado de un sistema orientado a objetos.

Es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema. UML ofrece un estándar para describir un modelo del sistema, incluyendo aspectos conceptuales tales como procesos de negocio y funciones del sistema, y aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y componentes reutilizables. (Larman, 2004.)

2.1.3- Metodología de ingeniería de software

RUP:

Rational Unified Process es la metodología a utilizar en el proceso de desarrollo de este trabajo, ya que RUP constituye la metodología estándar más utilizada para el análisis, implementación y documentación de sistemas. También influyó en su adopción la idea de que es la metodología usada en el desarrollo de los laboratorios virtuales dentro del proyecto y por las características que a continuación se referencia.

Principales características que influyen en su utilización:

- RUP no es un sistema con pasos firmemente establecidos, sino un conjunto de metodologías adaptables al contexto y necesidades de cada organización.
- RUP mantiene al equipo enfocado en producir incrementalmente software operativo a tiempo, con las características requeridas y con la calidad requerida.
- Las mejores prácticas probadas en la industria, contenidas en el RUP, incorporan las lecciones aprendidas de cientos de líderes de la industria y miles de proyectos.
- Ya no hay necesidad de re-inventar soluciones a desafíos de la ingeniería de software bien conocidos. Siguiendo el acercamiento al desarrollo iterativo del RUP, es posible entregar a tiempo y con confianza el software. (Jacobson, 2000.)

2.1.4- Herramientas a utilizar

Visual Paradigm:

Se seleccionó Visual Paradigm como herramienta CASE a utilizar debido a que está basada en UML, soporta el ciclo de vida completo del desarrollo del software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. Permite dibujar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación. La herramienta UML CASE también proporciona abundantes tutoriales de UML, demostraciones interactivas de UML y proyectos UML. Cuenta con una licencia gratuita y comercial.

QT:

Para la implementación del trabajo se escogió utilizar el Framework QT ya que es el framework utilizado actualmente en el proyecto Laboratorios Virtuales, por las posibilidades que presenta implícitas para el manejo de documentos, además de las grandes posibilidades que brinda para los diseños gráficos, como los editores de propiedades a utilizar en el desarrollo de la aplicación.

Qt es un framework que se utiliza para el desarrollo de aplicaciones de software con una interfaz gráfica de usuario y también se utiliza para el desarrollo de programas que no son GUI como herramientas de línea de comandos y las consolas para los servidores.

Qt usa estándar de C++, pero hace un amplio uso de un generador de código especial (llamado el Meta Object Compiler, o MOC), junto con varias macros para enriquecer el lenguaje.

Qt también se puede utilizar en varios lenguajes de programación a través de enlaces de lenguaje. Qt proporciona portabilidad de una sola fuente a través de MS Windows, Mac OS X, Linux, y todas las variantes comerciales más importantes de Unix. Qt también está disponible para dispositivos embebidos como Qt para Embedded Linux y Qt para Windows CE.

Es producido por Qt división de Nokia, que entró en vigor después de la adquisición de Nokia de la empresa noruega Trolltech, el productor original de Qt. Distribuido bajo los términos de la GNU Lesser General Public License, Qt es un software libre y de código abierto.

2.2- Modelo de dominio

Un Modelo de Dominio, es una representación visual estática del entorno real del proyecto. El objetivo del modelado del dominio es comprender y describir las clases más importantes dentro del contexto del sistema y ayudar a los usuarios, clientes, desarrolladores y otros interesados a utilizar un vocabulario común. (Jacobson, 2000.)

En la siguiente figura se muestra el modelo de dominio:

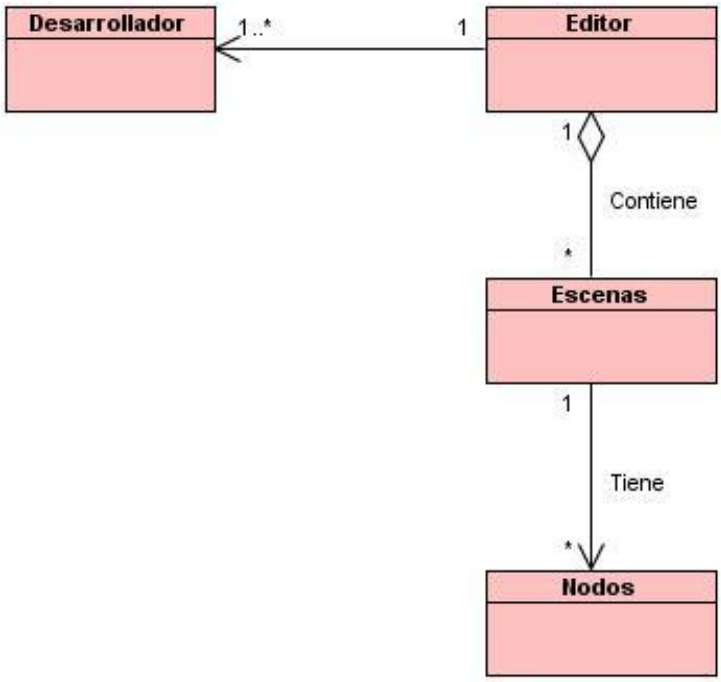


Figura 8 Modelo de Dominio

2.2.1- Definición de clases del modelo de dominio

Tabla 2 Definición de clases del modelo del dominio

Nombre de la clase	Definición
Desarrollador.	Representa a la persona que interactuará con la aplicación.
Editor.	Representa a la aplicación
Escenas.	Se refiere a los ficheros scene que serán tratados en el editor.
Nodos	Se refiere al contenido de la escena, organizada en nodos.

2.3- Captura de requisitos

Un requerimiento: es una condición o capacidad que tiene que ser alcanzada o poseída por un sistema o componente de un sistema, para satisfacer un contrato, estándar u otro documento impuesto formalmente. (Rumbaugh, 2000)

2.3.1- Requisitos funcionales

Un requerimiento funcional define el comportamiento interno del software: cálculos, detalles técnicos, manipulación de datos y otras funcionalidades específicas que muestran cómo las historias de usuario serán llevadas a la práctica. Los requerimientos funcionales se mantienen invariables sin importar con qué cualidades o propiedades se relacionen.

A continuación se muestran los requerimientos funcionales:

Tabla 3 Requisitos funcionales

Requisitos funcionales	Descripción
RF 1 Crear escena.	El sistema debe permitir crear una nueva escena.
RF 2 Modificar escena.	El sistema debe permitir modificar una escena que se cargue.
RF 3 Cargar escena.	El sistema debe permitir cargar una escena.
RF 4 Guardar escena.	El sistema debe permitir guardar las escenas que se creen y las que se modifiquen.
RF 5 Insertar nodo.	El sistema debe permitir insertar un nodo dentro de la escena.
RF 6 Eliminar nodo.	El sistema debe permitir eliminar un nodo dentro de la escena.

2.3.2- Requisitos no funcionales

Los requerimientos no funcionales son propiedades o cualidades que el producto debe tener. En muchos casos, estos requerimientos son fundamentales en el éxito del producto y generalmente

están vinculados a requerimientos funcionales, además corresponden a aspectos tales como la disponibilidad, soporte, seguridad, apariencia e interfaz externa entre otros.

Apariencia e interfaz externa: Se debe proporcionar un ambiente agradable y fácil para el usuario.

Usabilidad: El acceso a la aplicación debe realizarse de forma fácil y rápida. Cualquier operación debe realizarse con menos de 2 clics. Puede ser usada por personas que tengan conocimientos básicos en aplicaciones.

Rendimiento: La capacidad de respuesta de la aplicación debe ser rápida.

Soporte: Una vez concluida la aplicación, se realizarán varias pruebas para comprobar su funcionalidad. Con la puesta en práctica de la aplicación deben quedar satisfechas las necesidades de los usuarios.

Software: La aplicación permite ejecutarse en cualquier versión de los sistemas operativos MS Windows, Linux, Mac OS X y otras variantes de UNIX.

Hardware: Para la ejecución de la aplicación se precisará contar con:

- RAM mínimo de 256 MB.
- Microprocesador Intel Pentium de más de 1GHz.
- HDD mínimo 1 GB de almacenamiento.

Disponibilidad: La aplicación debe estar disponible a tiempo completo, y debe recuperarse rápidamente ante cualquier tipo de fallo.

Requisitos legales: Las tecnologías y herramientas que se utilicen para desarrollar la aplicación deben estar bajo la licencia de software libre.

Restricciones de diseño e implementación: La aplicación será implementada con el lenguaje de programación C++ y el paradigma Orientado a Objeto, como IDE de desarrollo se utilizará QT Creator. Se utilizará un estándar de codificación para la implementación.

2.4- Modelo de casos de uso del sistema

2.4.1- Actores del sistema

Tabla 4 Definición de los actores del sistema

Actor	Descripción
Desarrollador	Es quien interactúa con el editor para explotar las funcionalidades del mismo.

2.4.2- Diagrama de casos de uso del sistema

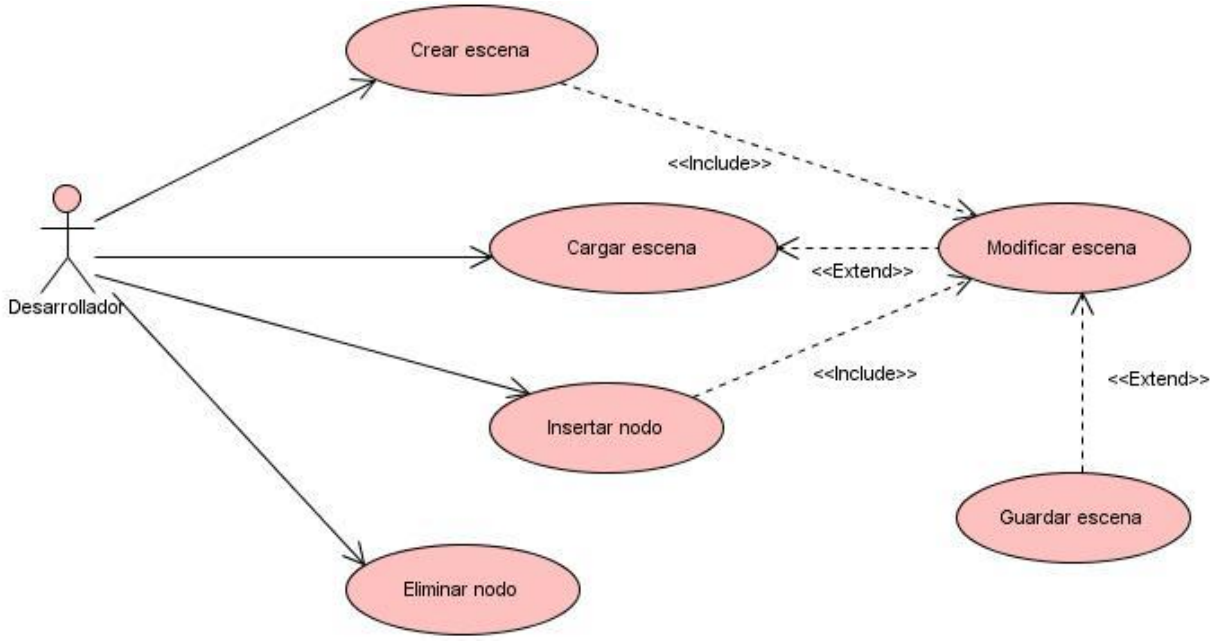


Figura 9 Diagrama de casos de uso del sistema

2.4.3- Descripción de casos de uso del sistema

Tabla 5 Caso de uso: Crear Escena

Caso de Uso:	Crear Escena.
Actores:	Desarrollador.
Resumen:	El caso de uso comienza cuando el usuario elige la opción de crear una nueva escena. El sistema le muestra la escena creada.
Referencias:	RF 1, RF 2.
Prioridad:	Crítica.

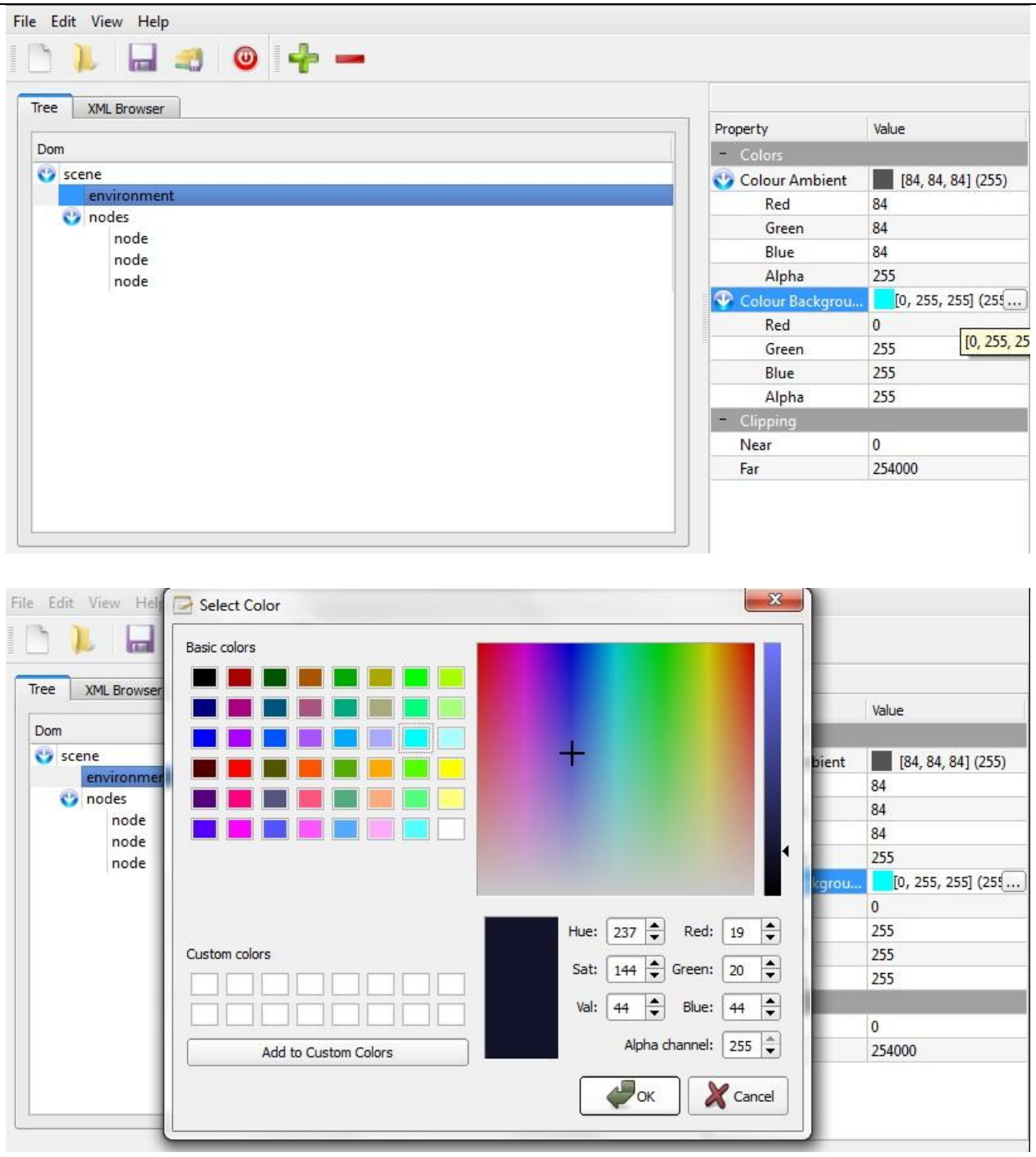
Flujo Normal de Eventos	
Acción del Actor:	**Respuesta del Sistema:**
1- Selecciona la opción de crear una escena.	1.1- Muestra la nueva escena.
2- Modifica las propiedades de la escena (Ir a Caso de Uso Modificar Escena).	

Prototipo de Interfaz:

Pos condiciones:	El sistema visualiza la escena creada con sus propiedades.

Tabla 6 Caso de uso: Modificar Escena

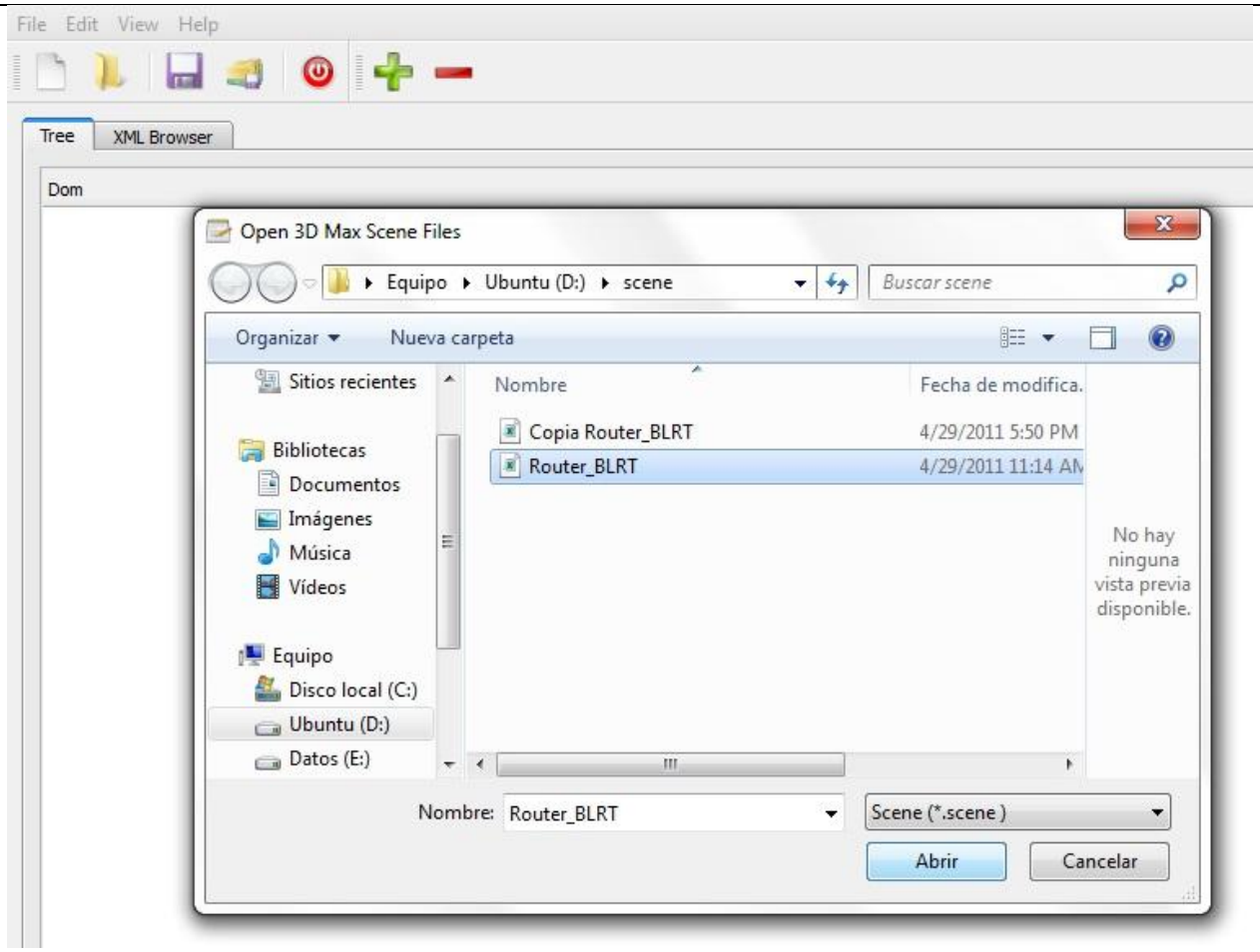
Caso de Uso:	Modificar Escena.
Actores:	Desarrollador.
Resumen:	El caso de uso inicia cuando el actor modifica las propiedades de la escena. El sistema modifica dichas propiedades.
Precondiciones:	Deben haberse ejecutado los casos de uso Crear Escena o Cargar Escena.
Referencias:	RF 1, RF 2, RF 3, RF 4.
Prioridad:	Crítica.
Flujo Normal de Eventos	
Acción del Actor:	**Respuesta del Sistema:**
1- Escoge el nodo a modificar.	1.1- Muestra las propiedades del nodo.
2- Modifica los valores de las propiedades.	2.1- Modifica las propiedades del nodo.
Flujos Alternos	
Acción del Actor:	**Respuesta del Sistema:**
1- Selecciona la opción de guardar escena (Ir a Caso de Uso Guardar Escena).	
Prototipo de Interfaz:	



Pos condiciones:	El sistema visualiza las propiedades modificadas.

Tabla 7 Caso de uso: Cargar Escena

Caso de Uso:	Cargar Escena.
Actores:	Desarrollador.
Resumen:	El caso de uso se inicia cuando el actor selecciona la opción de cargar la escena. Selecciona el fichero *.SCENE a cargar. El sistema procede a cargar la escena.
Referencias:	RF 2, RF 3.
Prioridad:	Crítica.
Flujo Normal de Eventos	
Acción del Actor:	**Respuesta del Sistema:**
1- Selecciona la opción de cargar la escena.	1.1- Muestra un cuadro de diálogo que le permitirá al actor seleccionar el directorio y el fichero que desea cargar.
2- Selecciona el directorio y fichero que desea cargar.	2.1- El cuadro de diálogo de abrir fichero solo mostrará los ficheros de la extensión *.SCENE.
3- Oprime el botón abrir.	3.1- Cierra el cuadro de diálogo y se procede a cargar el fichero seleccionado. 3.2- Muestra en pantalla el contenido de la escena.
Flujos Alternos	
Acción del Actor:	**Respuesta del Sistema:**
1- Modifica las propiedades de la escena cargada (Ir a Caso de uso Modificar Escena.).	
Prototipo de Interfaz:	



Pos condiciones:	El sistema visualiza la escena cargada.

Tabla 8 Caso de uso: Guardar Escena

Caso de Uso:	Guardar Escena.
Actores:	Desarrollador.
Resumen:	El caso de uso comienza cuando el actor elige la opción de guardar la escena. Escoge el directorio donde desea guardarla. El sistema procede a guardar la escena en el directorio escogido.
Precondiciones:	Deben haberse ejecutado los casos de uso Crear Escena o Cargar Escena.
Referencias:	RF 1, RF 3, RF 4.

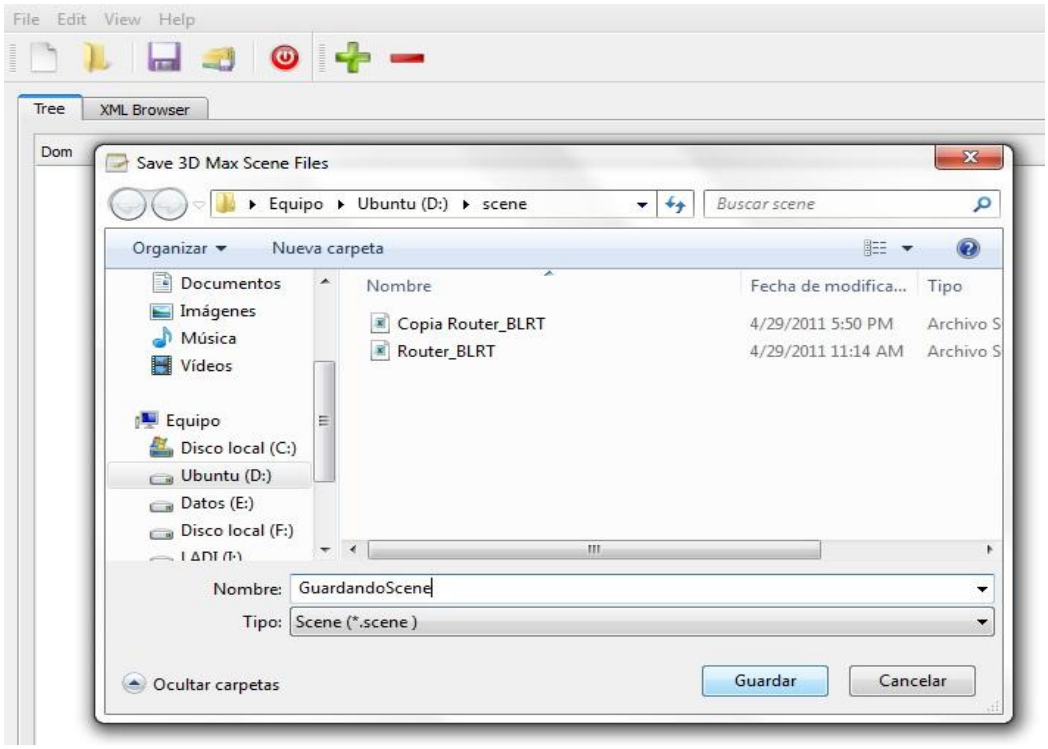
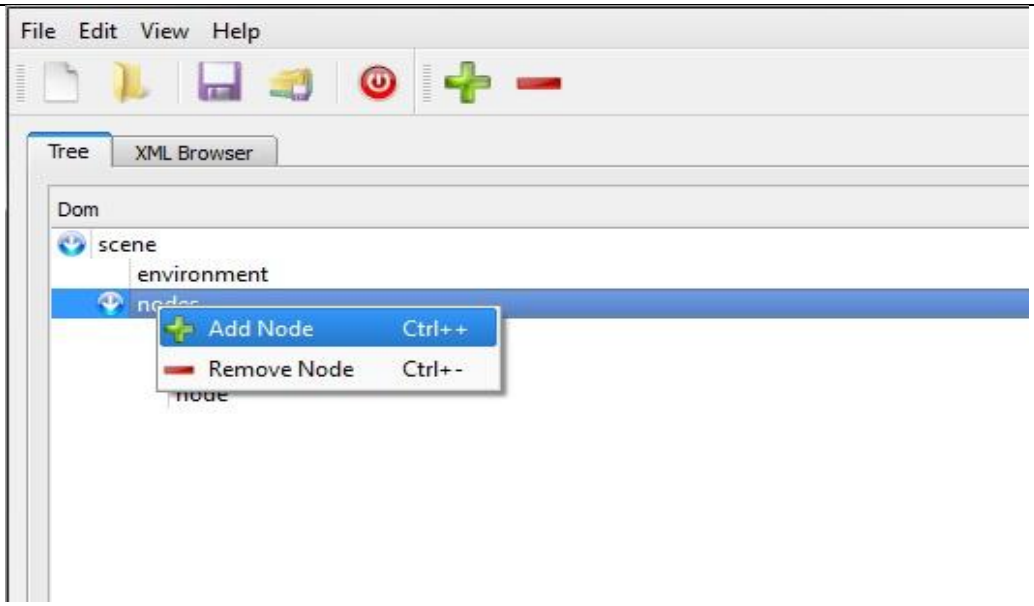
Prioridad:	Crítica.
Flujo Normal de Eventos	
Acción del Actor:	**Respuesta del Sistema:**
1- Selecciona la opción de guardar la escena.	1.1- Muestra un cuadro de diálogo para que el actor escoja el directorio donde será guardado y nombre del fichero.
2- Selecciona el directorio y escoge el nombre que desee. 3- Oprime el botón guardar.	3.1- Cierra el cuadro de diálogo y guarda en el directorio seleccionado.
Prototipo de Interfaz: 	
Pos condiciones:	El sistema guarda el fichero satisfactoriamente.

Tabla 9 Caso de uso: Insertar Nodo

Caso de Uso:	Insertar Nodo.
Actores:	Desarrollador.
Resumen:	El caso de uso comienza cuando el actor elige la opción de añadir un nodo. El sistema procede a añadir el nodo dentro del nodo padre seleccionado.
Precondiciones:	Deben haberse ejecutado los casos de uso Crear Escena o Cargar Escena.
Referencias:	RF 1,RF 2, RF 3, RF 5
Prioridad:	Crítica.
Flujo Normal de Eventos	
Acción del Actor:	**Respuesta del Sistema:**
1- Selecciona el nodo padre donde desea añadir el nodo. 2- Elige la opción añadir nodo.	2.1- Muestra en pantalla el nuevo nodo añadido.
3- Modifica las propiedades del nodo añadido (Ir al Caso de Uso Modificar Escena).	
Prototipo de Interfaz:	



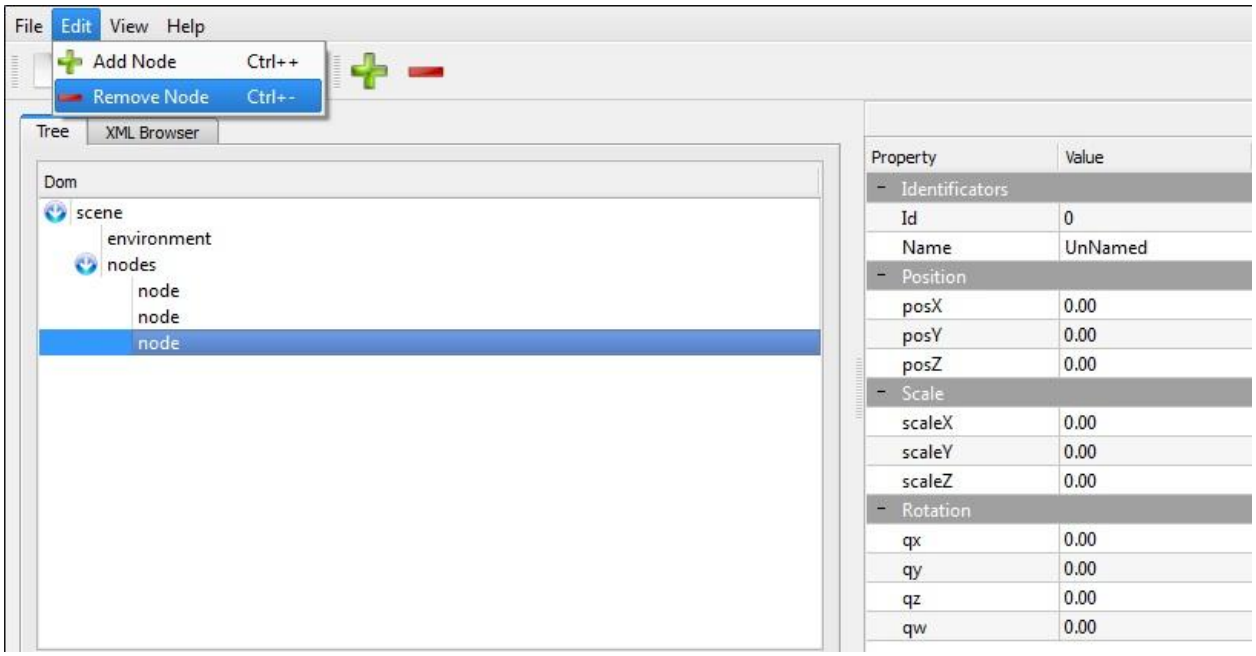
Pos condiciones:	El sistema visualiza el nodo añadido.

Tabla 10 Caso de uso: Eliminar Nodo

Caso de Uso:	Eliminar nodo
Actores:	Desarrollador
Resumen:	El caso de uso se inicia cuando el actor escoge la opción de eliminar nodo. El sistema procede a eliminar el nodo seleccionado.
Precondiciones:	Deben haberse ejecutado los casos de uso Crear Escena o Cargar Escena.
Referencias:	RF 1, RF 3, RF 6
Prioridad:	Crítica

Flujo Normal de Eventos	
Acción del Actor:	**Respuesta del Negocio:**
1- Selecciona el nodo que desea eliminar. 2- Selecciona la opción eliminar nodo.	2.1- Elimina el nodo escogido.

Prototipo de Interfaz:



Pos condiciones:	El sistema elimina el nodo seleccionado.

2.5- Patrones de diseño

Un patrón de diseño es un conjunto de reglas que describen cómo afrontar tareas y solucionar problemas que surgen durante el desarrollo de software, es la solución recurrente para un problema típico en un contexto determinado. La solución se refiere a la respuesta al problema, por lo que ayuda a resolver las dificultades de diseño en problemas similares. Los patrones comunican soluciones de diseño a los desarrolladores y arquitectos que los leen y los utilizan. A continuación se muestran algunos de los patrones más relevantes utilizados en el diseño de la propuesta de solución:

Controlador: Este patrón funciona como intermediario entre una determinada interfaz y el algoritmo que la implementa, de tal forma que esta recibe los datos y los envía a las distintas clases según el método llamado, la utilización de este patrón se evidencia en la clase de interfaz del dominio, ya que esta maneja el sistema de forma global.

Alta cohesión y bajo acoplamiento: Estos patrones se pueden separar, aunque están íntimamente ligados, de hecho si se aumenta mucho la cohesión del sistema, se tiene un alto

acoplamiento entre las clases, y por el contrario si reduce mucho el acoplamiento, se verá mermada la cohesión.

Alta cohesión: este patrón propone asignar la responsabilidad de manera que la complejidad se mantenga dentro de límites manejables asumiendo solamente las responsabilidades que deben manejar, evadiendo un trabajo excesivo. Su utilización mejora la claridad y facilidad con que se entiende el diseño, simplifica el mantenimiento y las mejoras de funcionalidad, generan bajo acoplamiento, soporta mayor capacidad de reutilización. Este patrón se pone de manifiesto en la mayoría de las clases porque cada una es capaz de realizar sus responsabilidades sin la utilización de las demás.

Bajo acoplamiento: Plantea la idea de tener las clases lo más desacopladas que se pueda. De tal forma que en caso de producirse una modificación en alguna de ellas, tenga la mínima repercusión posible en el resto de clases, potenciando la reutilización, y disminuyendo la dependencia entre las clases, este patrón se evidencia en la mayoría de las clases pues no dependen de ninguna otra clase para realizar sus responsabilidades.

Conclusiones del capítulo:

En este capítulo se hizo referencia a las herramientas, lenguaje y metodología de desarrollo a utilizar. Todas las herramientas están bajo la distribución de software libre. Se realizó además una descripción de los casos de uso críticos del sistema que facilitarán la gestión de los requerimientos funcionales y no funcionales. Culmina el capítulo con la descripción de los patrones de diseño utilizados.

CAPÍTULO 3: DISEÑO ARQUITECTÓNICO DEL SISTEMA

Introducción

En este capítulo se realiza el diseño arquitectónico del sistema primeramente conociendo los estándares de codificación utilizados en la implementación de la aplicación. Continúa el capítulo con una representación del diagrama de clases del diseño y la descripción previa de cada clase. Más adelante en el capítulo se evidencian los diagramas de componentes y de secuencia del sistema. Terminando con las diferentes pruebas de caja negra realizadas al software.

3.1- Estándar de codificación utilizado

El código del editor sigue algunos de los estándares de codificación para C++ utilizados actualmente, con modificaciones para hacerlo de un estilo personal. Está programado en idioma inglés en su totalidad, debido a que las palabras son mucho más simples, no se acentúan y es el idioma universal en el mundo informático. El conocimiento de los estándares que se relacionan a continuación permitirá un mayor entendimiento del código fuente.

Declaración de variables: Los nombres de las variables comenzarán con el "_", en caso de ser más de una palabra, siguen a continuación todas las palabras que tenga comenzando con mayúsculas, como se muestra a continuación.

QColor _colorBackground;

En el caso de que sean variables globales se les antepondrá la letra "g" luego del "_", y en caso de ser argumentos de algún método, comenzarán en minúscula sin anteponer "_". A continuación ejemplos de ambos:

DomManager _ gDomManager;

void UpdateScene (QtProperty property, QVariant value);

Las constantes se nombrarán con mayúsculas, con la misma definición de las variables anteponiendo el "_" y separadas también por "_", ejemplo:

_CONST_ZERO = 0;

Declaración de métodos: En el caso de los métodos, comenzarán por minúsculas y las demás palabras en mayúscula, el caso de los argumentos de estos ya fue explicado anteriormente. Algunos ejemplos de ello:

*void processSelectedItem (QTreeWidgetItem *item);*
void showSceneProperties ();

Los constructores y destructores, como lo exigen los compiladores, llevan el nombre de la clase.

*DomManager (QTreeWidget *treeWidget);*
~DomManager ();

3.2- Diagrama de clases del diseño

A continuación se muestra el diagrama de clases del diseño pero son válidas algunas aclaraciones con anterioridad:

- Para no extender el diagrama de clases se obviaron los tradicionales métodos "*gets*" y "*sets*" así como los destructores de las clases.
- La nomenclatura utilizada en el diagrama se pudo observar en el epígrafe anterior para una mejor comprensión del mismo.

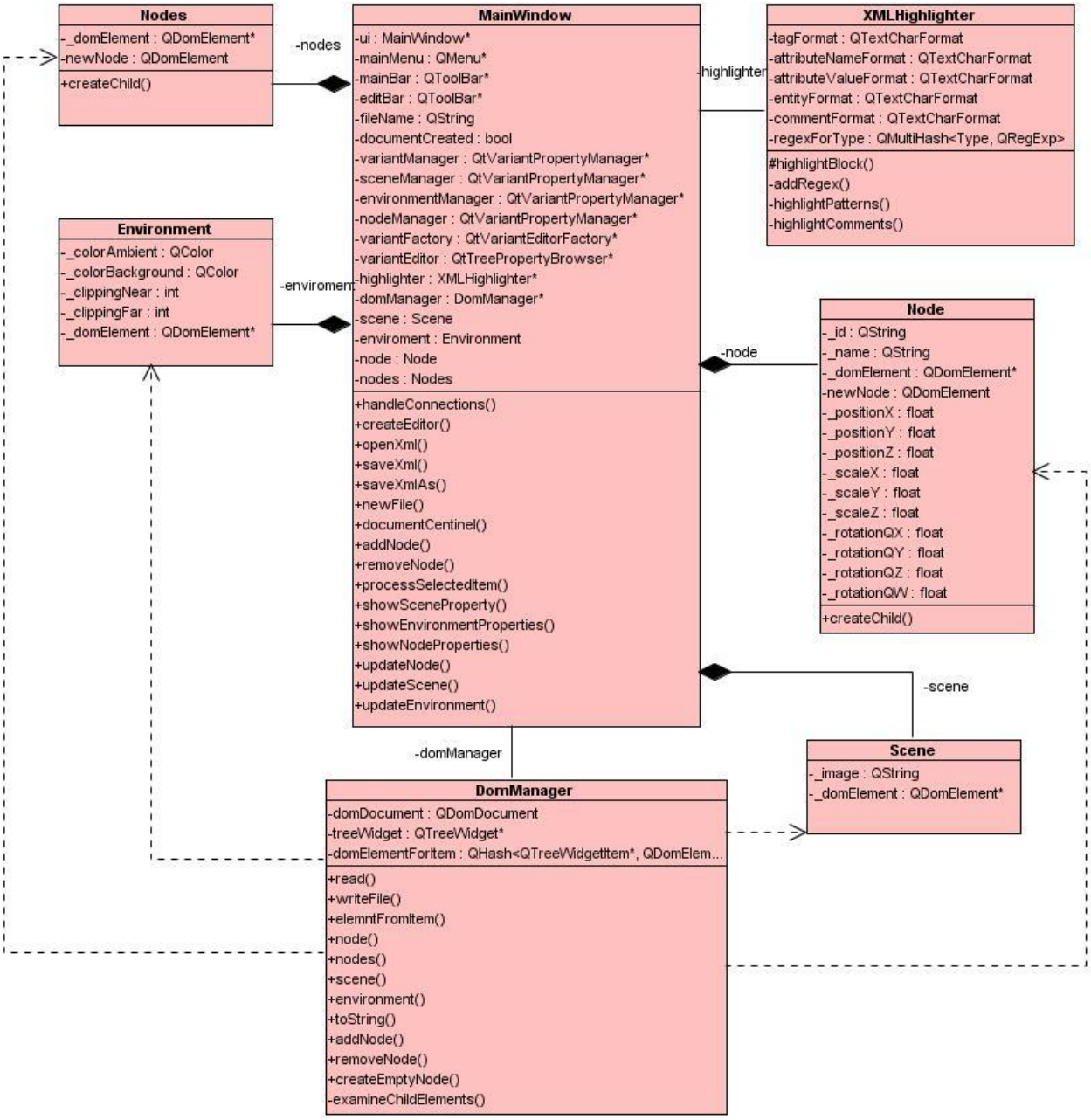


Figura 10 Diagrama de clases del diseño

3.2.1- Descripción de las clases del diseño

A continuación se muestran las diferentes tablas con las descripciones de cada clase del sistema. En la descripción de las clases también se obvian los métodos “set” y “gets” tradicionales para no extender el contenido de las mismas.

Tabla 11 Descripción de la clase DomManager

Nombre:	DomManager
Tipo de clase:	Control
Atributo:	**Tipo:**
domDocument	*QDomDocument*
**treeWidget*	*QTreeWidget*
domElementForItem;	*QHash<QTreeWidgetItem *, QDomElement>*
Por cada responsabilidad:	
Nombre:	*DomManager*
Descripción:	Constructor de la clase
Nombre:	*read*
Descripción:	Este método se encarga de leer el fichero XML. A medida que se recorre el fichero se van creando los objetos correspondientes a cada etiqueta del mismo.
Nombre:	*writeFile*
Descripción:	A través de este método se salva el estado en el que se encuentra el fichero que se está editando.
Nombre:	*elemntFromItem*
Descripción:	Este método busca el elemento DOM que está asociado al item seleccionado en ese momento y lo retorna.
Nombre:	*Node*
Descripción:	Este método retorna el elemento del DOM node.
Nombre:	*nodes*

Descripción:	Este método retorna el elemento del DOM nodes.
Nombre:	*scene*
Descripción:	Este método retorna el elemento del DOM scene.
Nombre:	*environment*
Descripción:	Este método retorna el elemento del DOM environment.
Nombre:	*addNode*
Descripción:	Este método añade un nodo hijo del item donde se está situado.
Nombre:	*removeNode*
Descripción:	Este método elimina el nodo marcado.
Nombre:	*createEmptyNode*
Descripción:	Este método se utiliza para crear un nuevo nodo con sus propiedades vacías.
Nombre:	*examineChildElement*
Descripción:	Este método examina las etiquetas, si la etiqueta que se está examinando es válida se crea un objeto de la misma y si no, se busca la siguiente y se descarta la que se está tratando.

Tabla 12 Descripción de la clase MainWindow

Nombre:	MainWindow
Tipo de clase:	Interfaz
Atributo:	**Tipo:**
ui	*MainWindow*

mainMenu	*QMenu*
mainBar	*QToolBar*
editBar	*QToolBar*
fileName	*QString*
documentCreated	*bool*
variantManager	*QVariantPropertyManager*
sceneManager	*QVariantPropertyManager*
environmentManager	*QVariantPropertyManager*
nodeManager	*QVariantPropertyManager*
variantFactory	*QtVariantEditorFactory*
variantEditor	*QtTreePropertyBrowser*
highlighter	*XMLHighlighter*
domManager	*DomManager*
scene	*Scene*
environment	*Environment*
nodes	*Nodes*
node	*Node*

Por cada responsabilidad:	
Nombre:	*MainWindow*
Descripción:	Constructor de la clase
Nombre:	*handleConnections*
Descripción:	En este método se conectan todas las señales y los slots.

Nombre:	*createEditor*
Descripción:	Se encarga de crear el editor y todos los componentes que contienen las propiedades de los elementos seleccionados.
Nombre:	*openXml*
Descripción:	Método que se encarga de abrir el archivo scene desde un directorio.
Nombre:	*saveXml*
Descripción:	Método que permite guardar los cambios hechos en el archivo creado o cargado.
Nombre:	*saveXmlAs*
Descripción:	Método que permite guardar el archivo con otra extensión y en otros directorios.
Nombre:	*newFile*
Descripción:	Método que permite la creación de una nueva escena vacía.
Nombre:	*documentCentinel*
Descripción:	Informa si es posible guardar el documento en el cual se está trabajando.
Nombre:	*addNode*
Descripción:	Método que captura el item al que se quiere añadir un nodo hijo y llama al método añadir nodo del DOM.
Nombre:	*removeNode*
Descripción:	Método que captura el nodo que se quiere eliminar y llama al método eliminar nodo del DOM.
Nombre:	*processSelectedItem*

Descripción:	Clasifica el item que se encuentra seleccionado y llama a los métodos que muestran las propiedades del mismo.
Nombre:	*showSceneProperty*
Descripción:	Método que muestra las propiedades del scene.
Nombre:	*showEnvironmentProperties*
Descripción:	Método que muestra las propiedades del environment.
Nombre:	*showNodeProperties*
Descripción:	Método que muestra las propiedades del node.
Nombre:	*updateNode*
Descripción:	Actualiza las propiedades del node.
Nombre:	*updateScene*
Descripción:	Actualiza las propiedades del scene.
Nombre:	*updateEnvironment*
Descripción:	Actualiza las propiedades del environment.

Tabla 13 Descripción de la clase Scene

Nombre:	Scene
Tipo de clase:	Entidad
Atributo:	**Tipo:**
image	*QString*
domElement	*QDomElement*

Por cada responsabilidad:	
Nombre:	*Scene*
Descripción:	Constructor de la clase

Tabla 14 Descripción de la clase Environment

Nombre:	Environment
Tipo de clase:	Entidad
Atributo:	**Tipo:**
colorAmbient	*QColor*
colorBackground	*QColor*
clippingNear	*int*
clippingFar	*int*
domElement	*QDomElement*
Por cada responsabilidad:	
Nombre:	*Environment*
Descripción:	Constructor de la clase

Tabla 15 Descripción de la clase Nodes

Nombre:	Nodes
Tipo de clase:	Entidad
Atributo:	**Tipo:**
domElement	*QDomElement*
newNode	*QDomElement*

Por cada responsabilidad:	
Nombre:	*Nodes*
Descripción:	Constructor de la clase
Nombre:	*createChild*
Descripción:	Crea un nuevo nodo hijo para almacenar los datos de la siguiente etiqueta.

Tabla 16 Descripción de la clase Node

Nombre:	Node
Tipo de clase:	Entidad
Atributo:	**Tipo:**
domElement	*QDomElement*
newNode	*QDomElement*
id	*QString*
name	*QString*
positionX	*float*
positionY	*float*
positionZ	*float*
scaleX	*float*
scaleY	*float*
scaleZ	*float*
rotationQX	*float*
rotationQY	*float*

rotationQZ	*float*
rotationQW	*float*
Por cada responsabilidad:	
Nombre:	*Nodes*
Descripción:	Constructor de la clase
Nombre:	*createChild*
Descripción:	Crea un nuevo nodo hijo para almacenar los datos de la siguiente etiqueta.

3.3- Diagramas de secuencia

El diagrama de secuencia es un tipo de diagrama usado para modelar la interacción entre objetos en un sistema según UML. Un diagrama de secuencia muestra la interacción de un conjunto de objetos en una aplicación a través del tiempo y se modela para cada caso de uso. Mientras que el diagrama de casos de uso permite el modelado de una vista business del escenario, el diagrama de secuencia contiene detalles de implementación del escenario, incluyendo los objetos y clases que se usan para implementar el escenario, y mensajes intercambiados entre los objetos.

A continuación se muestran los diagramas de secuencia agrupados por casos de uso:

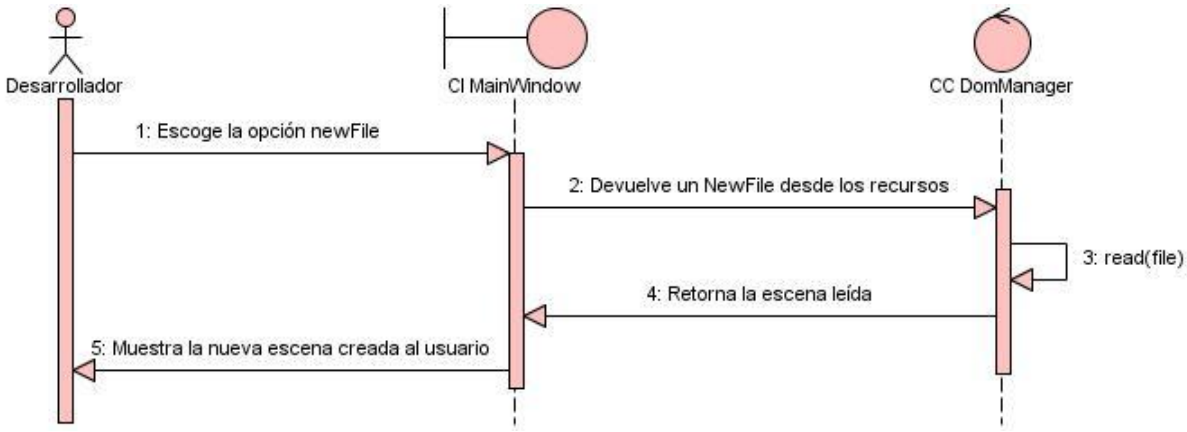


Figura 11 Diagrama de secuencia CU Crear Escena

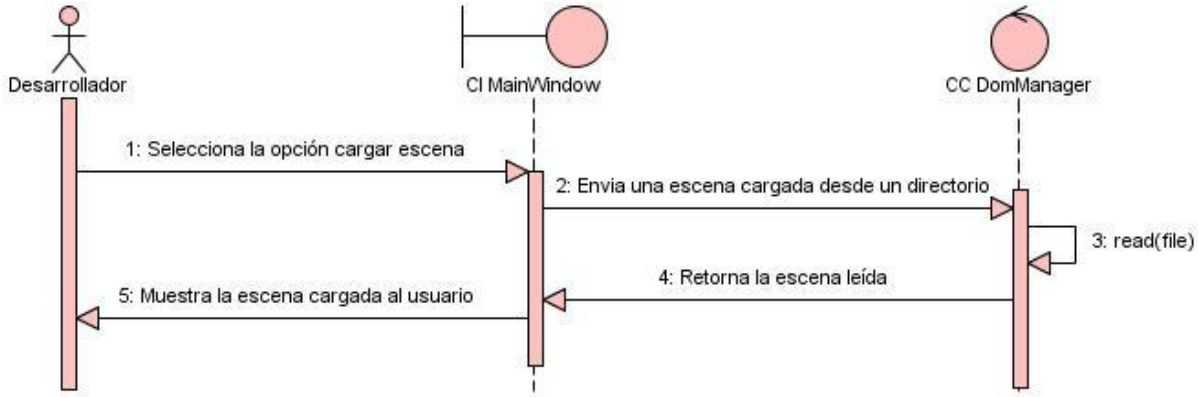


Figura 12 Diagrama de secuencia CU Cargar Escena

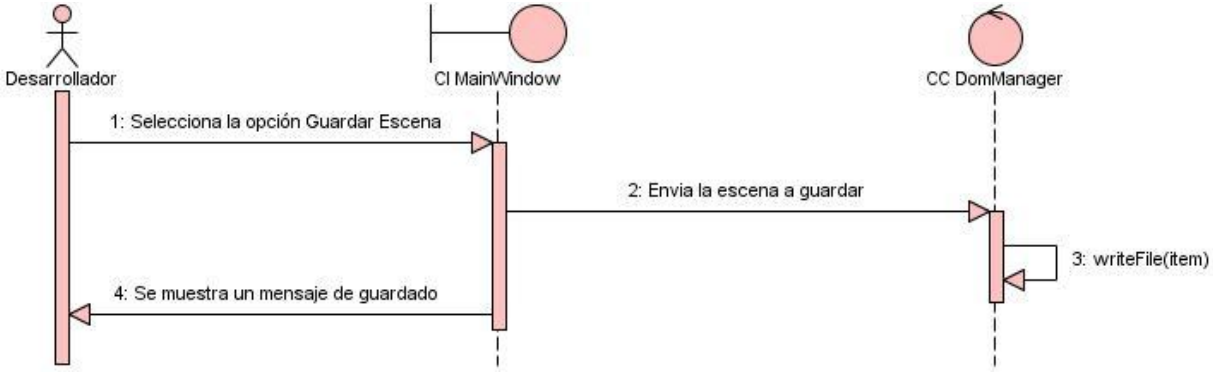


Figura 13 Diagrama de secuencia CU Guardar Escena

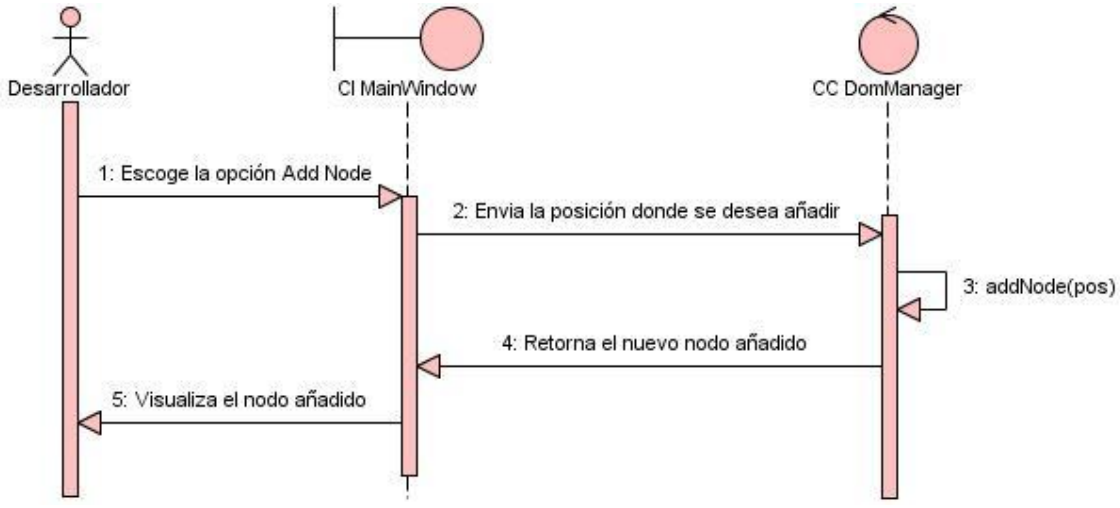


Figura 14 Diagrama de secuencia CU Insertar Nodo

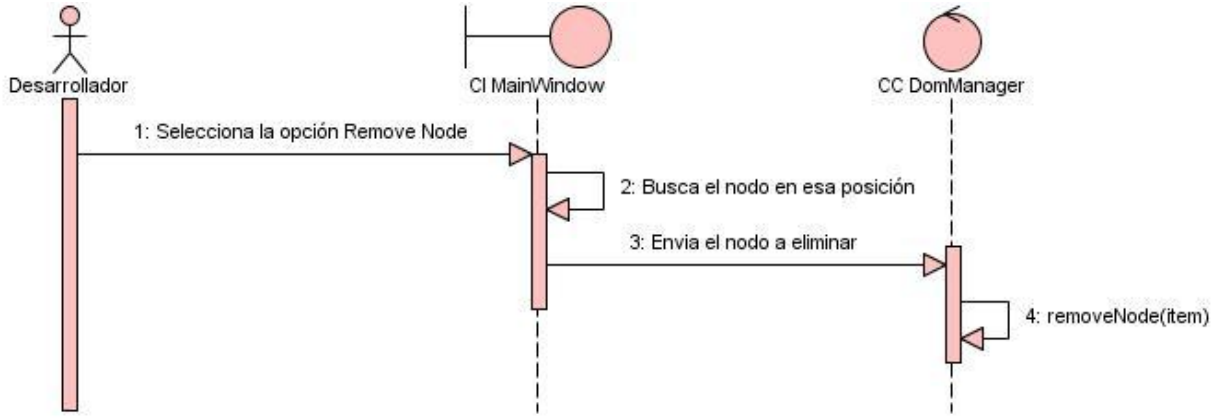


Figura 15 Diagrama de secuencia CU Eliminar Nodo

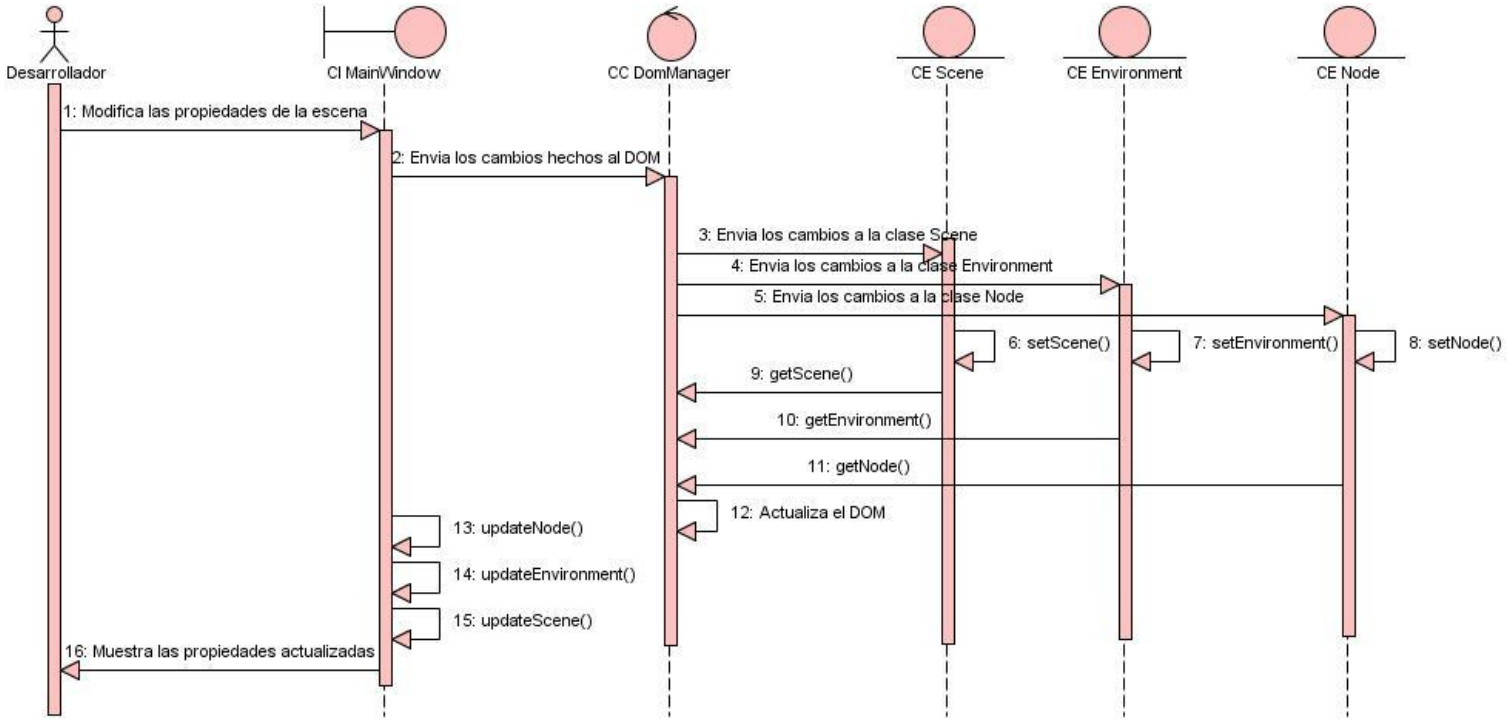


Figura 16 Diagrama de secuencia CU Modificar Escena

3.4- Diagramas de componentes

Un diagrama de componentes representa cómo un sistema de software es dividido en componentes y muestra las dependencias entre estos componentes. Los componentes físicos incluyen archivos, cabeceras, bibliotecas compartidas, módulos, ejecutables, o paquetes.

Para lograr una mejor comprensión de la implementación del módulo se dividió el diagrama de componentes en dos vistas, una vista de código fuente donde se muestran los componentes .h y .cpp y una vista binaria donde se muestran el ejecutable y las bibliotecas utilizadas, así como las relaciones de dependencia entre ellos.

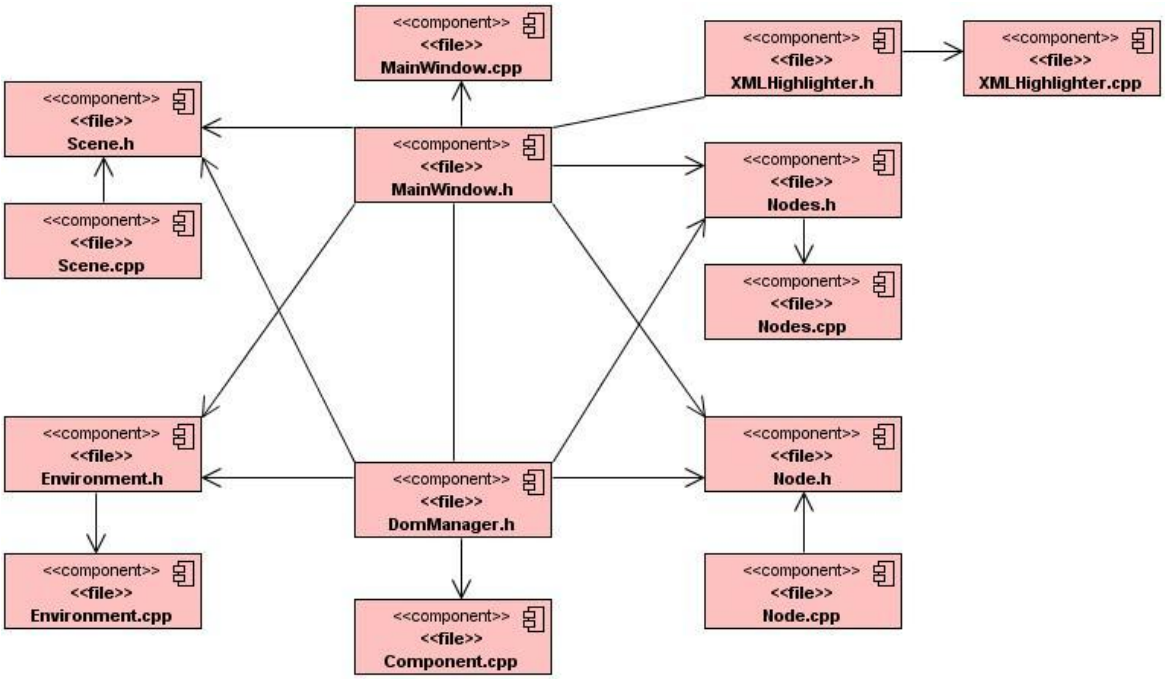


Figura 17 Diagrama de componentes de la vista de código fuente

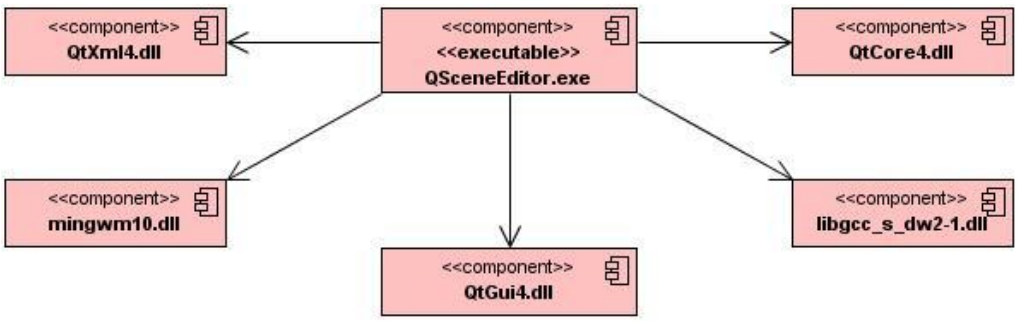


Figura 18 Diagrama de componentes de la vista binaria

3.5- Descripción de los casos de pruebas

A continuación se recogen los casos de prueba utilizados para dar validación a los casos de uso del sistema. El tipo de prueba usado es el de caja negra.

Prueba de caja negra: Se refiere a las pruebas que se llevan a cabo sobre la interfaz del software, por lo que los casos de prueba pretenden demostrar que las funciones del software son operativas, que la entrada se acepta de forma adecuada y que se produce una salida correcta, así como que la integridad de la información externa se mantiene. Este tipo de prueba examina algunos aspectos del modelo, fundamentalmente del sistema, sin tener en cuenta la estructura interna del software.

Tabla 17 Prueba de caja negra al CU Insertar Nodo

Caso de uso:	Insertar nodo.
Caso de prueba:	Inserción de nodos en lugar erróneo.
Entrada:	Se intenta la inserción de un nodo hijo en una etiqueta no permitida.
Resultado:	Muestra un mensaje de error informando las condiciones persistentes y no se inserta el nodo.
Condiciones:	Solo se pueden añadir nodos hijos dentro de la etiqueta nodes u otro nodo.

Tabla 18 Prueba de caja negra al CU Eliminar Nodo

Caso de uso:	Eliminar nodo.
Caso de prueba:	Intento de eliminación de nodo erróneo.
Entrada:	Se intenta eliminar un nodo no permitido.
Resultado:	Se muestra mensaje de error informando las condiciones persistentes y no se procede a eliminar el nodo.
Condiciones:	Solo se pueden eliminar las etiquetas “node”, todas las demás forman parte de la estructura del fichero.

Tabla 19 Prueba de caja negra al CU Modificar Escena

Caso de uso:	Modificar escena.
Caso de prueba:	Introducción de datos erróneos.
Entrada:	Se introducen datos erróneos en las propiedades de las diversas etiquetas de la escena.
Resultado:	El sistema permite validación para la mayoría de los campos pero en algunos casos si permite entradas erróneas.
Condiciones:	Tener conocimiento del valor permisible de algunos campos.

Conclusiones del capítulo

En este capítulo se pudo comprender el funcionamiento del sistema conociendo la estructura de sus clases y la descripción de las mismas, se conocieron además los estándares de codificación para un mejor entendimiento del código del software, así como el patrón arquitectónico utilizado.

CONCLUSIONES

Se desarrolló un módulo con la capacidad de editar los archivos que controlan las características de las escenas dentro de los laboratorios virtuales y luego de su validación mediante pruebas a los casos de uso, se dio cumplimiento a los objetivos generales propuestos en el trabajo.

Dentro de los logros más significativos alcanzados se encuentra el uso de la aplicación con independencia de hardware y sobre múltiples plataformas, además de presentar un diseño flexible posibilitando futuras incorporaciones de funcionalidades que hagan más fácil el manejo de los documentos tratados, cumpliendo de esta manera con las exigencias necesitadas por el proyecto.

Se realizaron estudios sobre varios temas quedando plasmados en el documento para futuras consultas y un mejor entendimiento del trabajo.

RECOMENDACIONES

Este trabajo se realizó con el objetivo de mejorar el desarrollo de los laboratorios virtuales dentro del proyecto PROLAVI. Cabe destacar que su uso puede ser de utilidad para todo el que necesite una correcta edición de archivos. Como futuros trabajos se recomiendan las siguientes mejoras.

- La aplicación maneja solo las propiedades de los archivos que se utilizan en el proyecto actualmente, se recomienda adicionar todas las propiedades que soporta la extensión scene para tener un control total de la estructura de las mismas.
- Incorporar plantillas predefinidas de estructuras de archivos scene más utilizadas.
- Agregar las opciones de deshacer/rehacer para las operaciones en el editor.

REFERENCIAS BIBLIOGRÁFICAS

Avila, Patricia y Martha Diana. 1999. *Virtual Environment for Learning, a new experience.* Düsseldorf, Alemania. : s.n., 1999.

Calderon, Francisco. 2006. *Actitud de los docentes de estudios dirigidos hacia el uso de las tic y su relación con la práctica pedagógica.* España : s.n., 2006.

Enríquez. 2007. slideshare. *slideshare.* [En línea] 2007. http://www.slideshare.net/ovruni/umbrello-uml-modeller.

Escribano, Gerardo Fernández. 2002. *Introducción a Extreme Programming.* 2002.

Herreros, J. R. 2009. *Nuevas aportaciones didácticas de los laboratorios virtuales y remotos en la enseñanza de la Física.* Lisboa, Portugal : s.n., 2009.

Jacobson, Ivar. 2000. *El Proceso Unificado de Desarrollo de Software por Ivar Jacobson, Grady Booch, James Rumbaugh.* Madrid : s.n., 2000.

Jacobson, Ivar. 2000.. *El Proceso Unificado de Desarrollo de Software.* 2000.

José Canós, Patricio Letelier y María Carmen Penadés. 2006. *Metodologías Ágiles en el Desarrollo de Software.* Valencia, España : s.n., 2006.

Kendall. 2006. *Análisis y diseño de sistemas.* Madrid, España : s.n., 2006.

Larman, Craig. 2004.. *UML y Patrones.* 2004.

Marchena, Daniel. 2008. *Entornos Virtuales de Aprendizaje.* Venezuela : s.n., 2008.

Mendoza, María. 2004. HDD. *Informatizate.* [En línea] 2004. http://www.informatizate.net.

Molpeceres, Alberto. 2002. *Procesos de desarrollo.* 2002.

Orallo, Enrique Hernández. 2005. *El lenguaje Unificado de Modelado (UML).* 2005.

Rumbaugh, Ivar. 2000. *El Proceso Unificado de Desarrollo de Software.* Madrid, España : s.n., 2000.

BILIOGRAFÍA

Barcalow, Joseph Yoder y Jeffrey. 1998. Architectural Patterns for Enabling Application Security. 1998.

Enríquez. 2007. slideshare. *slideshare.* [En línea] 2007. http://www.slideshare.net/ovruni/umbrello-uml-modeller.

Escribano, Gerardo Fernández. 2002. *Introducción a Extreme Programming.* 2002.

Isaías Carrillo, Rodrigo Pérez y Aureliano Rodríguez. 2008. *Metodologías del desarrollo del software.* 2008.

Juan Bernardo Quintero, Raquel Anaya y Juan Carlos Marín. 2004. *Estudio comparativo de herramientas para el modelado con UML.* 2004.

Larman, Craig. 2004.. *UML y Patrones.* 2004.

Lovelle, Juan Manuel Cueva. 1999. *Introducción a UML Lenguaje para modelar objetos.* 1999.

Marchena, Daniel. 2008. *Entornos Virtuales de Aprendizaje.* Venezuela : s.n., 2008.

Martin, Robert C. 2000. *Design Principles and Design Patterns.* 2000.

Mendoza, María. 2004. HDD. *Informatizate.* [En línea] 2004. http://www.informatizate.net.

Molpeceres, Alberto. 2002. *Procesos de desarrollo RUP, XP y FDD.* 2002.

Natalia Juristo, Ana M. Moreno y Sira Vegas. 2006. *Técnicas de evaluación de software.* 2006.

Orallo, Enrique Hernández. 2005. *El lenguaje Unificado de Modelado (UML).* 2005.

Ruiz, Francisco Belmonte. 2010. *Programación Extrema (XP).* 2010.

Rumbaugh, Ivar. 2000. *El Proceso Unificado de Desarrollo de Software.* Madrid, España : s.n., 2000.

Systems, Popkin Software and. 2003. *Modelado de sistemas con UML.* 2003.

GLOSARIO DE TÉRMINOS

CSS Las hojas de estilo en cascada (en inglés Cascading Style Sheets), CSS es un lenguaje usado para definir la presentación de un documento estructurado escrito en HTML o XML (y por extensión en XHTML).

DTD Una definición de tipo de documento o DTD (siglas en inglés de *document type definition*) es una descripción de estructura y sintaxis de un documento XML o SGML. Su función básica es la descripción del formato de datos, para usar un formato común y mantener la consistencia entre todos los documentos que utilicen la misma DTD. De esta forma, dichos documentos, pueden ser validados, conocen la estructura de los elementos y la descripción de los datos que trae consigo cada documento, y pueden además compartir la misma descripción y forma de validación dentro de un grupo de trabajo que usa el mismo tipo de información.

HTML siglas de Hyper Text Markup Language (Lenguaje de Marcado de Hipertexto), es el lenguaje de marcado predominante para la elaboración de páginas web. Es usado para describir la estructura y el contenido en forma de texto, así como para complementar el texto con objetos tales como imágenes.

IDE de las siglas en inglés *integrated development environment* (entorno de desarrollo integrado) es un programa informático compuesto por un conjunto de herramientas de programación. Puede dedicarse en exclusiva a un sólo lenguaje de programación o bien, poder utilizarse para varios.

MathML El MathML o *Mathematical Markup Language* es un lenguaje de marcado basado en XML, cuyo objetivo es expresar notación matemática de forma que distintas máquinas puedan entenderla, para su uso en combinación con XHTML en páginas web, y para intercambio de información entre programas de tipo matemático en general.

OGRE (acrónimo del inglés *Object-Oriented Graphics Rendering Engine*) es un motor de renderizado 3D orientado a escenas, escrito en el lenguaje de programación C++.

Quaternion Los cuaterniones (también llamados cuaternios) son una extensión de los números reales, similar a la de los números complejos.

RelaxNG Regular Language for XML Next Generation) es un lenguajes de esquema para XML.

SGML (Lenguaje de Marcas Estándar Generalizado, Standard Generalized Markup Language), especializado en la descripción de documentos en pantalla a través de marcas. El proyecto inicial se basaba en una colección de etiquetas que permitían describir documentos de texto y vínculos de hipertexto que permitían desplazarse entre diferentes documentos, siempre con independencia de la máquina.

SVG Los Gráficos Vectoriales Escalables (del inglés *Scalable Vector Graphics*) o SVG es una especificación para describir gráficos vectoriales bidimensionales, tanto estáticos como animados en formato XML.

UNESCO La Organización de las Naciones Unidas para la Educación, la Ciencia y la Cultura. Se fundó el 16 de noviembre de 1945 con el objetivo de contribuir a la paz y a la seguridad en el mundo mediante la educación, la ciencia, la cultura y las comunicaciones.

XHTML acrónimo en inglés de Extensible Hypertext Markup Language (lenguaje extensible de marcado de hipertexto), es el lenguaje de marcado pensado para sustituir a HTML como estándar para las páginas web.

XPath (*XML Path Language*) es un lenguaje que permite construir expresiones que recorren y procesan un documento XML. La idea es parecida a las expresiones regulares para seleccionar partes de un texto sin atributos.

XQuery es un lenguaje de consulta diseñado para consultar colecciones de datos XML. Es semánticamente similar a SQL, pero incluye algunas capacidades de programación.

XSD siglas de XML Schema Definition es un lenguaje de esquema utilizado para describir la estructura y las restricciones de los contenidos de los documentos XML de una forma muy precisa, más allá de las normas sintácticas impuestas por el propio lenguaje XML. Se consigue así una percepción del tipo de documento con un nivel alto de abstracción. Fue desarrollado por el World Wide Web Consortium (W3C) y alcanzó el nivel de recomendación en mayo de 2001.

XSLT o **Transformaciones XSL** es un estándar de la organización W3C que presenta una forma de transformar documentos XML en otros e incluso a formatos que no son XML.