UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

FACULTAD 5 ENTORNOS VIRTUALES

Herramienta de creación de grafos de caminos para la biblioteca "SceneToolkit"

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

Autores:

Yessy Cedeño González

Ismael Hernandez Gil

Tutor: Ing. Fernando Jiménez López.

Co-Tutor: Ing. Yanoski Camacho Román.

Junio 2008

DECLARACIÓN DE AUTORÍA

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

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

Autores:

_____ _____

Yessy Cedeño González Ismael Hernandez Gil

Tutor:

_____

Ing. Fernando Jiménez López

DATOS DE CONTACTO

Ing. Fernando Jiménez López.

Graduado de Ingeniería Informática en la Ciudad Universitaria José Antonio Echeverría. Actualmente es Profesor Instructor de la Universidad de Ciencias Informáticas. Imparte la asignatura Gráficos por Computadora y se desempeña como Jefe del Polo de Realidad Virtual de la Facultad 5.

E-Mail: fjimenez@uci.cu

Ing. Yanoski Camacho Román

Graduado de Ingeniería Informática en la Ciudad Universitaria José Antonio Echeverría. Actualmente es Profesor Instructor de la Universidad de Ciencias Informáticas. Imparte la asignatura Sistemas Operativos y es Jefe del Proyecto “Herramienta de Desarrollo de Sistemas de Realidad Virtual”, perteneciente al Polo de Realidad Virtual de la Facultad 5.

E-Mail: rcamacho@uci.cu

AGRADECIMIENTOS

Hacemos llegar nuestros más sinceros agradecimientos a todas aquellas personas que de una forma u otra han contribuido a la culminación de este trabajo. Agradecemos a la Revolución y a nuestro Comandante en Jefe Fidel Castro Ruz, por crear la Universidad de las Ciencias Informáticas, la cual nos ha brindado la oportunidad de superarnos y formarnos como futuros profesionales de la Patria. De forma especial agradecemos al Ing. Yanoski Rogelio Camacho Román por su disposición a brindarnos apoyo, y a nuestro tutor el Ing. Fernando Jiménez López, dispuesto en todo momento a atender las dudas y propuestas que fueron surgiendo.

A mi madre, por el cariño, la confianza y el apoyo infinito que me ha brindado a lo largo de todos estos años. A mi hermana que ha estado pendiente de mí en todo momento, por su cariño y comprensión. A mi abuela, quien siempre estuvo muy orgullosa y pendiente de mí. A mi familia en general. A Todos mis amigos de la UCI que compartimos conocimientos, buenos y malos momentos, y que me apoyaron cuando más lo necesitaba, todo eso contribuyó a mi formación profesional.

Ismael

A mi madre, por la preocupación, la dedicación y el amor que me ha brindado durante todo este tiempo. A mi padre, por sus consejos y apoyo. A mi esposa, por su amor. A mi hermana Marilyn, por su preocupación. A toda mi familia. A mis amigos de la UCI, que juntos hemos recorrido todos estos años universitarios. A mis amigos del barrio, que siempre estuvieron al tanto mío.

Yessy

DEDICATORIA

A mi madre, a mi hermana y a toda la familia.

Ismael

A mis padres, a mis hermanas, a Tina.

A toda mi familia.

Yessy

RESUMEN

El avance de la informática en los últimos tiempos ha propiciado el surgimiento y desarrollo de las técnicas y sistemas de “Realidad Virtual”. Iniciada en los programas de entrenamientos militares, simuladores de vuelo y centros de investigación, ha pasado a formar parte de programas muy variados, tanto en el ámbito profesional como doméstico.

La creación y modificación de mapas o grafos de caminos es un tema muy importante para las tecnologías de desarrollo de entornos virtuales, principalmente porque son la base para la inteligencia artificial. De ese modo, la mayoría de estas aplicaciones dispone de alguna herramienta para crear y modificar curvas tridimensionales para utilizarlas en animaciones de elementos. Estas tareas son muy difíciles de realizar “a mano”, ya que involucran ecuaciones paramétricas en un espacio tridimensional. Es por ello que este trabajo aborda el diseño e implementación de una herramienta gráfica para crear y modificar grafos de caminos utilizados en animaciones y pathfinding en entornos virtuales tratados por la biblioteca SceneToolkit.

Para alcanzar esta meta se estudian temas como: planificación de movimientos (*motion planning*), algoritmos de recorrido óptimo sobre grafos, formatos de fichero para almacenar información de grafos haciendo énfasis en aquellos basados en XML, técnicas de aproximación e interpolación, así como las principales características de la biblioteca SceneToolkit y del framework multiplataforma QT. Se exponen además las características técnicas que presentará el sistema como solución a los problemas planteados. Se analiza con profundidad el objeto de estudio, y a partir del mismo se confeccionan los distintos diagramas que posibilitarán un mayor entendimiento y proporcionarán la base para el diseño e implementación de la herramienta.

Palabras clave:

Planificación de movimientos, planificación de movimientos discreta, algoritmos de planificación, realidad virtual, entornos virtuales, curvas paramétricas, Bézier, Spline, B-Spline, NURBS.

ABSTRACT

The advance of information technology in recent times has led to the emergence and development of "Virtual Reality" techniques and systems. Launched in military training programs, flight simulators and research centers, has become part of varied programs, both professional and domestic.

The creation and modification of roadmaps is a very important issue for technology development of virtual environments, mainly because they are the basis for artificial intelligence. Thus, most of these applications have some tool to create and modify curves for use in three-dimensional animation of elements. These tasks are very difficult to carry out "by hand" because it involves parametric equations in a three-dimensional space. That is why this paper addresses the design and implementation of a graphical tool to create and modify roadmaps used in animations and pathfinding in virtual environments processed by the engine SceneToolkit.

To achieve this goal several subjects are studied, such is the case of: motion planning, minimal paths algorithms, file formats for storing graph information emphasizing those based on XML, approximation and interpolation methods, as well as the main features of the engine SceneToolkit and the cross-platform framework QT. It also outlines the technical characteristics that will introduce the system as a solution to the problems raised. It analyzes in depth the subject of study, and from it draws up the various diagrams that allow a greater understanding and provide a basis for the design and implementation of the tool.

Keywords:

Motion planning, discrete motion planning, planning algorithms, virtual reality, virtual environments, parametric curves, Bézier, Spline, B-Spline and NURBS.

TABLA DE CONTENIDOS

INTRODUCIÓN

La Realidad Virtual ha sido considerada fundamentalmente una poderosa herramienta de simulación, y en este sentido se ha pensado la mayor parte de sus aplicaciones potenciales. El aporte que ha realizado a una gran diversidad de actividades ha sido de una magnitud insospechada, es por ello que actualmente sirve de apoyo a una gran cantidad de tareas relacionadas con la ingeniería, la arquitectura, la medicina, la educación y muchas otras áreas. A su vez, importantes avances en la computación han sido debidos a los requerimientos que las tareas gráficas le han exigido.

Las técnicas de planificación de movimientos han logrado éxitos ampliamente generalizados en un gran número de industrias y disciplinas académicas, incluyendo confecciones, diseño de medicamentos y aplicaciones aeroespaciales. Su rápido crecimiento en años recientes indica que muchas más aplicaciones fascinantes están a la vuelta de la esquina. La creación y modificación de mapas o grafos de caminos es un tema muy importante para las tecnologías de desarrollo de entornos virtuales, principalmente porque son la base para la inteligencia artificial. De ese modo, la mayoría de estas aplicaciones dispone de alguna herramienta para crear y modificar curvas tridimensionales utilizadas en animaciones de elementos móviles.

En el Polo de Realidad Virtual de la Facultad 5 de la Universidad de las Ciencias Informáticas, Cuba, existe un proyecto dedicado al desarrollo de un motor gráfico 3D (SceneToolkit) para uso de los demás proyectos productivos. Una de las problemáticas existentes a la hora de configurar los entornos virtuales para STK, es la falta de herramientas gráficas que permitan crear y modificar mapas de caminos, aplicar técnicas de planificación sobre ellos y visualizar los resultados. Estas tareas son muy difíciles de realizar “a mano”, principalmente porque involucran ecuaciones paramétricas en espacios 3D; por lo que en un primer acercamiento y como solución temporal, se optó por utilizar el software propietario 3D Max, que si bien ofrece buenas prestaciones de diseño, no está pensado para ese propósito, viéndose limitadas las opciones del usuario únicamente a la creación de curvas tridimensionales. A todo esto se suma la necesidad de utilizar un script para exportar la información en el formato deseado, la obligación del pago de licencias para poder comercializar los productos finales y el hecho de que no es multiplataforma.

Ante estos inconvenientes surge la siguiente interrogante: *¿Cómo mejorar la creación de mapas de caminos utilizados en entornos virtuales tratados por la biblioteca "SceneToolkit" para contribuir a una mayor soberanía tecnológica?*

Para esto se plantea como **objeto de estudio** el proceso de planificación de movimientos en entornos virtuales y como **campo de acción** el proceso de utilización de las técnicas para la planificación discreta de movimientos con curvas paramétricas de aproximación en entornos virtuales.

Este trabajo se propone como **objetivo**: implementar una herramienta para la creación y modificación de grafos de caminos utilizados por la biblioteca "SceneToolkit" en el manejo de elementos móviles en entornos virtuales.

Se plantean entonces un grupo de tareas que permitirán satisfacer los objetivos, y que se resumen en las siguientes:

- Estudiar los principios básicos, características, conceptos y componentes de la planificación de movimientos.
- Determinar de el(los) algoritmo(s) de recorrido sobre grafos que mejor se ajusta(n) a las necesidades de este trabajo.
- Analizar las principales curvas paramétricas de aproximación e interpolación.
- Seleccionar un formato de fichero para almacenar información referente a grafos.
- Estudiar las características y funcionalidades de la biblioteca SceneToolkit y del framework QT.
- Valorar soluciones técnicas para alcanzar los objetivos propuestos.
- Diseñar una herramienta que solucione los problemas planteados.

Como resultado de este trabajo se pretende construir la herramienta gráfica "GraphBuilder", para la creación y modificación de grafos de caminos --con funcionalidades asociadas tales como deformación de aristas y resolución del camino mínimo entre cada par de nodos-- y el almacenamiento de toda esta información en un fichero que pueda ser utilizado por la biblioteca "SceneToolkit" en el manejo de elementos animados en entornos virtuales.

El contenido de este trabajo se encuentra estructurado de la siguiente manera: en su primer capítulo "Fundamentación Teórica", se hace un análisis bibliográfico donde se investigan las técnicas de

planificación de movimientos. Se analizan algoritmos de recorrido sobre grafos, distintas técnicas de interpolación y aproximación, los principales formatos de fichero para almacenamiento de grafos y se detallan las características principales de la biblioteca SceneToolkit así como del framework multiplataforma QT. En el capítulo 2, “Soluciones Técnicas”, se exponen las características técnicas que presentará el sistema como solución a los problemas planteados. En el capítulo 3, “Características del Sistema”, se crea el modelo del dominio, se hace la captura de requisitos y se crean los modelos de casos de uso del sistema. En el capítulo 4, “Análisis y Diseño del Sistema”, se presentan los diagramas de clases del análisis, los diagramas de clases del diseño y los diagramas de secuencia. En el capítulo 5, “Implementación y Prueba” se muestran los diagramas de despliegue y de componentes. Finalmente, se ofrece un glosario de términos para ayudar a la comprensión del lenguaje técnico utilizado a lo largo del trabajo.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

Introducción

En el presente capítulo se hace una reseña de las principales tecnologías relacionadas con el campo de acción de este trabajo a nivel mundial y nacional, se introducen las técnicas de planificación de movimientos, sus vertientes principales y sus componentes, en especial, de la planificación discreta de movimientos. Se hace un análisis de los principales algoritmos de recorrido sobre grafo. En cuanto al almacenamiento de la información de grafos, se hace una reseña del surgimiento de formatos de ficheros para almacenar información de grafos y se analizan los utilizados en la actualidad con este fin. Por su parte, en lo referente a la interpolación y aproximación, se detallan distintas técnicas. Por último, se detallan las características principales de la biblioteca SceneToolkit, del framework QT y del framework de STK Editor.

1.1 CAGD

Conocido como software de diseño asistido por computador, hace referencia a programas de computadora que asisten a los ingenieros y diseñadores en una amplia variedad de industrias en el diseño y manufactura de productos físicos. [1]

El término CAGD o diseño geométrico asistido por computador fue acuñado por R. Barnhill y R. Riesenfeld en 1974 cuando organizaron una conferencia del tema en la Universidad de Utah, que reunió a investigadores de EE.UU. y Europa, y se considera el evento fundador de este campo. El primer libro de textos “Computational Geometry for Design and Manufacture” por I. Faux y M. Pratt, apareció en 1979. Ya en 1984 es fundada la publicación “Computer Aided Geometric Design” por R. Barnhill y W. Boehm. El primer sistema gráfico interactivo fue creado por I. Sutherland en el MIT en el año 1963, como parte de su tesis de doctorado. [2]

1.2 Tendencias actuales.

En este epígrafe, como resultado de la investigación realizada, se enumeran y caracterizan un conjunto de aplicaciones CAGD y bibliotecas que se relacionan con el campo de acción de este trabajo.

openNURBS: posee herramientas para la transferencia de geometrías 3D entre aplicaciones. Provee la especificación de un formato de fichero, abundante documentación y el código fuente de bibliotecas C++ para manipularlo. Funciona en las plataformas Windows, Windows x64, Mac, y Linux. Cuenta con aseguramiento de la calidad, control de revisiones y varias bibliotecas de soporte y utilidades. Sus herramientas, soporte y membresía son libres. No posee restricciones de licencia ni obliga a los miembros a contribuir. Un detalle esencial es que solamente trabaja con modelos en el formato 3DM y su plataforma principal es Rhinoceros. [3]

Power NURBS: es una herramienta propietaria para la creación y edición de superficies y sólidos integrada a 3D Max, destinada principalmente a artistas 3D. Posee un dibujante de NURBS inteligente para delinear, dibujar o trazar redes de curvas NURBS complejas. Maneja directamente la curvatura y la tangencia, para un mayor control de formas con menos curvas. Incluye una completa y navegable historia por objeto. Ajusta el nivel de detalle para el viewport o el render. Es compatible con ficheros generados por Rhinoceros y Maya. [4]

Existen herramientas profesionales de tratamiento gráfico 3D que incorporan en una u otra medida soporte NURBS, tal es el caso de: 3D Studio, Blender, Maya, Rhinoceros y solidThinking.

Motion Strategy Library: es un entorno de simulación que permite desarrollar y probar algoritmos de planificación de movimientos para una amplia variedad de aplicaciones. Posee una arquitectura orientada a objetos y un diseño modular, basados en la STL de C++. Modela la cinemática y la dinámica de varios sistemas mecánicos, permitiendo la detección de colisiones y la posibilidad de definir obstáculos. Incluye varios modelos de planificadores y permite definir modelos de datos propios. Utiliza OpenGL como motor gráfico. [5]

KineoWorks: es un componente software comercial dedicado a la “planificación automática” de cualquier sistema mecánico o artefacto virtual en entornos 3D, asegurando la evasión de colisiones y

respetando las restricciones cinemáticas. Posee una arquitectura especialmente diseñada para fácil integración dentro de cualquier aplicación. Ofrece buen rendimiento, flexibilidad y fácil configuración. Permite desarrollar aplicaciones personalizadas basadas en configuraciones de cuerpo rígido o sistemas articulados, tales como brazos robóticos. Brinda APIs estándar para planificación de movimientos, el detector de colisiones Kineo, la posibilidad de crear detectores de colisiones por parte del usuario, un optimizador de caminos y código C++ portable. [6]

PathEngine: es un sofisticado middleware comercial con un juego de herramientas para la implementación de movimientos de agentes inteligentes, construido alrededor de la búsqueda de caminos en superficies terrestres tridimensionales basada en puntos de visibilidad. Posee un modelo de movimientos de agente bien definido, con búsqueda de caminos y detección de colisiones, ambos estrechamente relacionados. Este modelo permite búsquedas de caminos extremadamente rápidas en entornos complicados, con soporte para obstáculos situados dinámicamente y teniendo en cuenta la forma del agente. Viene con un poderoso procesador de contenidos (geometrías del mundo), funcionalidades de seguimiento de terrenos, una suite de prueba y herramientas asociadas. El espacio de estados manejado es continuo, permite la generación de caminos curvos y brinda soporte multi-hilos. [7]

VIZMO++: se emplea en el campo educacional para ayudar a los estudiantes en el entendimiento de las diferentes estrategias de planificación mediante la visualización, animación y manipulación de las mismas. Puede ser utilizado para crear o modificar entornos, mediante la adición o eliminación de obstáculos, cambiando la posición u orientación, y ejecutando un planificador para calcular un nuevo mapa de caminos para el nuevo entorno. Los estudiantes también pueden hacer nuevas consultas, salvar entornos para trabajar con ellos más tarde, o seleccionar, mover y/o agregar nuevos nodos, y de ese modo crear nuevos mapas de caminos. Soporta visualización de colisiones entre objetos. Permite la creación de problemas de planificación de movimientos, la comparación de diferentes estrategias de planificación y la visualización de los resultados generados por los diferentes planificadores. [8]

En Cuba:

En el año 2007 el Polo de realidad Virtual de la facultad 5 de la Universidad de las Ciencias Informáticas, presentó un trabajo de diploma titulado “Algoritmos para la manipulación eficiente de objetos dinámicos en escenas virtuales urbanas” de los autores Mariela Nogueira Collazo y Omar

Correa Madrigal. Esta tesis aborda el desarrollo de dos algoritmos que tienen como objetivo emplear eficientemente los recursos de hardware durante la manipulación de objetos dinámicos en entornos virtuales urbanos [9]. También se define una propuesta de diseño de un módulo de clases para incluir los algoritmos a la herramienta SceneToolKit. Como parte de su solución proponen una herramienta de software que permite la transición de un modelo tridimensional de ciudad creado en 3D Studio Max, a un fichero de extensión .grf que contiene información de las posiciones espaciales de todas las trayectorias existentes en un entorno 3D por donde se pueden trasladar los objetos dinámicos. Con esa información entonces podían controlar el movimiento de estos objetos, ya que permitía saber su posición en cada instante. Para realizar el procesamiento de la información contenida en el fichero .grf utilizaron como estructura de datos un grafo, al cual llamaron grafo de trayectorias del entorno.

1.3 Planificación de movimientos (Motion Planning).

Planificación es un término que tiene distintos significados para diferentes grupos de personas. Dentro de la robótica su objetivo es el diseño de algoritmos que generen movimientos útiles tras procesar modelos geométricos complicados. En la inteligencia artificial, su objetivo es el diseño de sistemas que utilicen modelos de decisión teóricos para computar acciones apropiadas, y en la teoría de control el objetivo está relacionado con algoritmos que computen caminos factibles con alguna cobertura adicional de retroalimentación. [10]

1.3.1 Componentes básicos de la Planificación.

Aunque la planificación de movimientos abarca una amplia gama de modelos y problemas, existen varios componentes básicos comunes a todos.

Estado: los tipos de movimientos que un robot puede realizar se referirán como grados de libertad (*dof, del inglés degrees of freedom*). En cada momento en el tiempo, un robot tiene exactamente una posición y una orientación, descritos mediante la asignación de un valor numérico a cada grado de libertad, lo cual constituye un estado. El conjunto de todas las posibles combinaciones numéricas de grados de libertad es llamado espacio de estados. [11]

La definición del espacio de estados es fundamental en la formulación de un problema de planificación y en el diseño y análisis de algoritmos que lo solucionen. Se permiten espacios de estados tanto

discretos (finitos, o contablemente infinitos) y continuos (incontablemente infinitos) [10]. Varias literaturas se refieren a estado con el término “configuración” y a espacio de estados “c” con el término “espacio de configuraciones” (configuration space), no obstante, a lo largo de este trabajo se utilizarán los términos estado y espacio de estados.

Mientras el espacio de estados puede contener un gran número de estados numéricamente posibles, solamente algunos son conceptual o físicamente posibles. Un robot puede moverse entre obstáculos, no a través de ellos. Los estados permitidos se llamarán estados libres y el subconjunto de c compuesto por ellos se llamará espacio de estados libres [11]. De esta manera, los estados no libres serán aquellos en los cuales el robot colisiona con obstáculos o aquellos que no puede alcanzar debido a las restricciones (grados de libertad) a las que está sujeto.

Tiempo: Todos los problemas de planificación involucran una secuencia de decisiones que deben ser aplicadas en el tiempo. Al igual que los espacios de estados, el tiempo puede ser discreto o continuo. [10]

Acciones: Un plan genera acciones que manipulan el estado. En alguna parte de la formulación de la planificación se debe especificar cómo cambia el estado cuando se le aplican acciones. [10]

Estados inicial y final: un problema de planificación generalmente comienza en un estado y trata de arribar al estado final especificado o a algún estado en un conjunto de estados finales. [10]

Criterio: codifica el resultado deseado de un plan en términos del estado y acciones que son ejecutadas. Existen dos consideraciones de planificación basadas en el criterio [10]

- **Factibilidad:** encuentra un plan que permite arribar a un estado meta independientemente de su eficiencia.
- **Optimización:** encuentra un plan factible que optimice el rendimiento de una manera cuidadosamente especificada, además de arribar a un estado meta.

Plan: En general, un plan impone una estrategia específica o comportamiento simplemente especificando una secuencia de acciones a ser ejecutadas. [10]

1.3.2 Enfoques de planificación.

Existen varios enfoques al problema de la planificación de movimientos, la mayoría de los cuales se reducen a las siguientes categorías, aunque no se descartan enfoques híbridos [11]:

- Descomposición por celdas (regiones, rejilla): el espacio de estados es dividido en conjuntos disjuntos (celdas) las cuales son representadas en un grafo. Los nodos se corresponden con celdas y las aristas conectan nodos si las respectivas celdas están conectadas.
- Campos de potenciales: la idea detrás de este enfoque es simple. Cuando un electrón (con carga negativa) es puesto dentro de un campo eléctrico será atraído por una carga positiva (la meta) y repelido por otras cargas negativas (obstáculos).
- Mapas de caminos: este enfoque es similar al primero, sin embargo, los grafos de mapas de caminos emplean nodos para representar estados individuales.

1.3.3 Planificadores basados en mapas de caminos.

Un planificador basado en mapas de caminos construye un grafo tratando de conectar dos estados mediante la búsqueda de una ruta o secuencia de nodos del espacio de estados libres, usados como puntos intermedios. Este enfoque provoca que el planificador se divida en dos fases: primero, el mapa de caminos codificado como un grafo, es construido en una fase de pre-procesamiento para un entorno dado (estático); luego, en una fase de consultas, el grafo es utilizado para resolver una pregunta específica de planificación dados un estado inicial y uno final. Cada vértice en el grafo representa un estado libre y cada arista representa un camino libre de colisiones entre dos estados. A su vez, la fase de pre-procesamiento se divide en dos pasos: construcción del mapa de caminos y refinamiento del mismo. La primera es el núcleo del planificador, mientras la segunda es opcional y trata de mejorar el mapa de caminos haciendo conexiones que el paso anterior omitió. [12]

1.3.4 Planes parciales.

Estos métodos por lo general explotan algunas estructuras que son particulares a la representación de cada problema. Aún más, numerosas heurísticas para acelerar el desempeño han sido desarrolladas a partir de estudios y pruebas, ofreciendo beneficios sustanciales; sin embargo, actualmente no son considerados lo suficientemente “competitivos” en comparación con métodos que buscan en el espacio

de estados, debido a que es muy difícil desarrollar heurísticas específicas de las aplicaciones sin referencias explícitas a estados. [10]

1.4 Planificación discreta.

Un tema central de la planificación de movimientos es la transformación de un modelo continuo a uno discreto, es decir, representar un problema como un grafo, para luego utilizar un algoritmo de búsqueda con el objetivo de encontrar un "camino solución" que conduzca desde el estado inicial hasta el estado meta. Actualmente existen dos alternativas o filosofías [10]:

- *Planificación de movimientos combinatoria*: los algoritmos toman un modelo de entrada y construyen una representación discreta que representa exactamente el problema original.
- *Planificación de movimientos muestral:* los algoritmos emplean la detección de colisiones para tomar muestras del espacio de estados y con ellas conducir búsquedas discretas. Esta filosofía permite el desarrollo de algoritmos independientes de modelos geométricos particulares.

En este tipo de planificación, un grafo es construido como un mapa de caminos, donde existe una clara "línea de observación" (sin obstáculos) entre los extremos de cada arista, de manera que un robot al recorrer el grafo no tropezará con ningún objeto. Una vez construido tal mapa, todo lo que se necesita hacer para mover un robot desde un lugar hacia otro es encontrar los puntos del grafo adyacentes a los puntos inicial y final, conectarlos al grafo y ejecutar una búsqueda. [11]

El mundo es transformado a través de la aplicación de un conjunto de acciones escogidas por un planificador. Cada acción, *u*, cuando es aplicada a partir del estado actual, x, produce un nuevo estado, *x′*, como se especifique por una función de transición de estados, **x′ = f(x, u)**. Como parte del problema de planificación un conjunto X_F de estados finales es definido. La tarea de un algoritmo de planificación es encontrar una secuencia finita de acciones que cuando aplicadas, transforme el estado inicial x_I a algún estado en X_F**.** A menudo es conveniente expresar la información anterior como un *grafo de transición de estados dirigido*. El conjunto de vértices es el espacio de estados. Una arista dirigida a partir de x hasta x′ existe en el grafo si y sólo si existe una acción u tal que x′ = f(x, u). [10]

1.4.1 Búsqueda de planes factibles.

La visión general de este tópico es el uso de algoritmos de búsqueda sobre grafos desde una perspectiva de planificación. Un importante requerimiento es que deben ser sistemáticos, es decir, el algoritmo visitará cada estado alcanzable, permitiéndole declarar en un tiempo finito si existe o no una solución. Para ello debe llevar la cuenta de los estados visitados, sino corre el peligro de caer en ciclos infinitos [10]. Existe un gran número de algoritmos que se ajustan a lo expuesto hasta ahora, pero debido a sus características y complejidades sólo se nombrarán los más conocidos, tal es el caso de los algoritmos de Dijkstra, Floyd-Warshall, A*, Best-first, Iterative Deepening, Backward Search y Bidirectional Search.

1.4.2 Búsqueda de planes óptimos.

Con este tipo de búsqueda en lugar de quedar satisfechos con alguna secuencia de acciones que guíen a la configuración final, suponga que deseáramos una solución que optimice algún criterio, tal como tiempo, distancia o energía consumida. Como con casi todos los problemas de optimización existe una decisión simétrica de si definir un criterio de maximización o minimización. Si el "*costo"* es una clase de energía o gasto, parece sensata la minimización. Si el "*costo"* es una clase de remuneración, la maximización es preferida [10]. Para los efectos de este trabajo siempre escogemos el criterio de minimización, ya que sólo nos interesa la dimensión espacial y siempre trataremos de disminuir la longitud de los caminos planificados. En este tema sobresale el caso del algoritmo de Dijkstra, el cual es un algoritmo de búsqueda sistemática que además busca planes óptimos o el camino más corto desde un nodo de un grafo al resto de los mismos.

1.5 Grafos.

La Teoría de Grafos juega un papel importante en la fundamentación matemática de las ciencias de la computación. Los grafos constituyen una herramienta básica para modelar fenómenos discretos y son fundamentales para la comprensión de las estructuras de datos y el análisis de algoritmos. El término grafo proviene del griego (*grafos*: dibujo, imagen). Informalmente, un grafo es un conjunto de objetos llamados vértices o nodos unidos por enlaces llamados aristas o arcos, que permiten representar relaciones entre elementos de un conjunto. Los grafos permiten estudiar las interrelaciones entre unidades que interactúan unas con otras, de ahí la gran variedad de problemas que pueden ser

modelados empleando estos conceptos, por ejemplo, una red de computadoras puede representarse mediante un grafo, en el que los vértices representan terminales y las aristas representan conexiones.

1.5.1 Algoritmos de recorrido óptimo sobre grafos.

En este tópico se hace referencia al problema que se presenta en muchas aplicaciones de los grafos, donde es necesario encontrar una solución óptima para recorrer los diferentes vértices que lo componen, teniendo en cuenta que el camino resultante debe ser el de menor costo. Para ello, cada arista debe tener un peso asociado, que puede interpretarse como el costo de trasladarse de un vértice a otro, por lo que el problema es modelado mediante un grafo dirigido y ponderado. Cuando se introdujeron los temas *Búsqueda de planes factibles* y *Búsqueda de planes óptimos* se hizo referencia a algunos algoritmos que permitían obtener soluciones factibles y óptimas. A continuación se describen los más importantes desde el punto de vista de esta investigación.

1.5.1.1 Algoritmo de Dijkstra.

Este algoritmo determina el costo del camino más corto desde un vértice origen a todos los demás, donde la longitud de un camino es la suma de los costos de los arcos que lo componen. Es una técnica ávida que opera a partir de un conjunto S de vértices cuya distancia más corta desde el origen ya es conocida. A continuación se muestra el seudo-código del algoritmo:

```
(1)  S = {1};
(2)  For i = 2 to n
(3)       D[i] = C [1, i]; (asigna valor inicial a D)
(4)  For i =1 to n-1
(5)       elige un vértice w en V-S tal que D[w] sea un mínimo
(6)       agrega w a S
(7)       For each vértice v en V-S
(8)            D[v] = min (D[v], D[w] +C [w.v])
```

En principio, S solamente contiene el vértice de origen. En cada paso, se agrega algún vértice restante v a S, cuya distancia desde el origen es la más corta posible. Suponiendo que todos los arcos tienen costo no negativo, siempre es posible encontrar un camino más corto entre el origen y v que pasa sólo

a través de los vértices de S, y que se llama *especial*. En cada paso del algoritmo, se utiliza un arreglo D para registrar la longitud del camino *especial* más corto a cada vértice. Una vez que S incluye todos los vértices, todos los caminos son *especiales,* así que D contendrá la distancia más corta del origen a cada vértice [13].

En el algoritmo se supone que existe un grafo dirigido G= (V, A) en el que V= {1,2,..., n} y el vértice 1 es el origen. C es un arreglo bidimensional de costos, donde C [i, j] es el costo de ir del vértice i al vértice j por el arco i→j, se supone que C [i, j] es ∞, un valor mucho mayor que cualquier costo real. En cada paso D[i] contiene la longitud del camino especial más corto actual para el vértice i. [13]

Para reconstruir el camino más corto del origen a cada vértice, se agrega otro arreglo P de vértices, tal que P[v] contenga el vértice inmediato anterior a v en el camino más corto. [13]

1.5.1.2 Algoritmo de Floyd-Warshall.

Supóngase que se tiene un grafo dirigido ponderado que da el tiempo de vuelo entre ciertas ciudades, y se desea construir una tabla que brinde el menor tiempo requerido para volar entre dos ciudades cualesquiera. Este es un ejemplo de problema de los caminos más cortos entre todos los pares. Para plantear el problema con precisión, se emplea un grafo dirigido G= (V, A) en el cual cada arco v→w tiene un costo no negativo C [v, w]. El problema está en encontrar el camino de longitud más corta entre v y w para cada par ordenado de vértices (v, w).

Podría resolverse este problema por medio del algoritmo de Dijkstra, tomando por turno cada vértice como origen, pero una forma más directa de solución es mediante el algoritmo creado por R.W. Floyd. Por conveniencia, se supone otra matriz A de n×n en la que se calculan las longitudes de los caminos más cortos [13]. A continuación se muestra el seudo-código del algoritmo:

```
Floyd (A: array [1...n, 1...n] of real; C: array [1...n, 1...n] of real)
(Floyd calcula la matriz A de caminos más cortos dada la matriz de costos de arcos C)
        i, j, k: integer
(1)     For i = 1 to n
(2)         For j = 1 to n
(3)             A [i, j] = C [i, j]
```

```
(4)   For i = 1 to n
(5)       A [i, i] = 0
(6)   For k =1 to n
(7)       For i =1 to n
(8)           For j =1 to n
(9)               If (A [i, k] + A [k, j]) < A [i, j])
(10)                  A [i, j] =A [i, k]+A [k, j]
```

Inicialmente se hace A [i, j] = C [i, j] para toda i ≠ j. Si no existe un arco que vaya de i a j, se supone que C [i, j]= ∞. Cada elemento de la diagonal se hace 0. Después se hacen n iteraciones en la matriz A. Al final de la k-ésima iteración, A [i, j] tendrá por valor la longitud mas pequeña de cualquier camino que vaya desde el vértice i hasta el vértice j y que no pase por un vértice con número mayor que k. Esto es, i y j, los vértices extremos del camino, pueden ser cualquier vértice, pero todo vértice intermedio debe ser menor o igual que k. En la k-ésima iteración se aplica la siguiente formula para calcular A: A[i,j] = min (A[i,j] , A[i,k] + A[k,j]). [13]

1.5.1.3 Algoritmo "Primero el mejor" (Best-first).

Best-first es un algoritmo de búsqueda que explora un grafo expandiendo el nodo más prometedor escogido de acuerdo a alguna regla. En [15] se describe este algoritmo como el estimado de la promesa del nodo *n* por una función de evaluación heurística, la cual, en general, puede depender de la descripción de *n*, de la descripción de la meta, de la información acumulada por la búsqueda hasta ese punto, o de cualquier conocimiento extra acerca del dominio del problema. Pudiera pensarse este algoritmo como una búsqueda con heurística que intenta predecir cuán cerca de una solución está el final de un camino, de modo que esos caminos son expandidos primero. A este tipo de búsqueda se le conoce como Greedy Best-first Search. [14]

Las soluciones obtenidas por esta vía no son necesariamente óptimas; por lo tanto, no importa si el estimado excede el verdadero costo óptimo para alcanzar la solución, lo cual es importante para mantener la optimización para la búsqueda A*. Aunque puede que no encuentre una solución óptima, en muchos casos, menos vértices distantes son explorados, lo cual resulta en tiempos de ejecución mucho más rápidos. [10]

La selección eficiente del mejor candidato para expansión es típicamente implementada con una cola con prioridad, ordenada de acuerdo a un estimado del costo para alcanzar la solución [10]. Ejemplos de algoritmos de búsqueda basados en el Best-first incluyen el A*, y a su vez, el Dijkstra (que puede ser considerado como una especialización del A*). Los algoritmos con estas características son usados a menudo para búsquedas de caminos combinatorias.

1.5.1.4 Algoritmo A*.

A* es un algoritmo de búsqueda sobre grafos tipo "Best-first" que busca el camino de menor costo desde un nodo inicial dado a un nodo meta. Utiliza una función heurística distancia-más-costo (usualmente denotada por *f(x)*) para determinar el orden en que la búsqueda visita nodos en el árbol. Dicha heurística es la sumatoria de dos funciones: la función costo-de-camino (usualmente denotada por *g(x)*, que puede o no ser heurística) y un "estimado heurístico" admisible de la distancia a la meta (usualmente denotado por *h(x)*). La función *g(x)* es el costo desde el nodo inicial hasta el nodo actual. [16]

Debido a que la parte *h(x)* de la función *f(x)* debe ser una heurística admisible, la misma no debe sobre-estimar la distancia a la meta. De esta manera para una aplicación como el routing (guiado), *h(x)* pudiera representar la distancia en línea recta a la meta, ya que es físicamente la distancia posible más pequeña entre dos puntos. [16]

A* busca incrementalmente todas las rutas desde el punto inicial hasta que halla el camino más corto a una meta. Como todos los algoritmos de búsqueda informados, busca primeramente las rutas que aparentan ser las que llevarán más rápidamente hasta la meta. Lo que lo diferencia de una Greedy Best-first Search es que también toma en cuenta la distancia ya recorrida (la parte g(x) de la heurística es el costo desde el comienzo, y no simplemente el costo local desde el nodo previamente expandido). [16]

Comenzando con un nodo dado, el algoritmo expande el nodo con el menor valor *f(x)*. A* mantiene un conjunto de soluciones parciales almacenados en una cola con prioridad. La prioridad asignada a un camino *x* es determinada por la función $f(x) = g(x) + h(x)$, mientras menor sea *f(x)* mayor será la prioridad. La función es evaluada hasta que una meta contenga un valor *f(x)* más bajo que cualquier nodo en la cola (o hasta que el árbol haya sido completamente recorrido). [16]

A* es completo en el sentido que siempre que exista una solución, la hallará. Si la función heurística *h* es admisible, significando que nunca sobre-estimará el actual costo mínimo de alcanzar la meta, entonces A* es en sí mismo admisible (o óptimo) si no usamos un conjunto cerrado. Si se utiliza un conjunto cerrado, entonces *h* debe ser también monótona o consistente para que A* sea óptimo. Monótona significa que si existe una conexión desde el nodo A al nodo C, una desde A a B y otra desde B a C, el costo estimado desde A a C siempre será menor o igual que el costo estimado desde A a B sumado al costo estimado desde B a C. [16]

Generalmente hablando, los algoritmos "Primero en profundidad" (Depth-first Search) y "Primero en anchura" (Breath-first Search) son dos casos especiales del A*. El algoritmo de Dijkstra, como otro ejemplo de un algoritmo de búsqueda tipo "Best-first", es el caso especial donde $h(x) = 0$ para todo x.

1.6 Curvas paramétricas de interpolación y aproximación.

Las curvas poseen una representación matemática precisa, bien estudiada y suficientemente flexible. No obstante, y salvo raras excepciones, no es factible, en un sistema de diseño asistido por computador, representar una curva mediante una ecuación, debido fundamentalmente a la necesidad de editar la representación. [17]

A nivel matemático, las curvas se pueden representar como ecuaciones de varias formas, atendiendo a como aparezcan las distintas variables involucradas: ecuaciones explícitas, ecuaciones implícitas y ecuaciones paramétricas. La forma más utilizada en gráficos por computadora es la paramétrica, expresada mediante un parámetro, *t*, que toma valores en un intervalo predeterminado, y dos o tres ecuaciones (según se defina en el plano o en el espacio) [17]. En la siguiente figura se puede apreciar cómo a medida que varía el parámetro t, las distintas componentes de la curva van generando sus coordenadas en 3D.

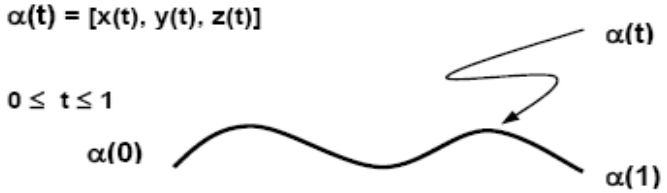


Fig. 1 Curva paramétrica.

Entre las características que distinguen a las diferentes curvas paramétricas desde el punto de vista de su uso para describir el contorno de formas sometidas a diseño, hay algunas que resultan destacables [18]:

- *Continuidad*: varios tramos de curva unidos por sus extremos necesitarán restricciones geométricas para asegurar un orden de continuidad adecuado.
- *Comportamiento local o global de la curva*: para controlar localmente la forma de la curva el desplazamiento de un punto de control sólo debe afectar una porción de curva próxima a él.
- *Suavidad de las curvas*: Algunas formulaciones matemáticas tienden a amplificar las irregularidades en la forma definida por los puntos de control. Otras, por el contrario las suavizan.

1.6.1 Métodos de diseño de curvas.

La mayor parte de los métodos de diseño de curvas se basan en la utilización de puntos de control, a partir de los cuales se definen las curvas como un promedio de los mismos. [17]

Existen diversos métodos de diseño de curvas, con diferentes características. Entre esas características, cabe destacar el *carácter* del método (que puede ser local o global) y el comportamiento respecto a los puntos de control (interpolante o no). Decimos que un método interpola a los puntos de control cuando la curva generada pasa por ellos [17], ver figura 2.

Fig.2 Interpolación.

Fig.3 *Aproximación.*

En las aplicaciones gráficas se utilizan las curvas de aproximación, figura 3, pues las curvas de interpolación necesitan funciones de alto grado (mayor complejidad computacional). [19]

1.6.2 Polinomio de Lagrange.

La interpolación de Lagrange, es sencilla, única y tiene una clara interpretación geométrica: la curva siempre pasará por los puntos. Salvo en casos muy simples, es poco utilizada debido a las oscilaciones del polinomio interpolante conforme aumenta su grado, estrechamente relacionado con el

número de puntos interpolados. Otra característica es el costo computacional: se requieren n operaciones para evaluar un punto sobre una curva. [18]

1.6.3 Curvas cúbicas paramétricas.

Se suelen utilizar las curvas paramétricas de grado 3 porque es el menor que permite crear curvas espaciales y garantizar las continuidades C^0, C^1 y C^2. La continuidad C^2 garantiza que las curvas sean suavizadas. Las de mayor grado necesitan más condiciones para calcularse y tienden a fluctuar, por lo que son más complejas de manipular. [20]

1.6.4 Hermite

La interpolación cúbica de Hermite se ha empleado con cierta frecuencia en el diseño óptimo de forma. Dado que emplea curvas paramétricas, la independencia del sistema de referencia queda garantizada. Suelen utilizarse para la especificación de trayectorias, se definen especificando el punto inicial, el final y sus respectivas tangentes. Poseen control global [18]. En este tipo de interpolación las curvas utilizadas no son invariantes a las transformaciones afines, además son sensibles a cambios en los valores de las derivadas, y por tanto aparecen oscilaciones si se emplea un polinomio de grado elevado. Por ello se eligen esquemas de interpolación cúbica a trozos. [18]

1.6.5 Curvas de Bézier.

Una curva de Bézier consiste en una curva cúbica paramétrica, definida por cuatro puntos, como se muestra en la figura 4, dos finales –situados en los extremos de la misma– y dos de control –que no aparecen sobre la propia curva. Las curvas de Bézier surgen precisamente de un promedio sobre las dos tangentes creadas por los puntos de control, que pueden arrastrarse a voluntad para cambiar libremente su forma hasta obtener prácticamente cualquier tipo de curva posible. [21]

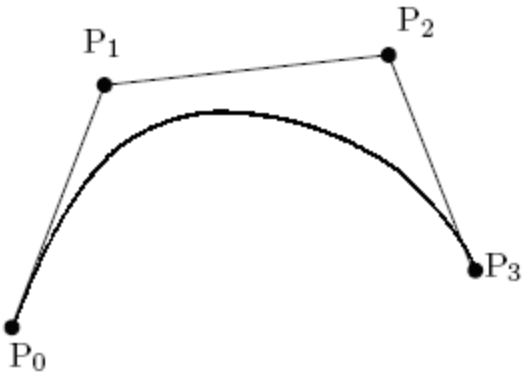


Fig. 4 ***Curva de Bézier cúbica.***

El origen de las curvas de Bézier se remonta a los primeros días del diseño asistido por ordenador a finales de los años sesenta. Aparecen descritas por vez primera en 1972 por Pierre Etienne Bézier, cuando trabajaba como ingeniero para la empresa Renault. Bézier desarrolló este sistema de ecuaciones simples que permite dibujar un número infinito de curvas en una pantalla. Dicho método fue utilizado en aquella época en el diseño de las carrocerías de los automóviles Renault. Las curvas de Bézier tienen varias propiedades que las hacen especialmente útiles y convenientes para representar formas y superficies de objetos en 2D y 3D mediante el ordenador. Estas han llegado a convertirse *de facto* en el fundamento de la práctica totalidad del software de dibujo vectorial más reciente. [21]

Las curvas de Bézier tienen una formulación paramétrica, lo que permite valores múltiples o representar una pendiente infinita (tangente vertical) por dos coordenadas finitas. Son independientes del sistema de referencia, poseen gran flexibilidad para representar una geometría compleja, tienden a suavizar las imperfecciones, pueden unirse fácilmente dos curvas de Bézier (de grado bajo) para formar otra más compleja, son curvas de aproximación y poseen control global. [18]

1.6.6 Curvas Spline.

El término Spline proviene de las tiras de metal flexibles utilizadas por los delineantes y proyectistas para representar superficies de aviones, coches y barcos. Se utilizaban guiándolas mediante puntos de apoyo y otros de donde se tiraba, como se muestra en la figura 5; con esta forma siempre se tenía garantizada una continuidad C^2. [20]

Fig.5 Spline.

El grado de una curva de Bézier determina el número de puntos que contiene el polígono de control. Debido a esto, cuando se necesita aumentar el número de puntos para tener más flexibilidad en la definición de la curva, se debe aumentar el grado. La extensión natural de estas curvas para obtener otras más flexibles, es la de las curvas polinómicas a trozos. Las curvas modeladas de esta manera se conocen como curvas compuestas de Bézier, curvas polinomiales a trozos, o de manera más general curvas Spline. Incorporan los nodos (RIBÓ 2003). Las Splines cúbicas (las más utilizadas) tienen gran flexibilidad, aseguran la continuidad de segundo orden y no necesitan, a priori, el valor de las pendientes en los puntos interpolados (salvo en sus extremos, si es el caso). [18]

Estas curvas poseen control global y para calcular los coeficientes de cada tramo hay que utilizar una matriz que aumenta de orden a medida que aumenta el número de puntos que la definen [20]. En las Spline cúbicas, las discontinuidades en la tercera derivada pueden producir oscilaciones; el desplazamiento de un punto de control obliga a recalcular la curva entera y es necesario especificar a priori los puntos por donde pasa [18].

1.6.7 Curvas B-Spline.

Las curvas B-Spline son curvas Spline representadas y definidas mediante el polígono B-Spline o polígono de Boor y la secuencia de nodos [22]. Son curvas polinómicas que se construyen conectando polinomios de un determinado grado. Los polinomios se obtienen por la combinación de n puntos de control, utilizando una base polinómica. [17]

Poseen muchas de las ventajas de las curvas de Bézier (las cuales generalizan). Permiten las modificaciones locales en la forma de la curva y reducen la necesidad de empalmar muchos trozos de curva para definir la forma. El número de puntos de control es independiente del grado de la curva, permitiendo modelar regiones de elevada curvatura. La posibilidad de elegir arbitrariamente el grado del polinomio unida a la de definir nodos múltiples, potencia la flexibilidad de las curvas B-Spline. [18]

La secuencia de valores de t, que se denominan nodos, define una partición del intervalo de variación del parámetro, permitiendo ajustar la zona de influencia de cada punto de control. Al conjunto de nodos se le llama vector de nodos, el cual es un conjunto de números reales en forma de lista no decreciente, normalmente definido en el intervalo [0, 1]. En función de la distribución de este vector, las B-Spline se clasifican en uniformes y no uniformes. [17]

Si un mismo valor del vector de nodos está repetido *r* veces, se dice que tiene multiplicidad *r*. Diferentes multiplicidades implicarán diferentes propiedades en relación al punto de control que corresponde al nodo. En concreto, una multiplicidad igual al orden supondrá que la curva interpole al punto correspondiente. El primer nodo de la lista y el último tendrán multiplicidad igual al orden para asegurar la interpolación de los puntos extremos. [22]

1.6.8 B-Spline racionales.

Son similares a las B-Spline pero con pesos asociados a cada uno de los puntos de control, permitiendo ajustar más o menos la curva al punto de control. Si todos los pesos son iguales, la expresión de la curva es nuevamente la misma que la del caso no racional. Todos los pesos deben ser positivos ya que de existir uno negativo pueden originarse singularidades. Se debe evitar el empleo de pesos demasiado diferentes entre puntos de control consecutivos pues puede dar lugar a ciertas oscilaciones numéricas en el momento de representar la curva. Los pesos se emplean como parámetros de forma. El aumentar el peso de un determinado punto de control hace que la curva tienda a acercársele más, como si "tirase más" de ella hacia su posición. El efecto de modificar el peso es diferente al de desplazar el punto de control. [23]

1.6.9 NURBS

Acrónimo inglés de la expresión Non Uniform Rational B-Spline. Las NURBS, B-Spline racionales no uniformes, son capaces de describir cualquier forma con precisión, desde simples líneas en 2D, círculos, arcos o curvas, hasta los más complejos sólidos o superficies orgánicas de forma libre en 3D. Gracias a su flexibilidad y precisión, se pueden utilizar modelos NURBS en cualquier proceso, desde la ilustración y animación hasta la fabricación. [18]

Son la representación más general de una forma, se trata de curvas no interpolantes, en general cúbicas, unidas con continuidad C^2 y cuyos puntos de control poseen control local. Pueden pensarse como un método de unión de curvas de Bézier, pero ahorrándonos muchos puntos de control innecesarios. Su diferencia respecto a las B-Spline radica en la definición del vector de nodos. [18]

1.7 Almacenamiento de información de grafos.

Existe una gran cantidad de programas que trabajan con grafos, pero la mayoría de ellos utilizan su propio formato de fichero. Como consecuencia, el intercambio de grafos entre diferentes programas es casi imposible. La idea de un formato de ficheros común nació en el “Graph Drawing 1995“, cuya propuesta, GML, ha sido tomado como referencia y adaptado a una gran cantidad de sistemas de dibujo, sus características claves son: portabilidad, sintaxis simple, extensibilidad y flexibilidad. [24]

1.7.1 XML

XML (e**X**tensible **M**arkup **L**anguage) o Lenguaje de Marcas Extensible en español, es un metalenguaje que permite crear etiquetas adaptadas a las necesidades del usuario (de ahí lo de "extensible"). El estándar define cómo pueden ser esas etiquetas y qué se puede hacer con ellas. Es además especialmente estricto en cuanto a lo que está permitido y lo que no, todo documento debe cumplir dos condiciones: ser *válido* y estar *bien formado*. Fue desarrollado por el World Wide Web Consortium y su objetivo principal era simplificar el SGML para adaptarlo a un campo muy preciso: documentos en Internet, sin embargo, pronto se observó que sus virtudes podían ser útiles en campos bien distintos y es en ese sentido que hoy en día se propone como un estándar para el intercambio de información estructurada entre diferentes plataformas. Su acierto fundamental es que logra un alto equilibrio entre simplicidad y flexibilidad. [25]

1.7.2 Principales formatos de fichero de grafos basados en XML.

A continuación se enumeran y caracterizan los principales formatos de fichero surgidos a partir de XML con el objetivo de almacenar información de grafos.

1.7.2.1 XGMML

Extensible Graph Markup and Modeling Language (XGMML) es una aplicación XML 1.0 basada en GML para describir grafos. Utiliza etiquetas para describir nodos y aristas. Su propósito es el intercambio de grafos entre diferentes herramientas editoras. Puede ser mezclado con otros lenguajes de marcas para describir grafos adicionales, nodos y/o información de aristas. Utiliza todas las etiquetas de GML, adiciona otras y la conversión de grafos de GML a XGMML es trivial. [26]

1.7.2.2 GraphXML

GraphXML es un lenguaje de descripción de grafos basado en XML cuyo objetivo es ser utilizado como formato de intercambio para paquetes de dibujo y visualización de grafos. Aunque GraphXML puede ser usado para la descripción de grafos puramente matemáticos, restringidos al conjunto de nodos y aristas, las aplicaciones de visualización de información requieren más bondades. Por ejemplo, con este formato es posible etiquetar grafos, nodos y aristas. [27]

1.7.2.3 GXL

GXL (Graph eXchange Language) es un formato de intercambio estándar basado en GML, XML y GraphXML para compartir datos entre herramientas. Representa grafos tipeados, con atributos, dirigidos y ordenados que pueden ser extendidos para representar hipergrafos y grafos jerárquicos. Este modelo de datos flexible puede ser utilizado para una gran variedad de grafos. En particular, GXL fue desarrollado para permitir interoperabilidad entre herramientas de reingeniería de software y componentes, tales como extractores de código (parseadores), analizadores y visualizadores. [28]

1.7.2.4 GraphML

El proyecto de GraphML fue iniciado en el “Graph Drawing Steering Committee” previo al “Graph Drawing 2000” de Williamsburg. Un taller sobre formatos de archivos se mantuvo en las vísperas del simposio, y se acordó formar un grupo que definiera un nuevo formato de grafo basado en XML para que fuera estándar en la comunidad de dibujo de grafos y otras comunidades relacionadas. La propuesta de la capa de estructura fue presentada en el siguiente simposio “Graph Drawing Symposium” de Viena. El principal predecesor de GraphML es GML, que fue el resultado de una

iniciativa que comenzó en el “Graph Drawing 1995” en Passau y finalizó en el “Graph Drawing 1996” de Berkeley. GML es todavía uno de los principales formato de grafos soportado por muchos sistemas de dibujo. GraphML es también muy similar al formato anterior GraphXML (1999), aunque presenta mayor difusión y actividad en la actualidad. [29]

El objetivo final de diseño se resume en la siguiente frase: “El formato de intercambio de grafos debe poder representar grafos arbitrarios con información adicional arbitraria, incluyendo información de disposición espacial, diseño, esquema y visualización. La información adicional debe poder ser guardada en un formato apropiado para la aplicación específica, pero no debe complicar o interferir con la representación de datos para otras.¨ [29]

GraphML es un formato comprensible y fácil de usar para especificar grafos. Consta de un mecanismo flexible de extensiones para agregar información específica de cada aplicación, las cuales pueden ser libremente combinadas o ignoradas sin afectar a los datos del grafo en sí mismo. Incluye soporte para grafos dirigidos, no dirigidos y mixtos; hipergrafos, grafos jerárquicos, representaciones gráficas, referencias a datos externos; datos de atributos específicos de aplicaciones y parseadores de peso ligero. [29]

1.8 SceneToolkit

SceneToolkit es un engine gráfico actualmente en desarrollo por el Proyecto Herramientas de Desarrollo para Sistemas de Realidad Virtual de la Facultad 5 de la Universidad de Ciencias Informáticas, Cuba. Utiliza las bibliotecas gráficas OpenGL y DirectX sobre las plataformas Windows y Linux. Posee su propio formato de fichero para almacenar la información de los entornos virtuales, “3dx”, aunque se encuentra en desarrollo su sucesor, el “stk”. Posee módulos de comunicación por redes, audio, física, seguimiento de terreno, shaders e interfaz gráfica de usuario. En la actualidad se encuentra en desarrollo un paquete de herramientas visuales que permitirán ahorrar tiempo y esfuerzo a la hora de configurar sus entornos y elementos. Su arquitectura se divide en tres capas: ***Engine*** donde se encuentra el procesamiento importante de carga y manejo de objetos; ***Renderer*** donde se define con qué biblioteca gráfica se van a dibujar los entornos; y una capa ***Application*** donde se hace el manejo de eventos del sistema operativo específico.

1.9 Qt: Framework de desarrollo multiplataforma.

Qt es un framework de desarrollo multiplataforma con una biblioteca de más de 400 clases, las cuales encapsulan toda la infraestructura necesaria para desarrollar aplicaciones robustas. Posee una utilidad, (QT Designer), para la rápida construcción de capas de interfaz gráfica de usuario con apariencia y aspecto nativos de las plataformas soportadas. Facilita el flujo de trabajo de internacionalización mediante QT Linguist. Está separado en 13 módulos y la API que brinda incluye funciones para conectividad a bases de datos, programación en red, integración con gráficos 3D, desarrollo multi-hilos y lectura/escritura de ficheros XML [30]

Introduce una alternativa innovadora para comunicación entre objetos, los llamados *signals* y *slots*, que reemplazan la antigua e insegura técnica callback. Además provee un modelo de eventos convencional para manipular clics del mouse, teclas presionadas y demás entradas del usuario. [30]

1.10 STK Editor.

STK Editor es una propuesta presentada en junio de 2008 por los ingenieros Yasmany Cubela Medina y Leonardo Nieblas Palau como parte de su tesis de diploma. Dentro de sus objetivos se propusieron: “crear una arquitectura flexible que soporte la adición de plugins” [31]. Esta arquitectura es un prototipo de prueba que será utilizado en el futuro diseño de un entorno de diseño integrado basado en STK y Qt, el cual brindará la opción de agregar extensiones y plugins; para de esa forma empotrar gran variedad de herramientas, tanto de configuración de los distintos aspectos y parámetros de los entornos virtuales de STK, como de uso general para proyectos de investigación académica y actividades docentes. El mayor acierto de la misma es que brinda al usuario una interfaz lo más abstracta posible para implementar el comportamiento de los distintos módulos que interactúan en el núcleo de SceneToolkit, logrando así utilizar las funcionalidades de la Herramienta gráfica de forma más fácil y adicionando soporte para interfaces visuales basadas en Qt.

Conclusiones

En el transcurso de este capítulo, como base para el entendimiento del tema en que se desenvolverá este proyecto, se introdujo al lector en la planificación de movimientos, se hizo un análisis de algunos algoritmos de recorrido sobre grafos, se detallaron las principales ecuaciones paramétricas de aproximación e interpolación, se presentaron diversos formatos de fichero para almacenar información de grafos y se logró entender algunos aspectos (los necesarios para este trabajo) del funcionamiento y estructura de la biblioteca SceneToolkit, del framework Qt y de la arquitectura de plugins de STK Editor.

CAPÍTULO 2: SOLUCIONES TÉCNICAS

Introducción

En este capítulo se hará una descripción de la propuesta a defender, en vista a solucionar la situación actual en el campo de acción. Se proponen soluciones técnicas para el funcionamiento de la herramienta de diseño de caminos, y soluciones específicas para lograr la representación de la información de los grafos, el almacenamiento de la misma en ficheros y la aplicación de técnicas de planificación.

2.1 Transformación del espacio de estados continuo a discreto.

La filosofía de planificación seleccionada para transformar el espacio de estados continuo a discreto es la Planificación de Movimientos Combinatoria. La decisión de su selección como base de la solución propuesta está respaldada por el hecho de que, para los efectos de este trabajo, basta con especificar cada uno de los estados y acciones de manera estática, pues la problemática que se está tratando de resolver no tiene que hacer reconocimiento del terreno o de obstáculos por medio de detección de colisiones ni otra técnica en particular. En otras palabras, las carreteras, aceras, vías ferroviarias, corredores aéreos y rutas marítimas siempre serán estáticos. Además, los algoritmos basados en esta filosofía son completos, lo cual significa que de existir una solución ellos la encontrarán, de otro modo reportan la falla.

El enfoque de planificación a utilizar será el basado en mapas de caminos, en el cual cada estado posee un valor numérico para cada grado de libertad.

De ese modo, los componentes de planificación a representar son:

- **Estado:** representará información sobre las propiedades de un nodo, los grados de libertad estarán relacionados con la posición en el mundo 3D, representada por las coordenadas x-y-z, así como el posible desplazamiento respecto al eje central del camino.
- **Espacio de estados:** será representado mediante el conjunto de nodos del grafo.

- **Acciones:** serán representadas mediante las aristas del grafo, es decir, las aristas del grafo especifican cómo cambiar de un nodo a otro.
- **Estados inicial y final:** serán representados mediante un nodo inicial y un nodo final.
- **Criterio:** se hallarán soluciones factibles y óptimas.
- **Plan:** será representado por una secuencia de aristas desde un nodo inicial a un nodo final.

2.2 Representación de la información del espacio de estados.

La representación de la información correspondiente a estados y acciones es un tema de suma importancia en la planificación discreta de movimientos. En este trabajo, siguiendo la línea estructural definida en [9] se optó por un grafo dirigido. Otras características que tendrá el grafo es que entre dos vértices sólo existirán dos aristas, las cuales no podrán tener el mismo sentido, y serán ponderadas.

Los estados, como se explicó anteriormente, se representarán mediante nodos del grafo, los cuales además de los valores numéricos de los grados de libertad, tendrán propiedades gráficas e información sobre lugares de interés en el mundo virtual, a saber, gasolineras, hoteles, objetivos, lugares destacados (arrancada, meta), etc.

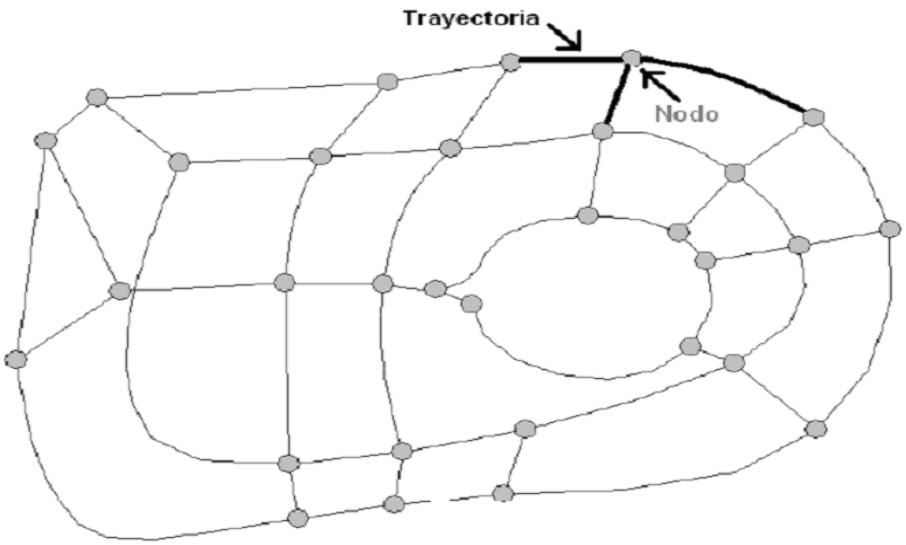


Fig. 6 Ejemplo de grafo de caminos.

En esta figura se aprecia cómo el grafo de caminos del entorno está compuesto por un conjunto de aristas y nodos. Las aristas representan los caminos del entorno y los nodos las intercepciones entre

los caminos o lugares destacados, que constituyen puntos de toma de decisión para los agentes inteligentes. Cuando un agente llegue a un nodo debe decidir qué nuevo camino va a seguir, pues de un nodo pueden partir varias aristas.

La información almacenada en el grafo de caminos del entorno es, a ciencia cierta, un conjunto de vértices 3D que marcan el eje central del camino, por ejemplo: una carretera por donde circulan los autos del entorno tal y como se muestra en la siguiente figura.

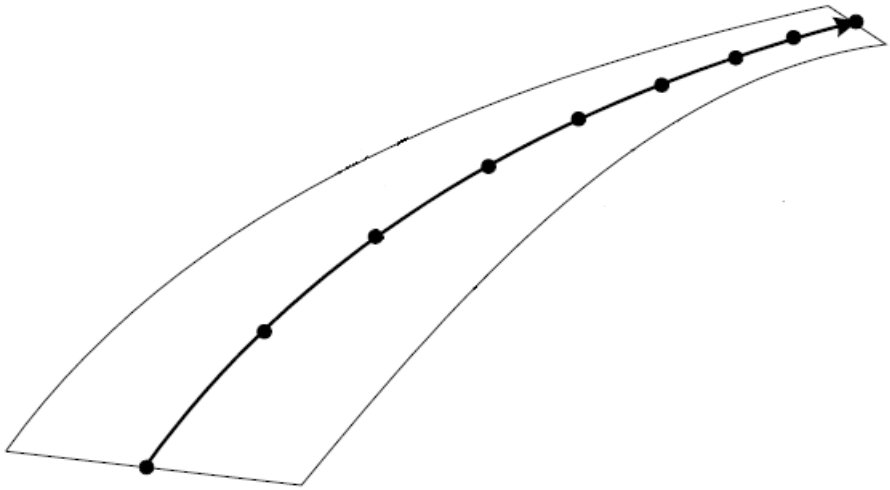


Fig. 7 Eje central de un camino.

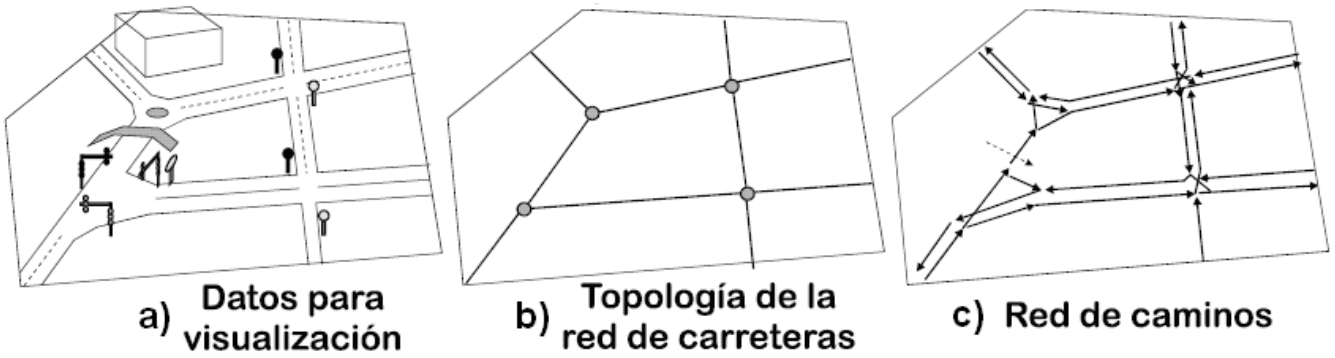


Fig. 8 Distintas vistas de un grafo de caminos del entorno

En la figura 8 se muestran las distintas vistas de un grafo de caminos del entorno. En a) se puede observar el grafo de caminos en el entorno virtual, una vez creado por parte del usuario. En b) se muestra la información topológica del grafo, que junto a la información de navegación mostrada en c), constituyen los únicos datos que necesita el algoritmo de planificación para ejecutarse.

2.3 Modelado de las aristas mediante curvas de aproximación.

Para brindar un mayor realismo a la representación de los grafos de caminos, las aristas serán modeladas mediante NURBS, debido principalmente a las razones expuestas en el capítulo anterior. Estas curvas también ofrecen una formulación matemática común tanto para representar exactamente formas analíticas estándar (por ejemplo: cónicas) como para diseñar con precisión curvas de forma libre. Debido a que todas las entidades, normalmente utilizadas en diseño geométrico, pueden expresarse bajo una única forma matemática, facilitan la integración funcional (todas las entidades tienen el mismo tratamiento: mismos algoritmos) y la física (se simplifica el almacenamiento en una unificada y mejor estructurada base de datos). Su evaluación es razonablemente rápida y computacionalmente estable; los resultados son precisos y consistentes debido al alto nivel de integración. Son invariantes a las transformaciones tales como el escalado, la rotación, la traslación y, en general, a las transformaciones afines y se comportan adecuadamente en las proyecciones paralela y perspectiva. Tienen una clara interpretación física, lo que las hace especialmente útiles para el diseño gráfico asistido por computador. Poseen un poderoso juego de herramientas geométricas: puntos de control (número y posición), elección del grado independiente del número de puntos de control, pesos (ponderación de los puntos de control), vector de nudos (definición, inserción, refinamiento, eliminación, etc.), lo que proporciona gran flexibilidad para diseñar interactiva y cómodamente una amplia gama de formas.

2.4 Algoritmo de planificación.

Como se explicó en el capítulo anterior, el enfoque de planificación basado en mapas de caminos se divide en dos fases: pre-procesamiento (construcción y refinamiento) y consultas. En este punto hay que hacer una aclaración, el objetivo de este trabajo no persigue realizar una planificación dinámica en tiempo de simulación, por lo que se hará una planificación parcial partiendo de que al disponer de una estructura en forma de grafo que almacena información topológica y de navegabilidad del entorno, entonces es muy sencillo desarrollar una heurística basada en la distancia a recorrer para acelerar las búsquedas sobre dicho grafo.

La primera etapa del pre-procesamiento corre a cargo del usuario, quien debe construir el grafo de forma manual. Para la segunda se propone el algoritmo de Floyd-Warshall, para refinar el grafo mediante la obtención de los caminos o secuencias de aristas parciales. La decisión de su selección

fue su capacidad para hallar los caminos más cortos entre cada par de nodos del grafo. Mediante un enfoque sistemático visitará cada estado alcanzable, permitiéndole declarar en un tiempo finito si existe o no una solución. Además establece un valor heurístico para cada nodo (costo mínimo para alcanzar el resto de los nodos), haciendo uso de los criterios factibilidad y optimización. La información obtenida será almacenada en fichero y con ella entonces es posible utilizar un algoritmo de búsqueda informado para hacer consultas y planificaciones dinámicas en tiempo de simulación.

Para hacer las consultas desde un punto de vista estático se seleccionó el algoritmo de búsqueda A*, porque encuentra eficientemente el camino más corto desde un nodo inicial a uno final en el grafo, partiendo de la información del camino recorrido y lo que queda por recorrer, haciendo las búsquedas mucho más rápidas. Además es completo, siempre que exista una solución la hallará en un tiempo finito, sino reportará el fallo. Por todo ello es muy utilizado en muchas situaciones de la programación de juegos.

2.5 Formato de fichero para almacenar información de grafos.

Existe una serie de razones a tener en cuenta al escoger un formato de fichero, una de ellas es que los formatos de intercambio a menudo no soportan todas las características de muchos productos y aspectos específicos de ciertas plataformas. Otro tema es que no podemos esperar que un formato universal sea más eficiente que uno diseñado con propósitos específicos, sobre todo en el caso de los grafos, por lo que a veces muchos desarrolladores toman un formato estándar universalizado y lo adaptan a sus necesidades.

¿Cuáles características son necesarias para un formato de fichero común de grafos? Primero, el formato debe ser independiente de la plataforma y fácil de implementar. Además, debe tener la capacidad de representar estructuras de datos arbitrarias, ya que programas avanzados pueden necesitar adjuntar sus datos específicos a nodos y aristas. Este debe ser lo suficientemente flexible, de manera que el orden específico de las declaraciones no sea significativo, y por tanto esos datos no esenciales puedan ser omitidos. [24]

El formato de fichero a utilizar para almacenar persistentemente la información de los grafos de caminos del entorno será GraphML. En este formato se almacenarán las propiedades de los elementos del grafo así como los caminos existentes en un entorno 3D por donde se pueden trasladar los agentes

inteligentes. Fue seleccionado porque es el más reciente y completo, brinda una serie de bondades que permiten representar más información de la que se necesitaba a los efectos de este trabajo, y por motivos de escalabilidad y adaptabilidad se consideró como el más adecuado.

2.5.1 Estructura del fichero GraphML propuesto.

El formato de fichero GraphML proporciona etiquetas para representar toda la información referente a las estructuras de datos que se tratan en este trabajo, como son los grafos, nodos, aristas y sus atributos. En esta sección se discutirán las partes del documento que son comunes a todos los documentos GraphML y que han sido seleccionadas para ser adecuadas a nuestras necesidades, por lo que se especificarán con ejemplos cada una de las estructuras a utilizar.

2.5.1.1 Definición del grafo.

El grafo será denotado por el elemento *graph*, anidadas dentro de él están las declaraciones de nodos, aristas, vértices de control y caminos. Además, no existirá orden en la definición de nodos, aristas, y caminos, es decir, se podrán definir en el orden deseado sin que esto afecte el grafo resultante.

Ejemplo de definición de un grafo:

```
<graph id="G">
    <node id="n0"/>
    <node id="n1"/>
    ...
    <node id="n10"/>
    <edge source="n0" target="n2"/>
    <edge source="n1" target="n2"/>
    ...
    <edge source="n8" target="n10"/>
</graph>
```

Aunque GraphML soporta tanto grafos dirigidos como no dirigidos en este caso solamente se van a definir grafos dirigidos, cuyas aristas siempre tendrán sentido. Si en la declaración de una arista no se especifica el sentido, se le debe aplicar la dirección *default*. Esta *dirección por defecto* es declarada

como el atributo XML *edgedefault* del elemento *graph* y siempre debe ser especificado con el valor *directed*.

Opcionalmente GraphML permite especificar un identificador para el grafo con el atributo XML *id*, el cual es usado para referenciar dicha estructura dentro del documento. Es necesario destacar que aunque GraphML permite almacenar más de un grafo, a los efectos de este trabajo solamente se almacenará uno, el cual tendrá como id la cadena “GraphBuilder”, que será utilizada para determinar la validez del fichero.

2.5.1.2 Declaración de un nodo.

Los nodos serán declarados con el elemento *node*. Cada nodo tendrá un identificador, el cual deberá ser único en el documento entero. El identificador de un nodo será definido por el atributo *id*.

2.5.1.3 Declaración de una arista.

Las aristas serán declaradas con el elemento *edge*. Cada arista debe definir obligatoriamente sus dos endpoints mediante los atributos *source* y *target*. Los valores de estos atributos deberán ser los identificadores de nodos en el mismo documento.

Las aristas con un solo endpoint, también llamadas bucles, no se deberán definir.

El atributo opcional *directed*, el cual declara si la arista es dirigida o no, de ser definido, deberá tener el valor *true*, el cual indica que la arista será dirigida. Recuerde que si la dirección no es definida explícitamente, se le atribuye la dirección por defecto declarada en el elemento *graph*, que como se explicó anteriormente deberá ser *directed*.

Se debe definir un identificador para cada arista con el atributo *id*, el cual es utilizado para referenciarla.

Una arista con todos sus atributos definidos:

```
...
<edge id="e1" directed="true" source="n0" target="n2"/>
```

2.5.1.4 Declaración de un vértice de control.

Este tipo de estructuras es particular del dominio del problema de este trabajo, y aunque GraphML no incluye etiquetas ni atributos para representarlas, ofrece flexibilidad para agregar los propios.

Los vértices de control serán declarados con el elemento *cv*, y deberán ser anidados dentro del elemento *edge* del cual forman parte, es decir, los vértices de control siempre estarán asociados a una arista del grafo.

Un vértice de control anidado en una arista:

```
...
<edge id="e1" directed="true" source="n0" target="n2">
        <cv id="cv1">
        </cv>
</edge>
...
```

2.5.1.5 Declaración de un camino.

Este tipo de estructuras también es particular del dominio del problema de este trabajo, y por eso se agregaron las etiquetas y atributos necesarios para representarlas.

Los caminos serán declarados con el elemento *path* y pueden definir opcionalmente un identificador con el atributo *id*. Cada camino debe definir obligatoriamente sus dos extremos mediante los atributos *source* y *target*, los cuales representan los identificadores de los nodos inicial y final. Además se debe definir el atributo *successor*, el cual representa el *id* de la arista que parte del *source* en dirección al *target* a lo largo de la secuencia de aristas calculadas con algún algoritmo de planificación parcial. Por último, se debe declarar el atributo *cost*, que representa el costo de ir del *source* al *target*.

Un camino con todos sus atributos definidos:

```
...
<path id="path1" source="n0" target="n1" successor="e2" cost="145.568"/>
...
```

2.5.1.6 Declaración de atributos GraphML.

Un atributo GraphML es definido por medio de un elemento *key*, el cual especifica el identificador, nombre, tipo y dominio del atributo. El identificador es especificado por el atributo XML *id* y es utilizado para hacer referencia al atributo GraphML dentro del documento. El nombre es definido por el atributo XML *attr.name* y deberá ser único de entre todos los atributos GraphML declarados en el documento. El propósito del nombre es que las aplicaciones puedan identificar el significado del atributo. El nombre del atributo GraphML no se utiliza dentro del documento para hacer referencia al elemento, el identificador es quien tiene dicha función. El tipo del atributo GraphML puede ser uno de los siguientes: *boolean*, *int*, *long*, *float*, *double* o *string*. El dominio del atributo GraphML especifica para cuál elemento del grafo dicho atributo GraphML es declarado. Posibles valores incluyen: *graph*, *node*, *edge* y *all*.

Declaración de un atributo GraphML:

```
...
<key id="d1" for="edge" attr.name="weight" attr.type="double"/>
```

Es posible definir valores por defecto para un atributo GraphML. El texto contenido en el elemento *default* define dicho valor por defecto.

Declaración de un atributo GraphML con valor por defecto:

```
...
<key id="d0" for="node" attr.name="color" attr.type="string">
  <default>yellow</default>
</key>
...
```

2.5.1.7 Definición de valores de atributos GraphML.

El valor de un atributo GraphML para un elemento es definido anidándole un elemento *data*. El elemento *data* tiene un atributo XML llamado *key*, el cual hace referencia al identificador del atributo GraphML. El valor del atributo GraphML es el texto contenido en el elemento *data*. Este valor deberá ser del tipo declarado en la correspondiente definición *key*.

Valores de atributo-GraphML.

```
...
<key id="d0" for="node" attr.name="color" attr.type="string">
    <default>yellow</default>
</key>
<key id="d1" for="edge" attr.name="weight" attr.type="double"/>
<graph id="GraphBuilder" edgedefault="directed">
    <node id="n0">
        <data key="d0">green</data>
    </node>
    <node id="n1"/>
    ...
    <edge id="e0" source="n0" target="n1">
        <data key="d1">1.0</data>
    </edge>
    <edge id="e1" source="n1" target="n0">
        <data key="d1">1.0</data>
    </edge>
    ...
</graph>
```

2.6 Interfaz gráfica de usuario.

Para el desarrollo de la interfaz gráfica de usuario se hará uso del framework de STK Editor ya que fue un requisito exigido por el cliente. Entre las características sobresalientes del mismo están la fácil integración de QT y STK, permitiendo la creación de interfaces visuales con el Motor Gráfico embebido. A partir del estudio y utilización de este framework, se hicieron aportes que permitieron mejorar la abstracción de la interfaz para la implementación del comportamiento de los distintos módulos que interactúan en el núcleo de SceneToolkit, logrando así utilizar las funcionalidades de la Herramienta gráfica de forma más fácil. Además brinda una serie de funcionales que permite al programador enfocarse en su solución específica, sin tener que preocuparse por modelar casos de uso generales como cargar mundo, maximizar/minimizar cámaras, activar el modo malla, cambiar de cámaras, entre otros.

2.7 Consideraciones generales.

El diseño e implementación se corresponderá con la filosofía de Programación Orientada a Objetos.

Todo el trabajo de la herramienta se hará a través de funciones programadas puramente en el lenguaje de programación C++ *Standard*.

El fichero de estructura XML que generará la herramienta propuesta podrá ser utilizado en otros programas que soporten el mismo formato, o visualizado e interpretado fácilmente a través de programas destinados a la lectura y tratamiento de XML. No toda la información guardada en este fichero será soportada por otros programas (y viceversa), aunque sí la estructura básica de nodos y aristas.

Conclusiones

Se quiere destacar lo novedoso de varias de las decisiones más importantes tomadas en este proyecto, como son: la posibilidad de portar la herramienta a otros sistemas operativos debido a la utilización de funciones de la STL de C++, de la biblioteca de clases QT y del motor gráfico SceneToolkit, todos multiplataforma. Además se destacan las posibilidades de: crear los caminos del entorno de manera visual e intuitiva, deformar las aristas del grafo, aplicar técnicas de planificación de movimientos desde un punto de vista estático y visualizar los resultados.

El presente capítulo deja sentadas las bases técnicas sobre las cuales será implementada la herramienta visual de diseño de caminos, para dar solución al objetivo propuesto.

CAPÍTULO 3: CARACTERÍSTICAS DEL SISTEMA

Introducción

En este capítulo se comienza a tener una visión más práctica del sistema a desarrollar. Se definen las reglas del negocio y el modelo de dominio. Se obtienen los requisitos funcionales (capacidades o funciones que el sistema debe cumplir) y los no funcionales (propiedades o cualidades que el producto debe tener). Además se describen los procesos que responden a las funcionalidades definidas en los requerimientos funcionales.

3.1 Reglas del negocio.

A continuación se enumeran las reglas del negocio:

1. Una arista sólo podrá relacionar a dos nodos del grafo de caminos.
2. Una arista estará dirigida en el sentido del nodo origen al nodo destino.
3. Entre dos nodos cualesquiera del grafo solamente pueden existir dos aristas, una en sentido del nodo 1 al nodo 2, y la otra en el sentido del nodo 2 al nodo 1.
4. Un camino será representado mediante un número de aristas mayor o igual que 1.
5. Un nodo puede estar representado en la secuencia de varios caminos y aristas, por tanto a la existencia de un nodo con más de una arista asociada se le denominará intersección.
6. La longitud de un camino se determinará mediante la sumatoria de la longitud de cada una de las aristas a lo largo de su secuencia de nodos.
7. Para determinar si existe un camino factible entre dos nodos se debe encontrar una secuencia de nodos cuyas aristas conduzcan hacia el nodo final.
8. Para determinar si existe un camino óptimo entre un par de nodos se debe encontrar una secuencia de nodos cuyas aristas conduzcan hacia el nodo final y la longitud del camino sea la menor de todos los que cumplen con la condición anterior.
9. Los caminos podrán ser de tres tipos: ***terrestrial***, ***aerial*** o ***maritime***. Dentro de los caminos terrestres se hacen tres distinciones: ***car***, ***person***, ***train***.
10. La cantidad de caminos de un grafo y el tipo de cada uno, deberán ser los especificados por el Jefe de Proyecto o diseñador a cargo.

11. La cantidad de nodos y aristas de cada camino deberá estar en directa correspondencia con el nivel de detalle de los caminos y la complejidad que se requieran.

3.2 Modelo del dominio.

En la siguiente figura se muestra el modelo del dominio, el cual muestra los conceptos necesarios para la solución que se pretende construir. Los conceptos se explican en el siguiente epígrafe.

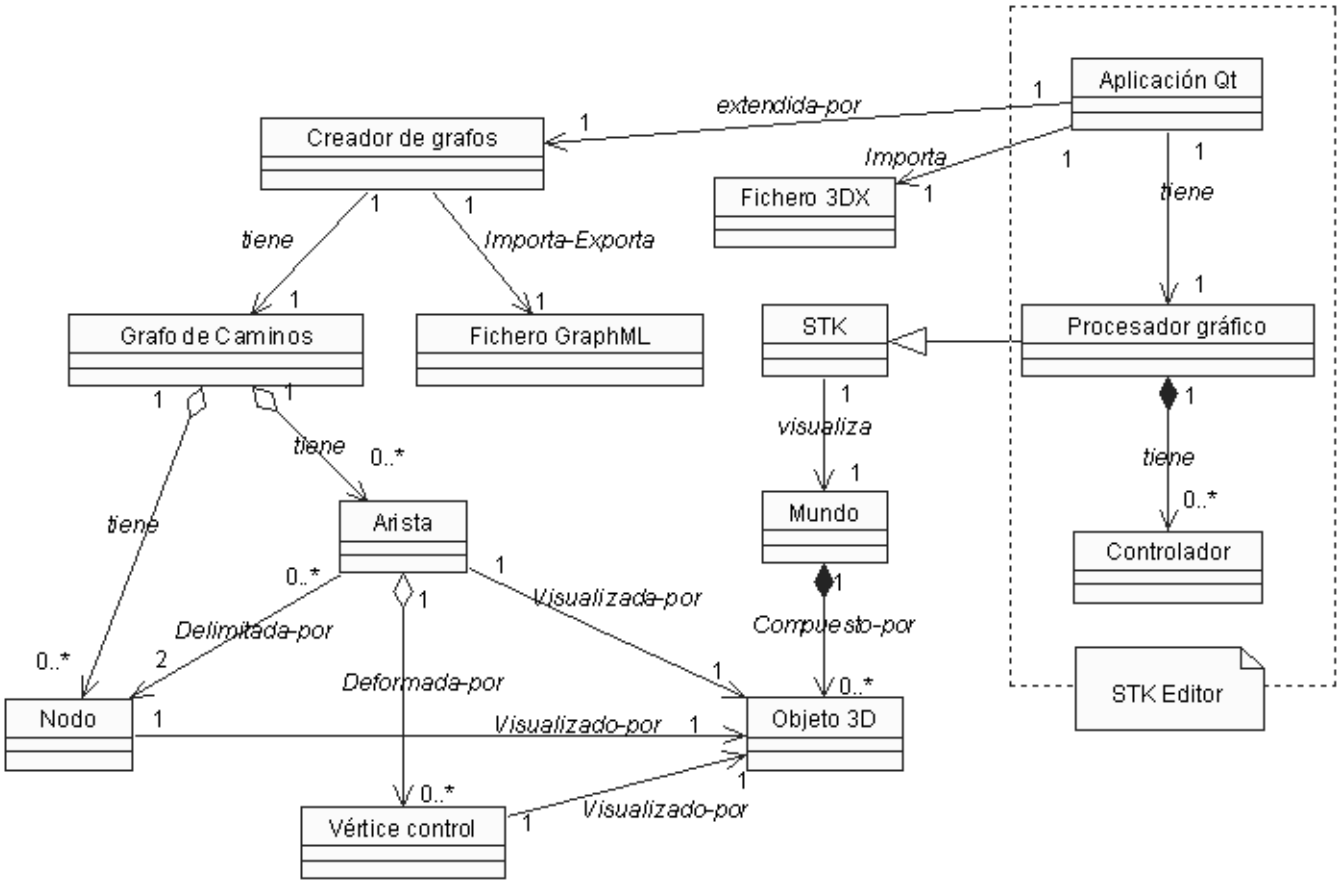


Fig. 9 Modelo del dominio.

Nótese en la figura 9 cómo la Aplicación Qt, el Procesador gráfico y los controladores del mismo constituyen el framework de STK Editor, el cual vincula a STK con Qt. Aplicación Qt brinda una interfaz tanto gráfica de usuario como de programación, que brinda las mismas funcionalidades de STK pero a un nivel de abstracción mucho más alto, incluyendo las de importar ficheros 3DX. Por su parte, el Creador de grafos es la extensión que permite a Aplicación Qt la manipulación de los grafos de caminos y salvar/cargar la información de estos mediante ficheros GraphML.

3.3 Glosario de términos del negocio.

- **Aplicación Qt:** interfaz gráfica de usuario basada en Qt que brinda todas las funcionalidades de STK de forma visual e intuitiva. Define interfaces para que se le puedan acoplar extensiones.
- **Arista:** son arcos o uniones creados entre un par de vértices o nodos, las cuales tendrán una dirección indicando la navegabilidad entre los nodos. Poseen un conjunto de vértices de control.
- **Camino:** hace referencia a un camino o trayectoria, que está integrado por un conjunto de aristas a lo largo de nodos predefinidos por el diseñador y que servirá para trasladar los agentes de una parte a otra en el mundo.
- **Controlador:** fragmento de funcionalidad de STK implementado por el usuario (programador de extensiones) para lograr un comportamiento personalizado de las funcionalidades del motor gráfico.
- **Creador de grafos:** responsable de la gestión de los elementos del grafo de caminos.
- **Fichero GraphML:** fichero con la información referente al grafo de caminos.
- **Fichero 3DX:** fichero con la información referente a un mundo 3D.
- **Grafo de caminos:** es una estructura integrada por un conjunto de caminos, así como los arcos y nodos relacionados con los mismos.
- **Mundo:** mundo 3D, entorno virtual, contiene la información gráfica de varios objetos 3D.
- **Nodo:** estructuras que sirven de guías a la hora de crear caminos en el entorno 3D, son vértices o posiciones específicas del entorno que describen la secuencia de los caminos.
- **Objeto 3D:** representación gráfica tridimensional computarizada de un objeto del mundo real.
- **Procesador gráfico:** es la instancia de STK que servirá de base a la arquitectura de STK Editor, especializa a su ancestro mediante la incorporación de los controladores.
- **STK:** motor gráfico para la visualización de objetos 3D.
- **Vértice de control:** estructuras que son utilizadas para deformar una arista y suavizar los caminos.

3.4 Captura de requisitos.

Los requisitos funcionales son capacidades o condiciones que el sistema debe cumplir y los requerimientos no funcionales son propiedades o cualidades que el producto debe tener. A continuación se expondrán los requisitos funcionales y los no funcionales del sistema.

3.4.1 Requisitos funcionales.

1 – Importar fichero de mundo 3D.
- 1.1 – Cargar fichero de mundo 3D.
- 1.2 – Crear escena con todos los objetos que se definan en el mundo virtual cargado.

2 – Gestionar grafo de caminos.
- 2.1 – Gestionar nodos del grafo.
  - 2.1.1 – Crear nodo.
  - 2.1.2 – Mostrar propiedades de nodo.
  - 2.1.3 – Modificar propiedades de nodo.
  - 2.1.4 – Desplazar nodo.
  - 2.1.5 – Eliminar nodo.
- 2.2 – Gestionar aristas del grafo.
  - 2.2.1 – Crear arista.
  - 2.2.2 – Mostrar propiedades de arista.
  - 2.2.3 – Modificar propiedades de arista.
  - 2.2.4 – Deformar arista.
  - 2.2.5 – Eliminar arista.
  - 2.2.6 – Gestionar vértices de control de la arista.
    - 2.2.6.1 – Agregar vértice de control.
    - 2.2.6.2 – Modificar propiedades de vértice de control.
    - 2.2.6.3 – Desplazar vértice de control.
    - 2.2.6.4 – Eliminar vértice de control.
    - 2.2.6.5 – Mostrar propiedades de vértice de control.

3 – Gestionar almacenamiento persistente de grafos.
- 3.1 – Cargar fichero de grafo.
- 3.2 – Verificar existencia de fichero GraphML y si tiene extensión válida.
- 3.3 – Verifica la validez de fichero GraphML en el encabezado XML.
- 3.4 – Guardar la información del grafo de caminos en fichero de grafo.

4 – Aplicar algoritmos de planificación de movimientos a grafo de caminos.
- 4.1 – Precalcular el camino más corto entre cada par de nodos del grafo de caminos.
- 4.2 – Devolver el camino más corto desde un nodo del grafo hacia otro.

3.4.2 Requisitos no funcionales.

Software:

- Sistema operativo Linux, distribución Debian, Kernel 2.6 con entornos de escritorio KDE o GNOME.
- Freeglut 2.4.0 o superior.

Hardware:

- PC Pentium 4, 2.4 GHz, 512 MB de memoria RAM.
- Tarjeta aceleradora de gráficos Nvidia de 128 MB o superior.

Diseño e implementación:

- Lenguaje de programación C++.
- Se regirá por la filosofía de Programación Orientada a Objetos.
- Se utilizará la herramienta de desarrollo integrado Code::Blocks para programar.
- Se utilizará la aplicación QT Designer para diseñar las interfaces gráficas basadas en widgets.
- Se utilizará la biblioteca SceneToolkit para el tratamiento de entornos virtuales, la que a su vez hace uso de OpenGL que es una biblioteca gráfica libre de más bajo nivel.
- Se utilizará la biblioteca de clases QT para construir la aplicación visual.
- Se utilizará el framework de STK Editor para la vinculación QT-STK.
- Se hará uso de la STL de C++ para representar las estructuras de datos.

Soporte:

- En una versión inicial deberá ser compatible con la plataforma Linux Debian, pero debe estar preparado para que con rápidas modificaciones pueda migrar a otras, tales como Ubuntu, Nova y Windows.

Rendimiento:

- Como aplicación de tiempo real, debe tener alto grado de velocidad de procesamiento o cálculo, tiempo de respuesta y de recuperación, y disponibilidad. La aplicación resultante deberá ser capaz de dibujar 300 nodos y 500 aristas a una velocidad de al menos 30 FPS.

Usabilidad:

- Los usuarios del sistema serán diseñadores con conocimientos básicos. El producto debe ser concebido para que el usuario piense qué desea hacer y no cómo hacerlo.

Apariencia:

- El sistema debe proporcionar una interfaz gráfica de usuario con ventanas, de manera que permita una fácil interacción.

- Deben proveerse varias ventanas de trabajo: superior, frontal, lateral y perspectiva, donde se van a mostrar los elementos del entorno virtual.
- Se debe proveer un menú con todas las funcionalidades organizadas por categorías.

3.5 Modelo de casos de uso del sistema.

En esta sección se reconocen los posibles actores del sistema a desarrollar y se conciben, a través de la agrupación de los requisitos funcionales anteriormente enumerados, los posibles resultados de valor que le pueda brindar a sus actores, o lo que es lo mismo, los casos de uso del sistema.

3.5.1 Actor del sistema.

Tabla 1 Actor del sistema.

Actores	Justificación
Diseñador	Es quien hará uso y se beneficiará de las funcionalidades que brinda la herramienta, a grosso modo: cargar desde ficheros, crear elementos del grafo, actualizar sus datos y aplicar acciones.

3.5.2 Casos de uso del sistema.

1. Cargar entorno.
2. Salvar grafo.
3. Cargar grafo.
4. Crear nodo.
5. Crear nodo manualmente.
6. Crear nodo con asistente.
7. Crear arista.
8. Crear arista manualmente.
9. Crear arista con asistente.
10. Crear vértice de control.
11. Crear vértice de control manualmente.
12. Mostrar propiedades.
13. Buscar nodo.
14. Modificar nodo.
15. Modificar arista.
16. Modificar vértice de control.
17. Desplazar nodo.
18. Desplazar vértice de control.
19. Eliminar nodo.
20. Eliminar arista.
21. Eliminar arista manualmente.
22. Eliminar vértice de control.
23. Actualizar arista.
24. Calcular caminos óptimos.
25. Consultar camino óptimo.

3.5.3 Paquetes de casos de uso del sistema.

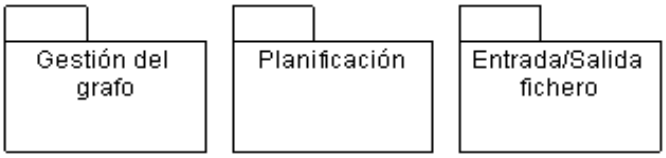


Fig. 10 Paquetes de casos de uso del sistema.

En la figura anterior se representan los 3 paquetes más generales del sistema. Gestión del grafo adiciona, modifica y elimina elementos del grafo. Planificación calcula y consulta caminos mínimos, mientras Entrada/Salida fichero se encarga de leer y escribir la información de forma persistente.

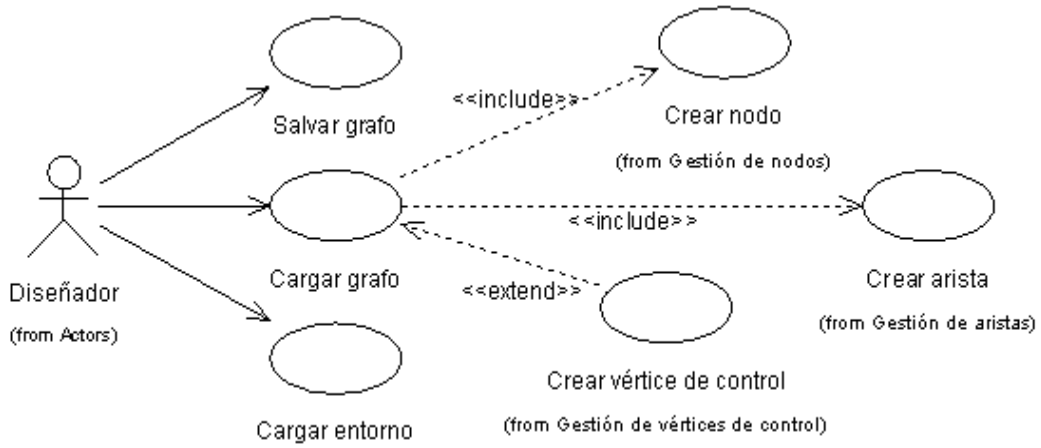


Fig. 11 Diagrama de casos de uso del paquete "Entrada/Salida fichero".

Es necesario destacar en esta imagen que el caso de uso Cargar grafo invocará a Crear nodo, Crear arista y Crear vértice de control (si existen), reutilizando las funcionalidades que brindan los mismos.

Fig. 12 Paquete "Gestión del grafo".

En esta figura se representan los 2 sub-paquetes que integran a Gestión del grafo. Gestión de nodos y Gestión de aristas adicionarán, modificarán y eliminarán nodos y aristas, respectivamente.

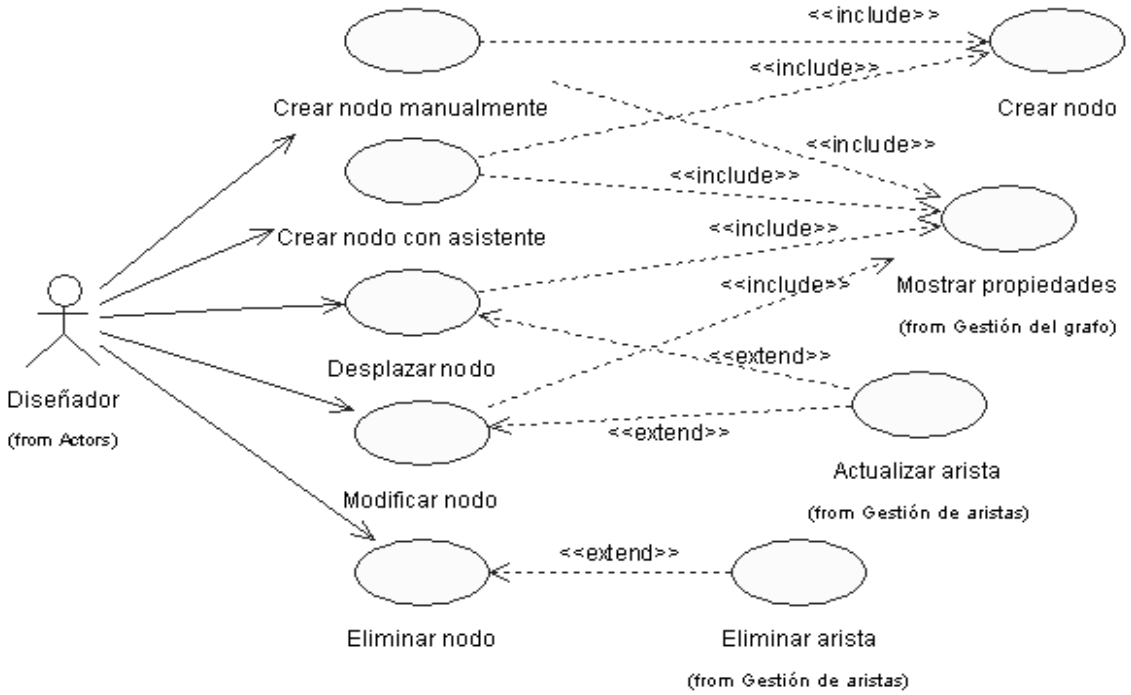


Fig. 13 Diagrama de casos de uso del sub-paquete "Gestión de nodos".

En la fig. 13 el CU Actualizar arista solamente se invocará si un nodo modificado o desplazado tiene alguna arista asociada. Similar ocurre con Eliminar arista.

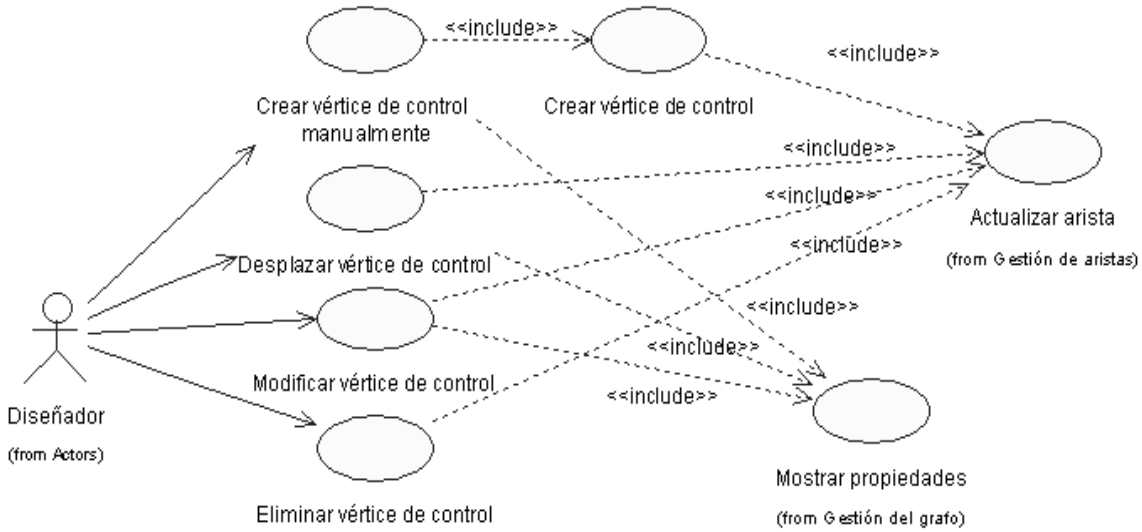


Fig. 14 Diagrama de casos de uso del sub-paquete "Gestión de vértices de control".

En esta imagen se puede apreciar cómo es invocado el CU Actualizar arista cada vez que ocurren cambios en los vértices de control de la misma.

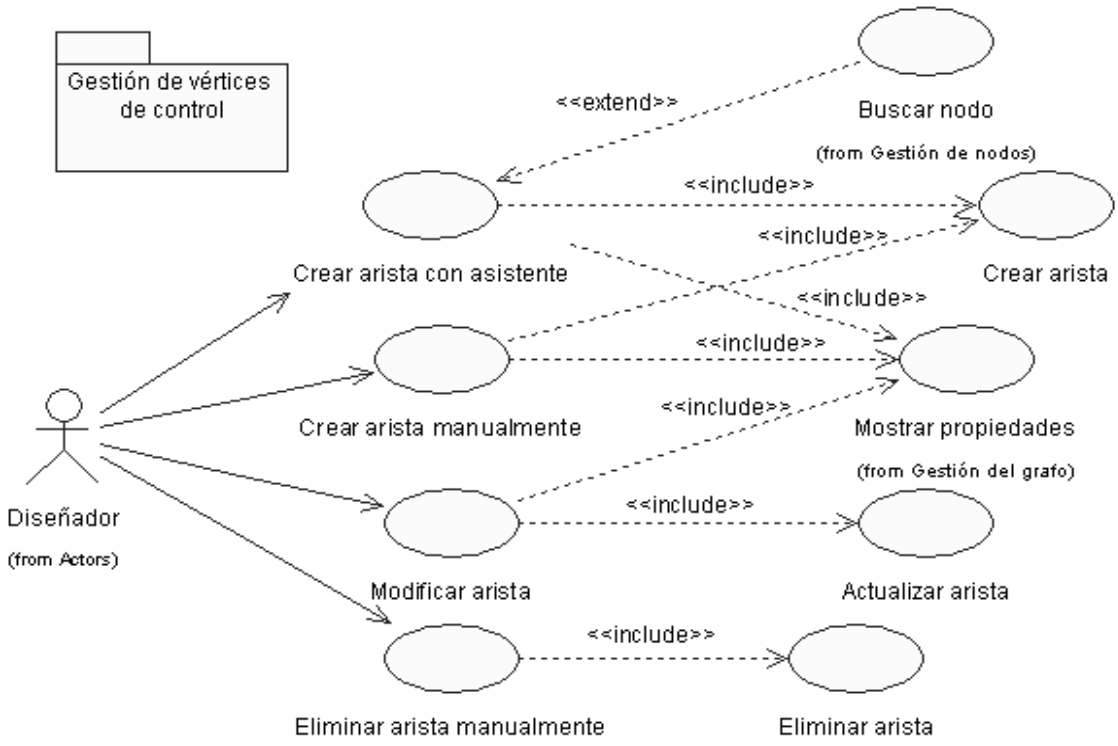


Fig. 15 Diagrama de casos de uso del sub-paquete "Gestión de aristas".

En la fig. 15 el CU Buscar nodo se invocará siempre y cuando el usuario desee seleccionar un nodo del grafo de entre los mostrados en una lista. La otra opción sería especificarlo directamente.

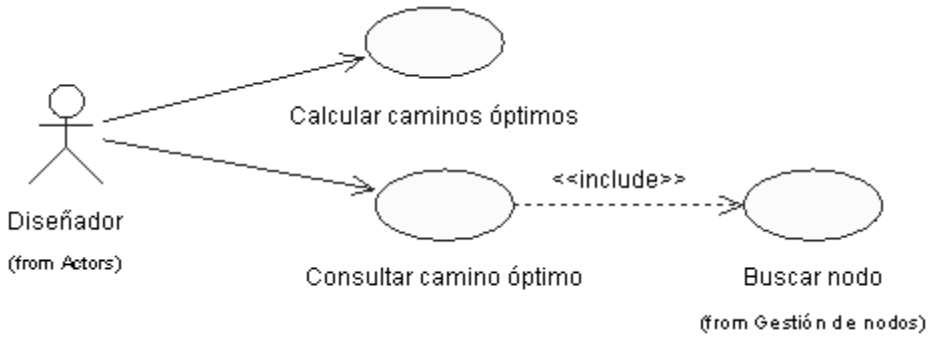


Fig. 16 Diagrama de casos de uso del paquete "Planificación".

El CU Consultar camino óptimo invoca el CU Buscar nodo para seleccionar nodos del grafo de entre los mostrados en una lista.

3.5.4 Especificación de los casos de uso.

Paquete "Entrada/Salida fichero".

Tabla 2 Descripción del caso de uso "Cargar grafo".

<table>
<tr><td>Caso de uso:</td><td colspan="2">Cargar grafo.</td></tr>
<tr><td>Actor(es):</td><td colspan="2">Diseñador (inicia).</td></tr>
<tr><td>Propósito:</td><td colspan="2">Cargar información de grafo de caminos almacenado en fichero.</td></tr>
<tr><td>Resumen:</td><td colspan="2">El caso de uso inicia cuando el usuario del sistema accede a la opción de cargar grafo, especifica el camino y nombre del fichero. A continuación el sistema lee el fichero y si no existe ningún problema se almacena la información en memoria y se muestra al usuario. En caso contrario se notifica al usuario el problema acontecido.</td></tr>
<tr><td>Referencias:</td><td colspan="2">CU3, RF3.1, RF3.2, RF3.3, CU4(include), CU7(include), CU10(extend)</td></tr>
<tr><td colspan="3">Curso Normal de los Eventos</td></tr>
<tr><td colspan="2">Acción del Actor</td><td>Respuesta del Sistema</td></tr>
<tr><td colspan="2">1 – Selecciona la opción "Cargar fichero de grafo".</td><td>1.1 – Solicita el nombre del fichero y el camino donde se encuentra.</td></tr>
<tr><td colspan="2">2 – Especifica los datos solicitados.</td><td>2.1 – Verifica la existencia del fichero y si tiene extensión válida.
2.3 – Se verifica la validez del fichero en el encabezado XML.
2.4 – Por cada estructura de nodo encontrada se leen cada uno de sus atributos y se invoca el caso de uso: Crear nodo.
2.5 – Por cada estructura de arista encontrada se leen cada uno de sus atributos y se invoca el caso de uso: Crear arista.
2.6 – Por cada estructura de vértice de control encontrada se leen cada uno de sus atributos y se invoca el caso de uso: Crear vértice de control.
2.7 – Termina el caso de uso.</td></tr>
<tr><td colspan="3">Cursos Alternos</td></tr>
<tr><td colspan="3">2.1 – Si el fichero no existe o su extensión no es una de las esperadas se informa al usuario.
2.3 – Si el fichero no es válido se detiene la lectura del fichero y se informa al usuario.</td></tr>
<tr><td>Post-condiciones:</td><td colspan="2">Cargada en memoria la estructura del grafo almacenado en fichero.</td></tr>
<tr><td>Prioridad:</td><td colspan="2">Crítico.</td></tr>
</table>

Tabla 3 Descripción del caso de uso "Cargar entorno".

Caso de uso:	**Cargar entorno.**
Actor(es):	Diseñador (inicia).
Propósito:	Cargar información de entorno 3D almacenada en fichero.
Resumen:	El caso de uso inicia cuando el usuario del sistema accede a la opción de cargar entorno, donde deberá especificar el camino donde se encuentra el mismo. Si no existe ningún problema en el proceso de lectura del fichero se almacena el modelo 3D en memoria y se muestra al usuario. En caso contrario se notifica al usuario el problema acontecido.
Referencias:	CU1, RF1.1, RF1.2
Curso Normal de los Eventos	
Acción del Actor	**Respuesta del Sistema**
1 – Selecciona la opción "Cargar entorno".	1.1 – Solicita el nombre del fichero y el camino donde se encuentra.
2 – Especifica los datos solicitados.	2.1 – Se pasan los datos del fichero a la estructura de control de visualización 3D para que lea la información de objetos 3D que contiene el mismo y los muestre en el componente de visualización 3D. 2.2 – Termina el caso de uso.
Cursos Alternos	
2.1 – Si el fichero no existe o su extensión no es válida se informa al usuario.	
Post-condiciones:	Creada la escena del mundo 3D.
Prioridad:	Crítico.

Tabla 4 Descripción del caso de uso "Salvar grafo".

Caso de uso:	**Salvar grafo.**
Actor(es):	Diseñador (inicia).
Propósito:	Salvar información del grafo de caminos en fichero.
Resumen:	El caso de uso inicia cuando el usuario accede a la opción "Salvar grafo", donde deberá especificar el camino donde se guardará el fichero con dicha información. Si no existe ningún problema en el proceso de escritura del fichero, se almacena el grafo de forma persistente. En caso contrario se notifica al usuario el problema acontecido.

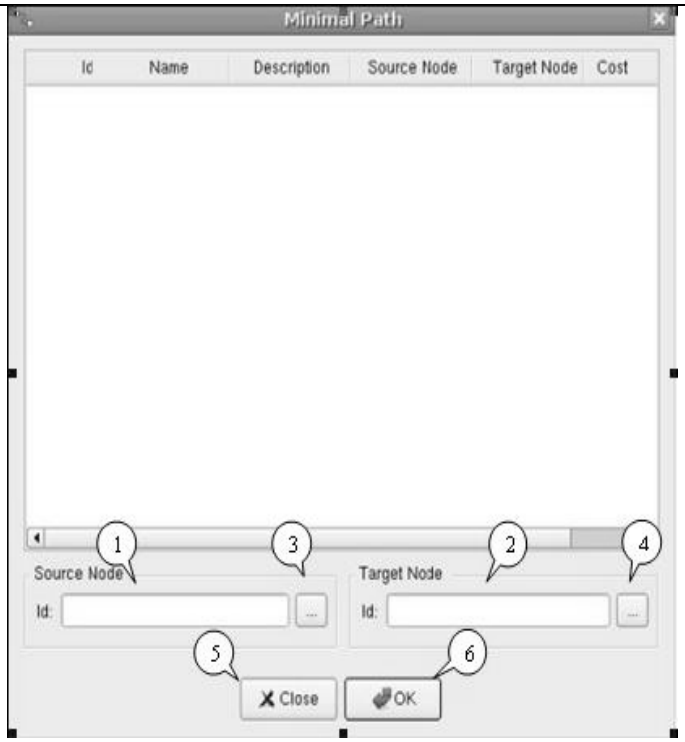
Referencias:	CU2, RF3.4
Curso Normal de los Eventos	
Acción del Actor	**Respuesta del Sistema**
1 – Selecciona la opción “Guardar grafo”.	1.1 – Solicita el nombre del fichero y el camino donde se guardará.
2 – Especifica los datos solicitados.	2.1 – Se guarda toda la información del grafo en el fichero. 2.2 – Termina el caso de uso.
Cursos Alternos	
2.1 – Si existe algún problema en el proceso de almacenamiento se informa al usuario.	
Post-condiciones:	Almacenado el grafo de caminos de manera persistente.
Prioridad:	Crítico.

Paquete “Planificación”.

Tabla 5 Descripción del caso de uso “Calcular caminos óptimos”.

Caso de uso:	**Calcular caminos óptimos.**
Actor(es):	Diseñador (inicia).
Propósito:	Calcular el camino más corto entre cada par de nodos del grafo de caminos.
Resumen:	El caso de uso inicia cuando el usuario del sistema accede a la opción de calcular caminos óptimos. A continuación el sistema efectúa un ciclo de cálculos y determina si existe un camino óptimo entre cada par de nodos y almacena la longitud de camino calculado.
Referencias:	RF4.1, CU24
Curso Normal de los Eventos	
Acción del Actor	**Respuesta del Sistema**
1 – Selecciona la opción “Calcular caminos óptimos”.	1.1 – Por cada par de nodos determina si existen caminos desde el nodo inicial al final y guarda la longitud del más corto. 1.2 – Termina el caso de uso.
Post-condiciones:	Almacenadas en memoria las longitudes de los caminos óptimos.
Prioridad:	Crítico.

Tabla 6 Descripción del caso de uso "Consultar camino óptimo".

Caso de uso:	**Consultar camino óptimo.**	
Actor(es):	Diseñador (inicia).	
Propósito:	Devolver el camino más corto entre dos nodos del grafo de caminos.	
Resumen:	El caso de uso inicia cuando el usuario del sistema accede a la opción de consultar camino más corto entre dos nodos. Especifica los nodos inicial y final, y a continuación el sistema localiza en memoria los datos requeridos y los muestra al usuario.	
Referencias:	CU25, RF4.2, CU13 (include)	
Precondiciones:	Precalculados los caminos más cortos entre cada par de nodos.	
Interfaz I:		(1) Id del nodo origen. (2) Id del nodo destino (3) Buscar Nodo origen (4) Buscar el nodo destino (5) Cancelar (6) Aceptar

Curso Normal de los Eventos	
Acción del Actor	**Respuesta del Sistema**
1 – Selecciona la opción "Consultar camino óptimo" entre dos nodos.	1.1 – Muestra la interfaz I, donde el usuario debe seleccionar los nodos inicial y final.
2 – Selecciona la opción Buscar nodo origen (1).	2.1 – Invoca el caso de uso: **Buscar nodo** **Resultado:** id de nodo (origen) seleccionado
3 – Selecciona la opción Buscar nodo destino (4).	1.2 – Invoca el caso de uso: **Buscar nodo** **Resultado:** id de nodo (destino) seleccionado

4 – Pulsa el botón Aceptar (6).	4.1 – Verifica la validez de los nodos entrados. 4.2 – Localiza en memoria los datos del camino más corto desde el nodo inicial hasta el nodo final. 4.3 – Muestra los datos al usuario. 4.4 – Termina el caso de uso.
Cursos Alternos	
4.1 – Si existe algún problema con alguno de los nodos indicados se detiene la ejecución del caso de uso y se informa al usuario lo ocurrido. 4.2 – Si no existe un camino desde el nodo inicial al final se muestra un mensaje al usuario.	
Prioridad:	Opcional.

Paquete "Gestión grafos".

Tabla 7 Descripción del caso de uso "Mostrar propiedades".

Caso de uso:	**Mostrar propiedades.**
Actor(es):	Diseñador (inicia)
Propósito	Mostrar las propiedades de un elemento.
Precondiciones	Elemento previamente seleccionado.
Resumen:	El caso de uso es invocado externamente habiendo sido previamente seleccionado un elemento. A continuación el sistema muestra sus propiedades.
Referencias:	CU12, RF2.1.2, RF2.2.2, RF2.2.6.5
Curso Normal de los Eventos	
Respuesta del Sistema	
1.1 – Limpia el editor de propiedades. 1.2 – Muestra las propiedades del elemento en el editor de propiedades. 1.3 – Termina el caso de uso.	
Prioridad:	Secundario.

Sub-paquete "Gestión de Nodos".

Tabla 8 Descripción del caso de uso "Crear nodo".

Caso de uso:	**Crear nodo.**
Propósito:	Construir y agregar nodos al grafo de caminos.
Resumen:	El caso de uso es invocado externamente y recibe los datos del nuevo nodo, construye el nodo y lo agrega al grafo de caminos.
Referencias:	CU4, RF2.1.1
Curso Normal de los Eventos	
Respuesta del Sistema	
1.1 – Crea un nodo con los atributos provistos y lo adiciona al grafo de caminos. 1.2 – Actualiza la estructura de control de visualización para agregar la representación gráfica del nodo. 1.3 – Termina el caso de uso.	
Post-condiciones:	Se creó un nodo, se añadió al grafo y se visualizó al usuario.
Prioridad:	Crítico.

Tabla 9 Descripción del caso de uso "Crear nodo manualmente".

Caso de uso:	**Crear nodo manualmente.**
Actor(es):	Diseñador (inicia).
Propósito:	Crear nodos del grafo de caminos.
Resumen:	El caso de uso inicia cuando el usuario del sistema da clic sobre el componente de visualización 3D. A continuación el sistema determina las coordenadas del clic, crea un nodo con propiedades por defecto y lo visualiza en dicha posición.
Referencias:	CU5, CU12 (include), CU4 (include).
Precondiciones	Opción nodo en la barra de herramientas previamente activada.
Curso Normal de los Eventos	
Acción del Actor	**Respuesta del Sistema**
1 – Da clic sobre el componente de visualización 3D, donde desee crear el nodo.	1.1 – Determina las coordenadas donde se creará el nodo. 1.2 – Invoca el caso de uso: **Crear nodo.** 1.3 – Invoca el caso de uso: **Mostrar propiedades.** 1.4 – Termina el caso de uso.
Prioridad:	Crítico.

Tabla 10 Descripción del caso de uso "Crear nodo con asistente".

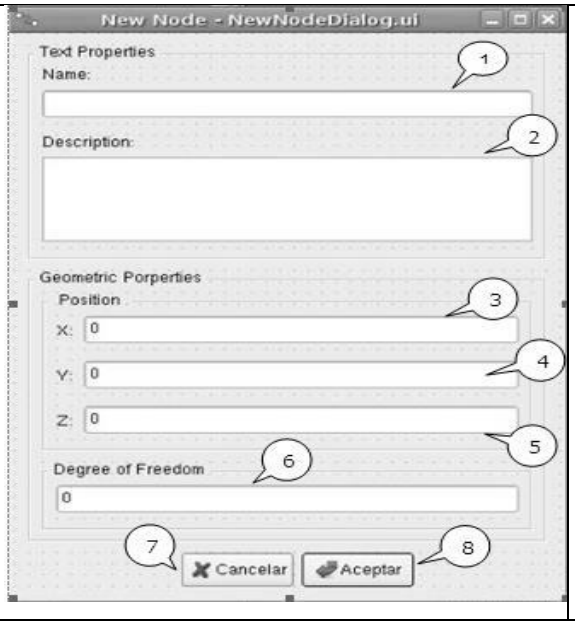
<table>
<tr><td>Caso de uso:</td><td colspan="2">Crear nodo con asistente.</td></tr>
<tr><td>Actor(es):</td><td colspan="2">Diseñador (inicia).</td></tr>
<tr><td>Propósito:</td><td colspan="2">Crear nodos del grafo de caminos.</td></tr>
<tr><td>Resumen:</td><td colspan="2">El caso de uso inicia cuando el usuario del sistema selecciona la opción "Nuevo nodo". A continuación el sistema solicita los datos del nuevo nodo, el usuario los provee y luego el sistema lo crea y lo visualiza.</td></tr>
<tr><td>Referencias:</td><td colspan="2">CU6, CU12 (include), CU4 (include).</td></tr>
<tr><td>Interfaz I</td><td> 
</td><td>1 – Nombre del nodo.
2 – Descripción
3 – Posición en el eje de las x
4 – Posición en el eje de las y
5 – Posición en el eje de las x
6 – Grado de libertad del nodo.
7 – Cancelar
8 – Aceptar</td></tr>
<tr><td colspan="3">Curso Normal de los Eventos</td></tr>
<tr><td>Acción del Actor</td><td colspan="2">Respuesta del Sistema</td></tr>
<tr><td>1 – Selecciona la opción "Nuevo nodo".</td><td colspan="2">1.1 – Muestra la Interfaz I, con valores por defecto para cada una de las propiedades del nodo, permitiendo modificarlos</td></tr>
<tr><td>2 – Modifica los atributos.
3 – Da clic en Aceptar.</td><td colspan="2">3.1 – Invoca el caso de uso: Crear nodo.
3.2 – Invoca el caso de uso: Mostrar propiedades.
3.3 – Termina caso de uso.</td></tr>
<tr><td colspan="3">Cursos Alternos</td></tr>
<tr><td colspan="3">3 – Si selecciona Cancelar, termina el caso de uso.</td></tr>
<tr><td>Prioridad:</td><td colspan="2">Crítico.</td></tr>
</table>

Tabla 11 Descripción del caso de uso "Desplazar nodo".

Caso de uso:	**Desplazar nodo.**	
Actor(es):	Diseñador (inicia).	
Propósito:	Trasladar de posición los nodos del grafo de caminos.	
Resumen:	El caso de uso inicia cuando el usuario del sistema, haciendo uso del teclado, desplaza un nodo previamente seleccionado. A continuación el sistema determina la nueva posición y actualiza las coordenadas del nodo.	
Referencias:	CU17, RF2.1.4, CU23 (extend), CU12 (include).	
Precondiciones:	Nodo previamente seleccionado.	
Curso Normal de los Eventos		
Acción del Actor		**Respuesta del Sistema**
1 – Desplaza el nodo en el componente de visualización 3D haciendo uso del teclado.		1.1 – Actualiza las propiedades del nodo. 1.2 – Invoca el caso de uso: **Mostrar propiedades.** 1.3 – Se actualizan las propiedades de las aristas asociadas, invocando el caso de uso: **Actualizar arista.** 1.4 – Se actualiza la representación geométrica del nodo en el componente de visualización 3D. 1.5 – Termina el caso de uso.
Prioridad:	Crítico.	

Tabla 12 Descripción del caso de uso "Modificar nodo".

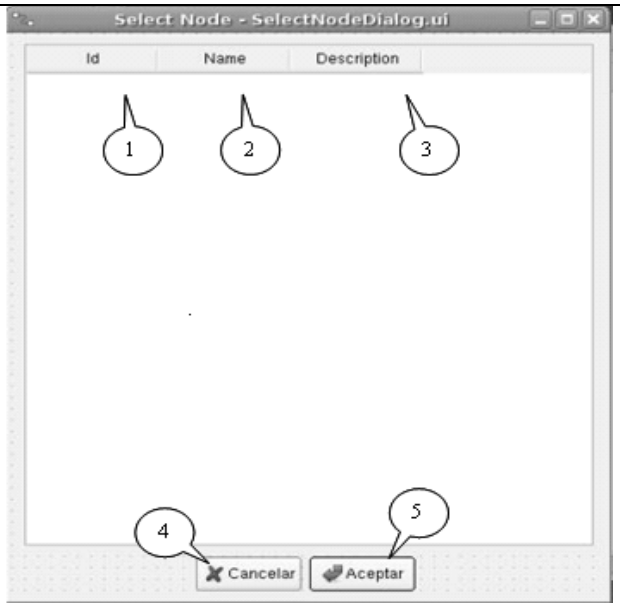
Caso de uso:	**Modificar nodo.**
Actor(es):	Diseñador (inicia).
Propósito:	Cambiar las propiedades de un nodo del grafo de caminos.
Resumen:	El caso de uso inicia cuando el usuario del sistema modifica el nodo, cambiando sus propiedades en el editor de propiedades. A continuación el sistema actualiza las propiedades y la representación geométrica en el componente de visualización 3D.
Referencias:	CU14, RF2.1.3, CU12 (include), CU23 (extend).
Precondiciones:	Nodo previamente seleccionado, y sus propiedades mostradas en el editor de propiedades.

Curso Normal de los Eventos	
Acción del Actor	**Respuesta del Sistema**
1 – Modifica los datos en el editor de propiedades	1.1 – Actualiza las propiedades del nodo. 1.2 – Modifica la geometría, en el componente de visualización de 3D. 1.3 – Actualiza cada una de las aristas asociadas al nodo (de haberlas) invocando el caso de uso: **Actualizar arista.** 1.4 – Invoca el caso de uso: **Mostrar propiedades.** 1.5 – Termina el caso de uso.
Post-condiciones:	Modificadas las propiedades geométricas del nodo.
Prioridad:	Crítico.

Tabla 13 Descripción del caso de uso "Eliminar nodo"

Caso de uso:	**Eliminar nodo.**
Propósito:	Permitir eliminar un nodo en el grafo de caminos.
Resumen:	El caso de uso inicia cuando el usuario del sistema accede a la opción de eliminar nodo. A continuación el nodo seleccionado es eliminado del grafo de caminos.
Referencias:	CU19, RF2.1.5, CU20 (extend).
Precondiciones	Nodo previamente seleccionado.
Curso Normal de los Eventos	
Acción del Actor	**Respuesta del Sistema**
1 – Selecciona la opción "Eliminar".	1.1 – En caso que el nodo tenga aristas asociadas, las elimina invocando el caso de uso: **Eliminar arista**. 1.2 – Limpia el editor de propiedades. 1.3 – Elimina el nodo. 1.4 – Elimina la visualización correspondiente al nodo. 1.5 – Termina el caso de uso.
Post-condiciones:	Eliminado el nodo del grafo de caminos y su respectiva representación gráfica.
Prioridad:	Crítico.

Tabla 14 Descripción del caso de uso "Buscar nodo".

<table>
<tr><td>Caso de uso:</td><td colspan="2">Buscar nodo.</td></tr>
<tr><td>Propósito:</td><td colspan="2">Permitir buscar cualquier nodo del grafo de caminos.</td></tr>
<tr><td>Resumen:</td><td colspan="2">El caso de uso es invocado externamente y brinda funcionalidades para buscar cualquier nodo del grafo de caminos, devolviendo el ID del nodo seleccionado.</td></tr>
<tr><td>Referencias:</td><td colspan="2">CU13.</td></tr>
<tr><td>Interfaz I</td><td> 
</td><td>1 – Identificador del nodo
2 – Nombre del nodo
3 – Descripción del nodo
4 – Cancelar
5 – Aceptar</td></tr>
<tr><td colspan="3">Curso Normal de los Eventos</td></tr>
<tr><td colspan="2">Acción del Actor</td><td>Respuesta del Sistema</td></tr>
<tr><td colspan="2"></td><td>1.1 – Muestra la interfaz I con un listado de los nodos del grafo.</td></tr>
<tr><td colspan="2">2 – Selecciona el nodo y luego la opción "Aceptar".</td><td>2.1 – Devuelve el ID del nodo seleccionado
2.2 – Termina el caso de uso.</td></tr>
<tr><td colspan="3">Cursos Alternos</td></tr>
<tr><td colspan="3">2 – Si selecciona la opción (4) cancela la operación y termina el caso de uso
2.1 – Si no se selecciona ningún nodo se muestra un mensaje al usuario.</td></tr>
<tr><td>Post-condiciones:</td><td colspan="2">Se devuelve el ID del nodo seleccionado.</td></tr>
<tr><td>Prioridad:</td><td colspan="2">Secundario.</td></tr>
</table>

Sub-paquete "Gestión de Aristas".

Tabla 15 Descripción del caso de uso "Crear aristas".

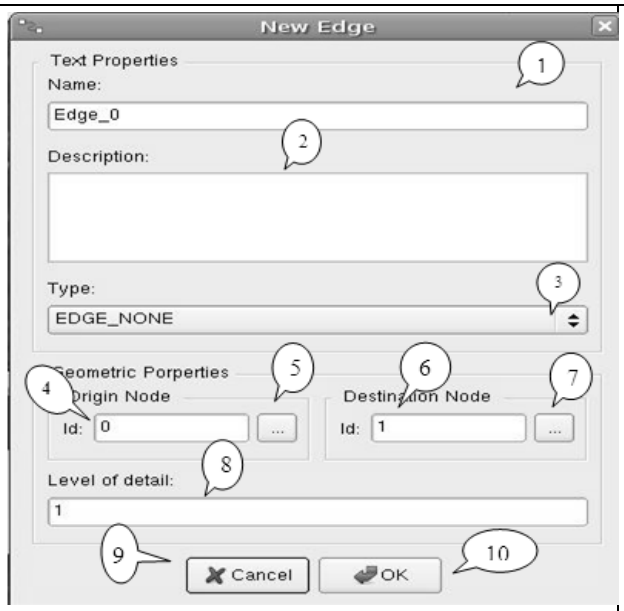
Caso de uso:	**Crear arista.**
Propósito:	Construir y agregar aristas al grafo de caminos.
Resumen:	El caso de uso es invocado externamente y recibe los datos de la nueva arista, construye la arista y la agrega al grafo de caminos.
Referencias:	CU7, RF2.2.1
Curso Normal de los Eventos	
Respuesta del Sistema	
1.1 – Crea una arista con los atributos provistos y lo adiciona al grafo de caminos. 1.2 – Actualiza la estructura de control de visualización para agregar la representación gráfica de la nueva arista. 1.3 – Termina el caso de uso.	
Post-condiciones:	Se creó una arista, se añadió al grafo y se visualizó al usuario.
Prioridad:	Crítico.

Tabla 16 Descripción del caso de uso "Crear arista manualmente".

Caso de uso:	**Crear arista manualmente.**
Propósito:	Construir y agregar aristas manualmente.
Resumen:	El caso de uso inicia cuando el usuario desea crear una nueva arista manualmente. Para ello selecciona el nodo origen, luego el nodo destino, y el sistema crea una arista entre ellos, la visualiza y muestra sus propiedades.
Referencias:	CU8, CU7 (Include), CU12 (Include).
Precondiciones	Opción arista en la barra de herramienta previamente activada.
Curso Normal de los Eventos	
Acción del Actor	**Respuesta del Sistema**
1 – Da clic sobre el primer nodo.	1.1 – Determina el elemento seleccionado. 1.2 – Invoca el caso de uso: **Mostrar propiedades**, mostrando propiedades del nodo inicial.

2 – Da clic sobre el segundo nodo.	2.1 – Determina el elemento seleccionado. 2.2 – Invoca el caso de uso: **Crear arista.** 2.3 – Invoca el caso de uso: **Mostrar propiedades**, mostrando propiedades de la arista. 2.4 – Termina el caso de uso.
Cursos Alternos	
1.1 , 2.1 – Si no se seleccionó un nodo termina el caso de uso.	
Post-condiciones:	Se creó una arista y se visualizó al usuario.
Prioridad:	Crítico.

Tabla 17 Descripción del caso de uso "Crear arista con asistente".

Caso de uso:	**Crear arista con asistente.**
Actor(es):	Diseñador (inicia).
Propósito:	Crear aristas del grafo de caminos.
Resumen:	El caso de uso inicia cuando el usuario del sistema selecciona la opción "Nueva arista". A continuación el sistema solicita los datos de la nueva arista, el usuario los provee y luego el sistema la crea y la visualiza.
Referencias:	CU9, CU7 (Include), CU12 (Include), CU13 (Extend).
Interfaz I	 1 – Nombre 2 – Descripción 3 – Tipo de arista 4 – Introducir Id de nodo origen 5 – Buscar nodo origen. 6 – Introducir Id de nodo destino 7 – Buscar nodo destino 8 – Nivel de detalle 9 – Cancelar 10 – Aceptar
Curso Normal de los Eventos	
Acción del Actor	**Respuesta del Sistema**

1 – Selecciona la opción “Nueva arista”.	1.1 – Muestra la Interfaz I con datos por defecto de la nueva arista, permitiendo modificarlos.
2 – Llena los campos del formulario. 3 – Si desea buscar el nodo origen manualmente selecciona la opción (5), (ir a la sección Seleccionar nodo). 4 – Si desea buscar el nodo destino manualmente selecciona la opción (7), ir a la sección seleccionar nodo. 5 – Da clic en Aceptar.	5.1 – Invoca el caso de uso: **Crear arista.** 5.2 – Invoca el caso de uso: **Mostrar propiedades**. 5.3 – Termina el caso de uso.
Cursos Alternos	
5 – Si selecciona Cancelar, termina el caso de uso.	
Sección Seleccionar nodo.	
	1 – Invoca el caso de uso: **Buscar nodo.** **Resultado:** ID del nodo seleccionado.
Prioridad:	Crítico.

Tabla 18 Descripción del caso de uso “Modificar arista”.

Caso de uso:	**Modificar arista.**
Actor(es):	Diseñador (inicia).
Propósito:	Modificar las propiedades de una arista del grafo de caminos.
Resumen:	El caso de uso inicia cuando el usuario modifica las propiedades de la arista en el editor de propiedades. A continuación el sistema actualiza las propiedades de la arista y su representación geométrica.
Referencias:	CU15, CU12(include), CU23(include)
Precondiciones:	Arista previamente seleccionada, y sus propiedades mostradas en el editor de propiedades.
Curso Normal de los Eventos	
Acción del Actor	**Respuesta del Sistema**
1 – Modifica los datos de la arista en el editor de propiedades	1.1 – Invoca al caso de uso: **Actualizar arista** 1.2 – Invoca al caso de uso: **Mostrar propiedades** 1.3 – Termina el caso de uso.
Post-condiciones:	Modificadas las propiedades de la arista.

Prioridad:	Crítico.

Tabla 19 Descripción del caso de uso "Actualizar arista".

Caso de uso:	**Actualizar arista.**
Actor(es):	Diseñador (inicia).
Propósito:	Actualizar las propiedades de una arista del grafo de caminos.
Resumen:	El caso de uso es invocado externamente por otros casos de uso que necesiten modificar las propiedades de una arista. A continuación el sistema actualiza las propiedades de la arista y su representación geométrica.
Referencias:	CU23, RF2.2.3
Curso Normal de los Eventos	
Respuesta del Sistema	
1.1 – Actualiza las propiedades de la arista en el grafo de caminos. 1.2 – Actualiza la geometría de la arista, en el componente de visualización 3D. 1.3 – Termina el caso de uso.	
Post-condiciones:	Actualizadas las propiedades de la arista.
Prioridad:	Crítico.

Tabla 20 Descripción del caso de uso "Eliminar arista manualmente".

Caso de uso:	**Eliminar arista manualmente.**	
Propósito:	Permitir eliminar aristas del grafo de caminos.	
Resumen:	El caso de uso inicia cuando el usuario del sistema accede a la opción de eliminar elemento, y la arista previamente seleccionada es eliminada del grafo de caminos.	
Referencias:	CU21, CU20 (include).	
Precondiciones	Arista previamente seleccionada.	
Curso Normal de los Eventos		
Acción del Actor		**Respuesta del Sistema**
1- Selecciona la opción "Eliminar".		1.1 – Invoca el caso de uso: **Eliminar arista.** 1.2 – Limpia el editor de propiedades. 1.3 – Termina el caso de uso.
Post-condiciones:	Eliminada la arista del grafo de caminos y su respectiva representación gráfica.	

Prioridad:	Crítico.

Tabla 21 Descripción del caso de uso "Eliminar arista".

Caso de uso:	**Eliminar arista.**
Propósito:	Permitir eliminar aristas del grafo de caminos.
Resumen:	El caso de uso es invocado externamente con el objetivo de eliminar una arista determinada.
Referencias:	CU20, RF2.2.5
Precondiciones	Arista previamente seleccionada.
Curso Normal de los Eventos	
Respuesta del Sistema	
1 – Elimina la arista del grafo de caminos. 2 – Elimina la representación gráfica de la arista del componente de visualización 3D.	
Post-condiciones:	Eliminada la arista del grafo de caminos y su respectiva representación gráfica.
Prioridad:	Crítico.

Sub-paquete "Gestión de vértices de control".

Tabla 22 Descripción del caso de uso "Crear vértice de control manualmente".

Caso de uso:	**Crear vértice de control manualmente.**
Propósito:	Construir y agregar puntos de control a la arista.
Resumen:	El caso de uso inicia cuando el usuario selecciona la opción "Nuevo vértice de control" para agregar un punto de control a la arista previamente seleccionada. A continuación el sistema crea el nuevo vértice de control, actualiza la geometría de la arista y muestra las propiedades y visualización del vértice de control creado.
Referencias:	CU11, CU10 (Include), CU12 (Include).
Precondiciones	Arista previamente seleccionada.

Curso Normal de los Eventos	
Acción del Actor	**Respuesta del Sistema**
1 – Selecciona la opción "Nuevo vértice de control".	1.1 – Invoca el caso de uso: **Crear vértice de control.** 1.2 – Invoca el caso de uso: **Mostrar propiedades.** 1.3 – Termina el caso de uso.

Post-condiciones:	Se agregó un nuevo vértice de control a la arista seleccionada y se visualizó.
Prioridad:	Crítico.

Tabla 23 Descripción del caso de uso "Crear vértice de control".

Caso de uso:	**Crear vértice de control.**
Propósito:	Construir y agregar puntos de control a una arista.
Resumen:	El caso de uso es invocado externamente con el objetivo de crear un nuevo vértice de control a una arista determinada. A continuación el sistema crea el nuevo vértice de control y actualiza la geometría de la arista.
Referencias:	CU10, RF2.2.6.1, CU23(include)
Curso Normal de los Eventos	
Respuesta del Sistema	
1.1 – Crea el vértice de control. 1.2 – Muestra el vértice de control en el componente de visualización 3D. 1.3 – Invoca el caso de uso: **Actualizar arista.**	
Post-condiciones:	Se creó un nuevo vértice de control a la arista seleccionada y se actualizó la geometría de dicha arista.
Prioridad:	Crítico.

Tabla 24 Descripción del caso de uso "Modificar vértice de control".

Caso de uso:	**Modificar vértice de control.**
Propósito:	Modificar las propiedades de un vértice de control del grafo de caminos.
Resumen:	El caso de uso inicia cuando el usuario modifica las propiedades del vértice de control en el editor de propiedades. A continuación el sistema actualiza las propiedades y actualiza la representación del vértice de control en el componente de visualización, así como actualiza la arista correspondiente al vértice de control.
Referencias:	CU16, RF2.2.6.2, CU12 (Include), CU23 (Include).
Precondiciones	Vértice de control previamente seleccionado, y sus propiedades mostradas en el editor de propiedades.
Curso Normal de los Eventos	
Acción del Actor	**Respuesta del Sistema**

1 – Modifica los datos en el editor de propiedades.	1.1 – Actualiza las propiedades del vértice de control. 1.2 – Actualiza la representación del vértice de control en el componente de visualización 3D. 1.3 – Invoca el caso de uso: **Actualizar arista.** 1.4 – Invoca el caso de uso: **Mostrar propiedades**. 1.5 – Termina el caso de uso.
Post-condiciones:	Se modificó el vértice de control y se visualizó al usuario.
Prioridad:	Crítico.

Tabla 25 Descripción del caso de uso "Desplazar vértice de control".

Caso de uso:	**Desplazar vértice control.**
Actor(es):	Diseñador (inicia).
Propósito:	Trasladar de posición los vértices de control del grafo de caminos, con el objetivo de deformar la arista a la cual están asociados.
Resumen:	El caso de uso inicia cuando el usuario se dispone a desplazar un vértice de control previamente seleccionado, haciendo uso del teclado. A continuación el sistema determina la nueva posición y actualiza las propiedades del vértice y la arista correspondiente.
Referencias:	CU18, RF2.2.6.3, CU12 (Include), CU23 (Include).
Precondiciones:	Vértice de control previamente seleccionado.
Curso Normal de los Eventos	
Acción del Actor	**Respuesta del Sistema**
1 – Desplaza el vértice de control en el componente de visualización 3D haciendo uso del teclado.	1.1 – Actualiza las propiedades del vértice de control 1.2 – Actualiza la representación geométrica del vértice de control en el componente de visualización 3D. 1.3 – Actualiza la arista asociada al vértice de control, invocando el caso de uso: **Actualizar arista.** 1.4 – Invoca el caso de uso: **Mostrar propiedades.** 1.5 – Termina el caso de uso.
Prioridad:	Crítico.

Tabla 26 Descripción del caso de uso "Eliminar vértice de control".

Caso de uso:	**Eliminar vértice de control.**
Propósito:	Permitir eliminar un vértice de control del grafo de caminos.
Resumen:	El caso de uso inicia cuando el usuario del sistema accede a la opción de eliminar elemento en la barra de herramientas. A continuación el sistema elimina el vértice de control previamente seleccionado y actualiza la arista a la cual estaba asociado.
Referencias:	CU22, RF2.2.6.4, CU23 (include).
Precondiciones	Vértice de control previamente seleccionado.
Curso Normal de los Eventos	
Acción del Actor	**Respuesta del Sistema**
1 – Selecciona la opción "Eliminar".	1.1 – Elimina el vértice de control. 1.2 – Invoca el caso de uso: **Actualizar arista.** 1.3 – Limpia el editor de propiedades. 1.4 – Termina el caso de uso.
Post-condiciones:	Eliminado el vértice de control del grafo de caminos y actualizada su arista correspondiente.
Prioridad:	Crítico.

Conclusiones

En el presente capítulo se definió qué espera el usuario con este sistema. Para ello quedaron establecidos sus requisitos funcionales y se describieron los casos de uso que permitirán al usuario obtener los resultados esperados.

CAPÍTULO 4: ANÁLISIS Y DISEÑO DEL SISTEMA

Introducción

La primera parte del capítulo tratará todo lo referente a los diagramas de clases del análisis del sistema propuesto. Su objetivo es brindar una primera visión de las posibles clases del diseño a un alto nivel, pero donde se dejan ver sus relaciones y responsabilidades.

A continuación se presentan los diagramas de clases del diseño por paquetes como resultado del refinamiento de las etapas anteriores. El diagrama de clases es uno de los elementos más importantes dentro de un proyecto de software, ya que brinda una visión bastante completa de todo el sistema, mostrando sus clases, métodos, atributos y las relaciones entre ellos. Se presentan además los diagramas de secuencia de la realización de los casos de uso que intervendrán en el primer ciclo de desarrollo del proyecto.

4.1 Diagrama de paquetes de clases del análisis.

En el siguiente diagrama se muestra la dependencia entre los paquetes del análisis.

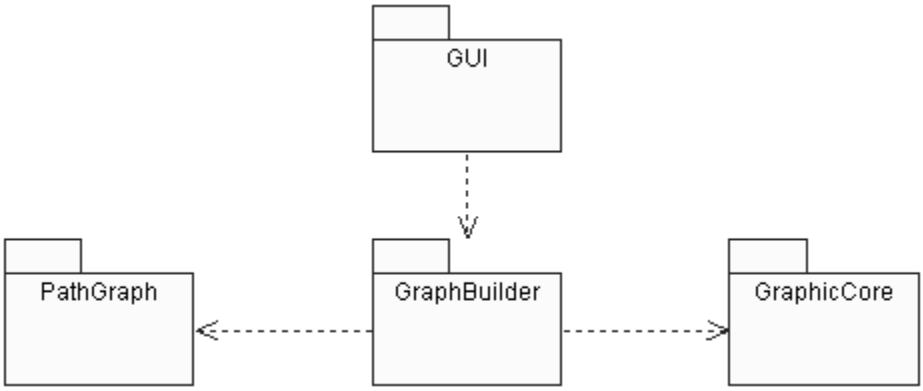


Fig. 17 Diagrama de paquetes del análisis.

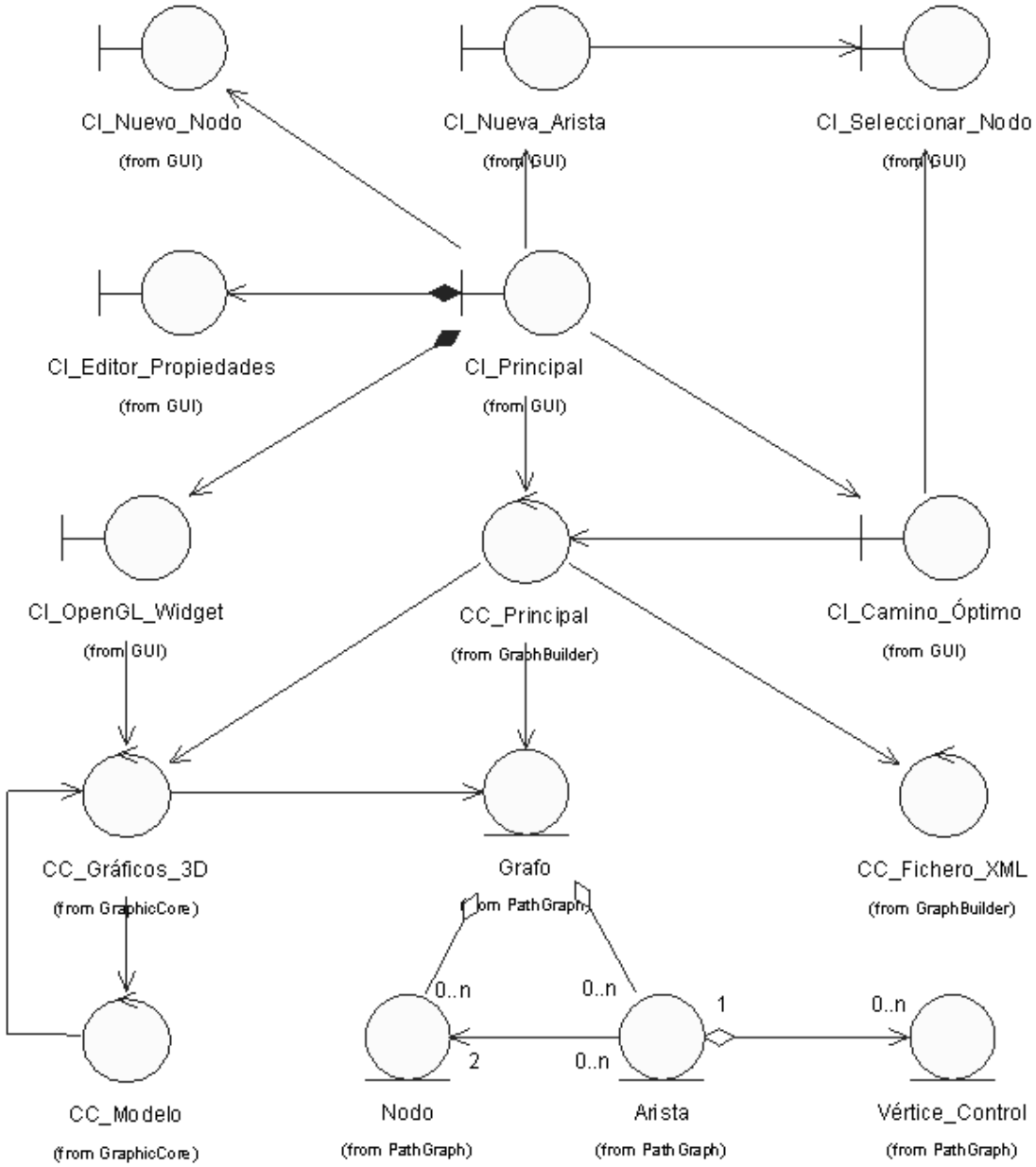


Fig. 18 Diagrama de clases del análisis.

Este diagrama muestra la relación entre las distintas clases de análisis de los distintos paquetes. CI_Principal es la ventana principal de la aplicación. CI_OpenGL_Widget es un componente visual que será la interfaz donde se dibujarán los gráficos 3D. CC_Gráficos_3D es la representación de la interfaz de SceneToolkit.

4.2 Diagrama de clases del diseño.

Antes de mostrar los diagramas de clases del diseño, se aclararán dos cuestiones importantes para su comprensión:

- La nomenclatura utilizada en dichos diagramas se explica en el epígrafe “Estándares de codificación” del siguiente capítulo.
- En la realización de este proyecto, el código se generó con la herramienta Rational Rose Enterprise Edition (2003), la cual brinda la posibilidad de generar los métodos de acceso a miembros de clases (“gets” y “sets”), constructores por defecto, constructores de copia y destructores, aunque no se hallan incluido en las especificaciones de las clases; por tanto, y para evitar su repetición, se omiten en las clases los métodos antes mencionados. En algunos casos se incluyen algunos constructores, pero no son los “por defecto”.

El sistema propuesto consta de cinco paquetes: GUI, GraphBuilder, GraphicCore, PathGraph y libSceneToolkitQT. En el paquete GUI se encuentran las clases de manejo de interfaz de usuario, encargadas de la entrada y salida de información de forma visual, así como del manejo de eventos. En el paquete GraphBuilder se encuentra el controlador principal de la aplicación y la lógica del negocio. En el paquete PathGraph se encuentran las entidades del negocio, encargadas del almacenamiento de la información del grafo. El paquete GraphicCore brinda una interfaz para el trabajo con SceneToolkit, facilitando el trabajo con la misma y abstrayendo al programador de su estructura interna y funcionamiento. Por último, libSceneToolkitQT contiene los binarios de SceneToolkit, utilizados para las operaciones de dibujado 3D y tratamiento de ficheros 3DX.

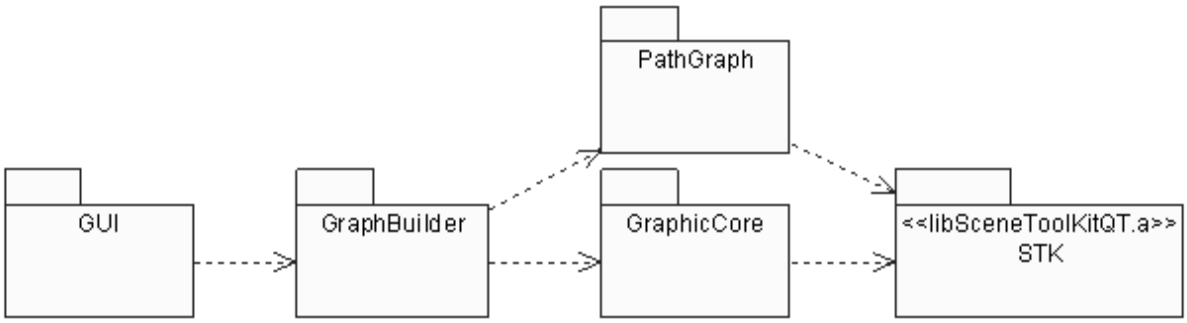


Fig. 19 Diagrama de paquetes del “diseño”.

La figura 19 muestra la dependencia entre los distintos paquetes del diseño. En este punto hay que destacar que GraphicCore depende completamente de SceneToolkit, de la que también depende PathGraph, pero de manera leve ya que solamente utiliza la clase CVector3.

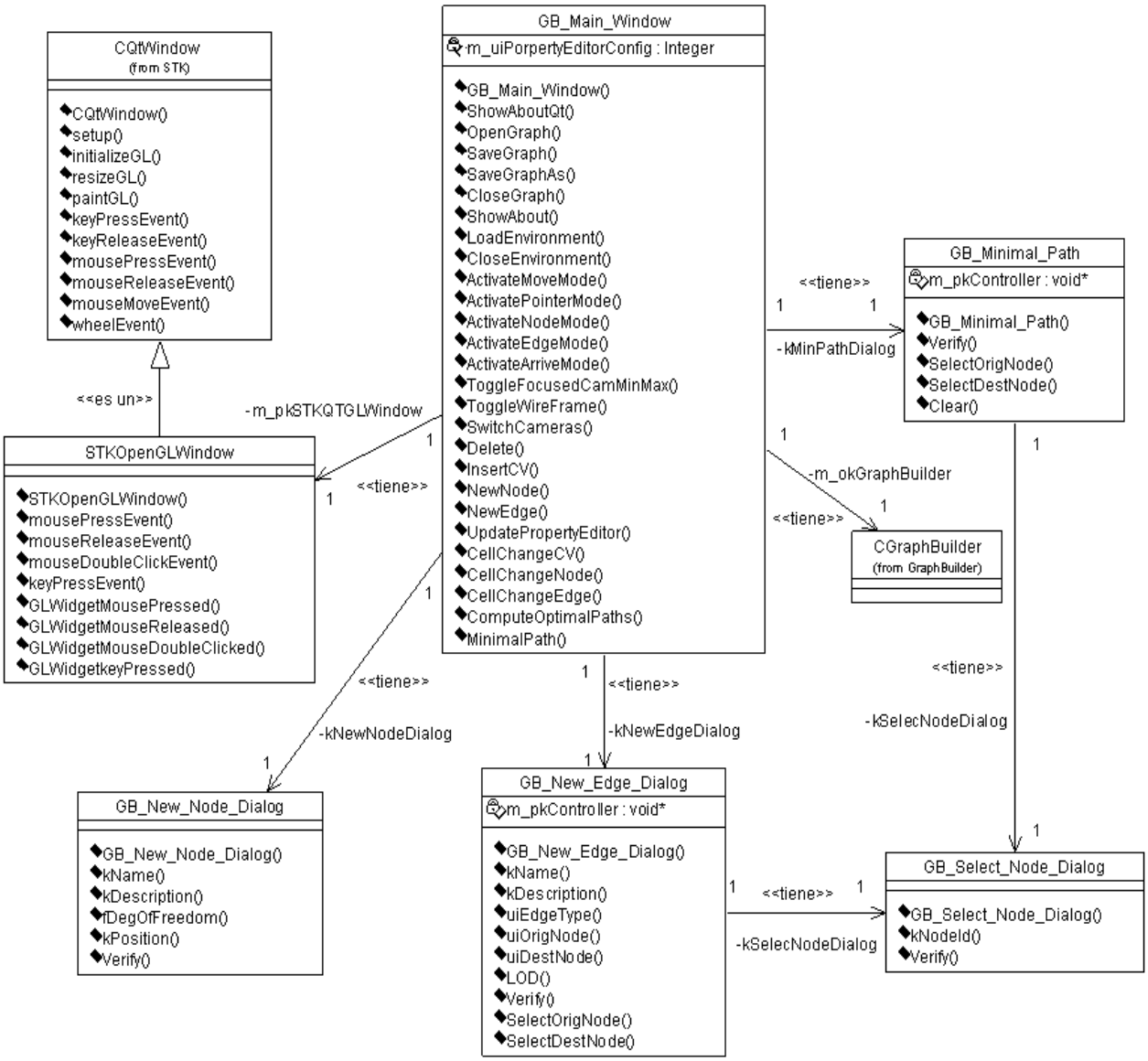


Fig. 20 Diagrama de clases del diseño, paquete "GUI".

En este diagrama se muestran todas las clases de interfaz gráfica de usuario y las relaciones entre ellas. Fíjese que GB_Main_Window es la ventana principal. STKOpenGLWindow hereda de CQtWindow, quien se encarga de gestionar los eventos de STK y de dibujar la escena 3D.

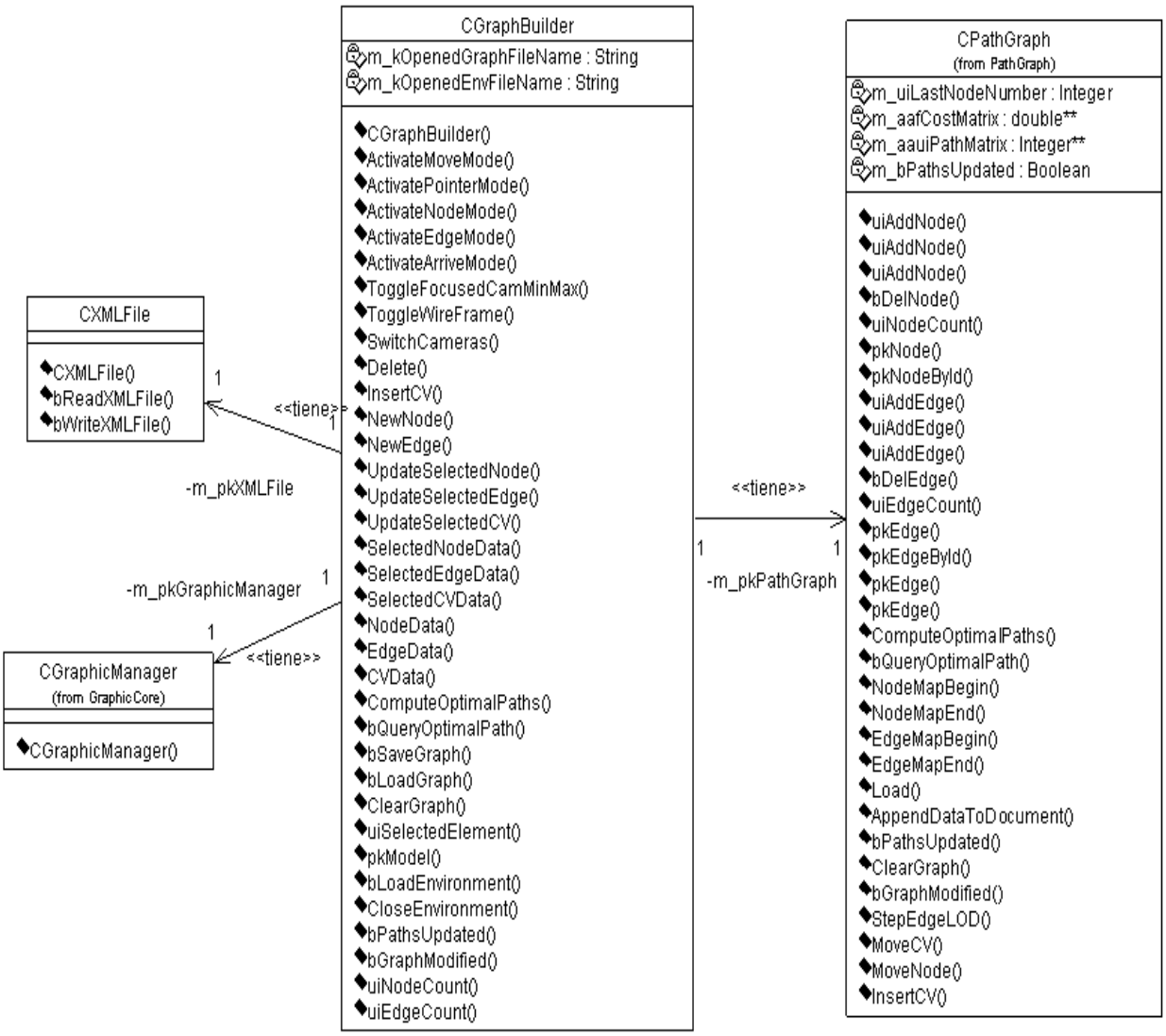


Fig. 21 Diagrama de clases del diseño, paquete "GraphBuilder".

En este diagrama se aprecia la relación entre las clases CPathGraph y CGraphicManager con CGraphBuilder, que contiene la lógica del negocio y representa el controlador principal de la aplicación.

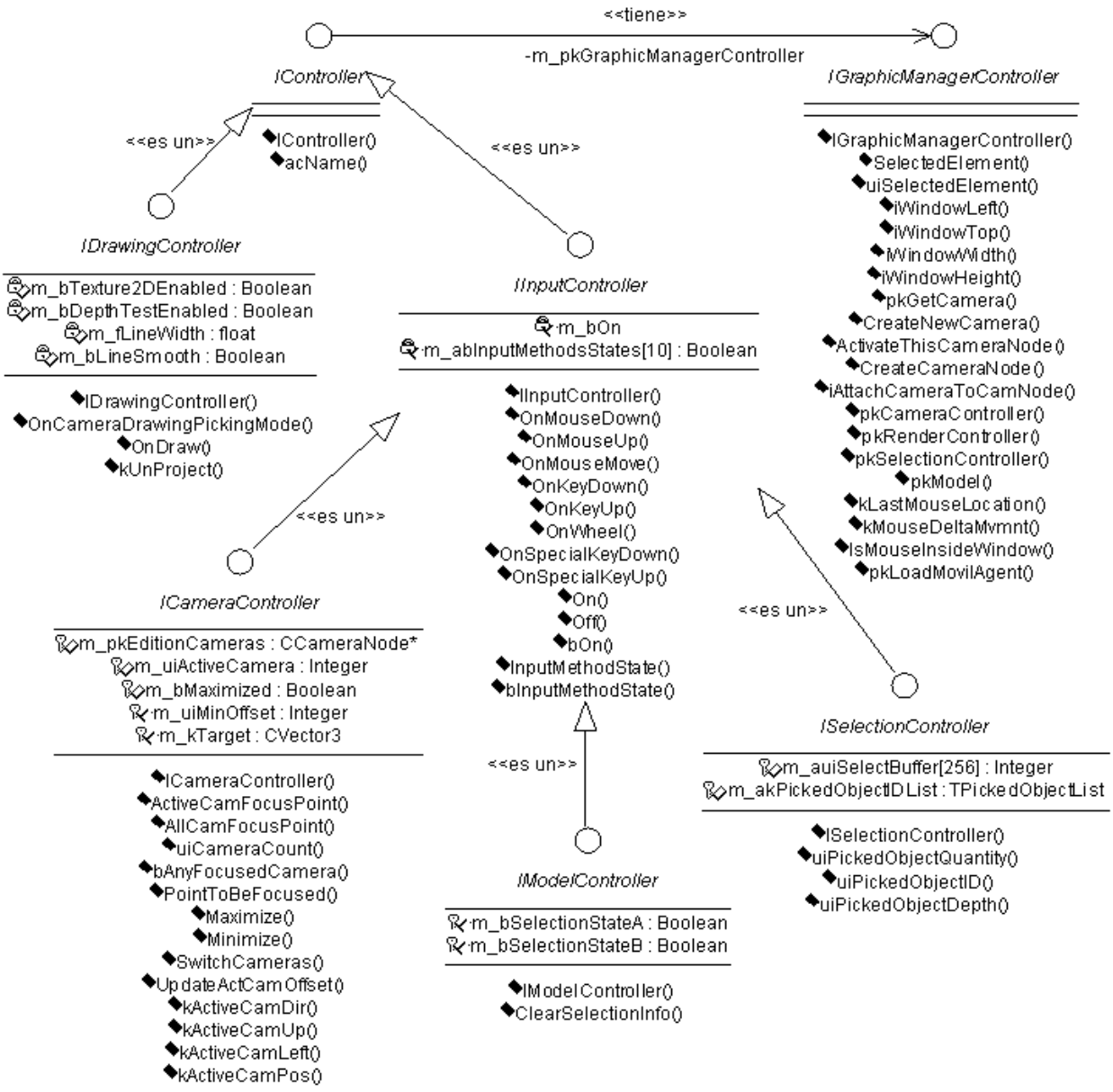


Fig. 22 Diagrama de clases del diseño, paquete "Graphic Core", vista "Interfaces".

En esta imagen se puede observar cada uno de los tipos de controladores del framework de STK Editor. IGraphicManagerController es el encargado de brindarles el acceso a las funcionalidades de CGraphicManager. Es necesario aclarar que esta jerarquía de controladores fue un aporte de este trabajo al desarrollo del Framework.

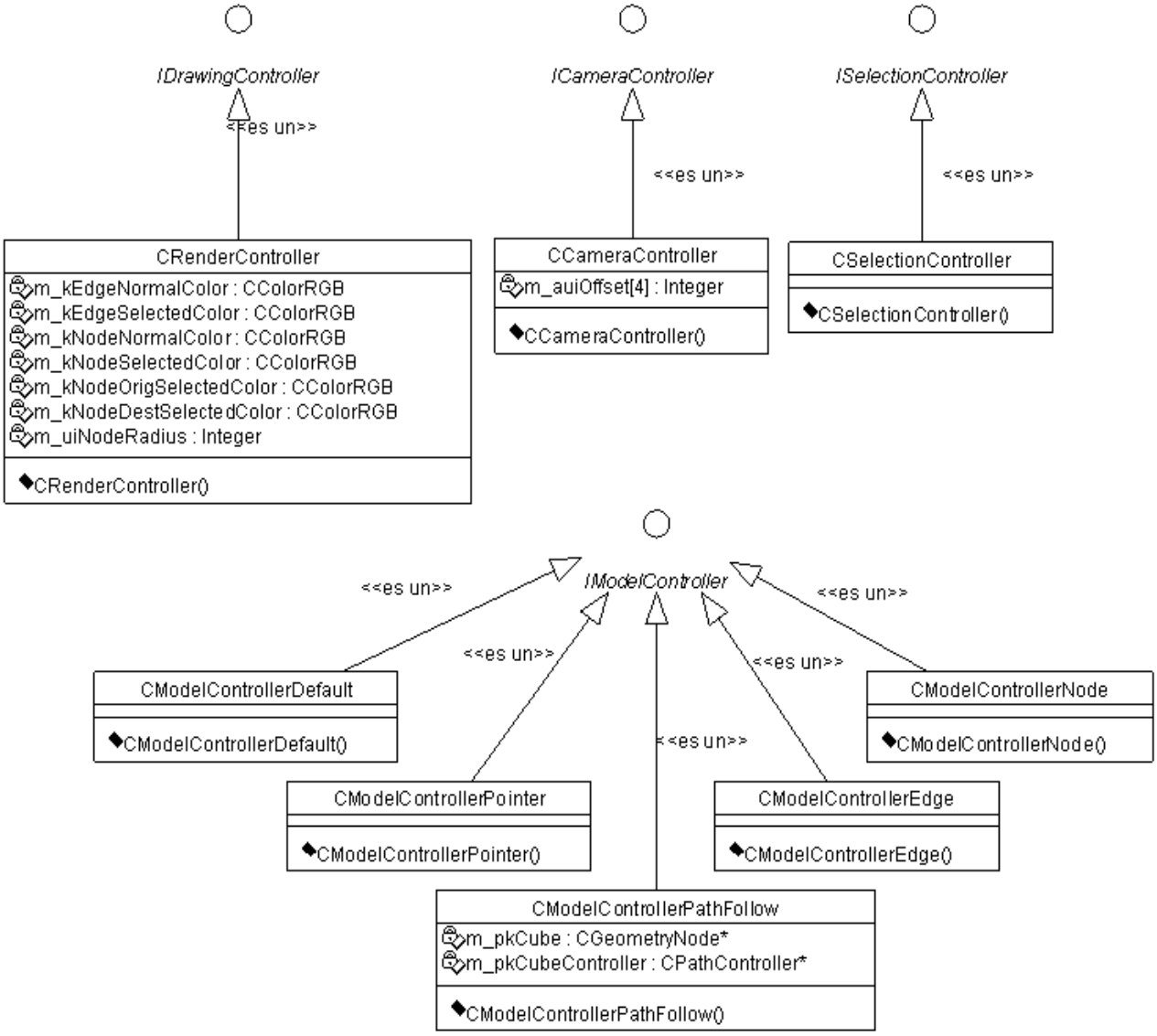


Fig. 23 Diagrama del diseño, paquete "Graphic Core", vista "Controladores".

Esta figura especifica cómo el usuario del framework de STK Editor debe heredar las interfaces de los controladores e implementar su propio comportamiento.

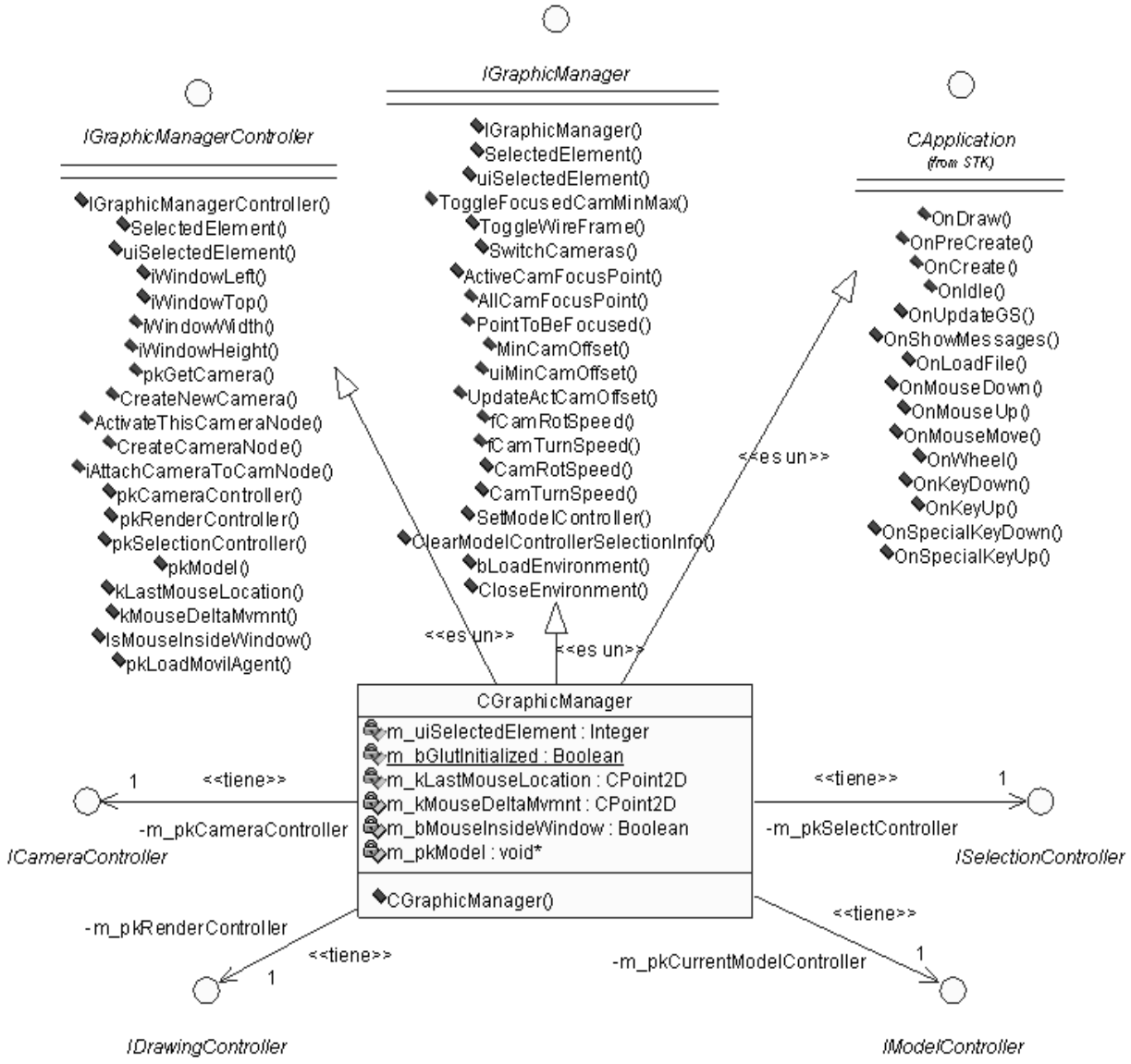


Fig. 24 Diagrama de clases del diseño, paquete "Graphic Core", vista "Graphic Manager".

Observe en esta imagen cómo CGraphicManager hereda de 3 interfaces; al heredar de CApplication garantiza las funcionalidades de STK; implementa la interfaz IGraphicManager para brindar una interfaz al exterior del paquete (a GraphBuilder) el cual no debe conocer otra información que no sea la que se le brinda; implementa la interfaz IGraphicManagerController para brindar a los controladores acceso a aquella funcionalidad que les es permitida (por ejemplo: un controlador no tiene permitido llamar funciones de manejo de eventos en CGraphicManager, ya que la aplicación podría caer en un ciclo infinito).

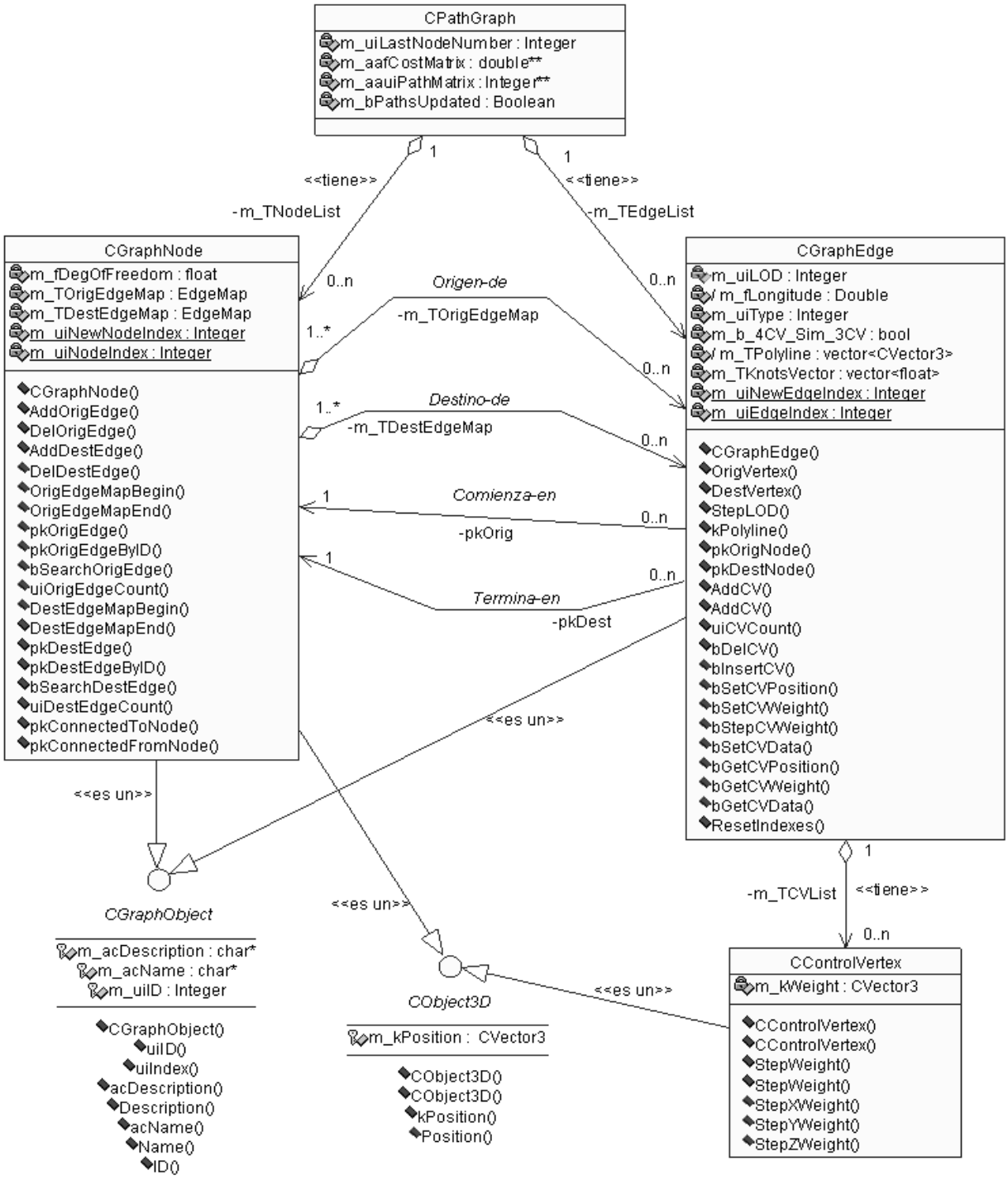


Fig. 25 Diagrama de clases del diseño, paquete "PathGraph".

Esta imagen muestra la relación entre las distintas clases del dominio.

4.3 Diagramas de secuencia

A continuación siguen los diagramas de secuencia agrupados por paquetes de casos de uso.

Paquete "Entrada/Salida fichero".

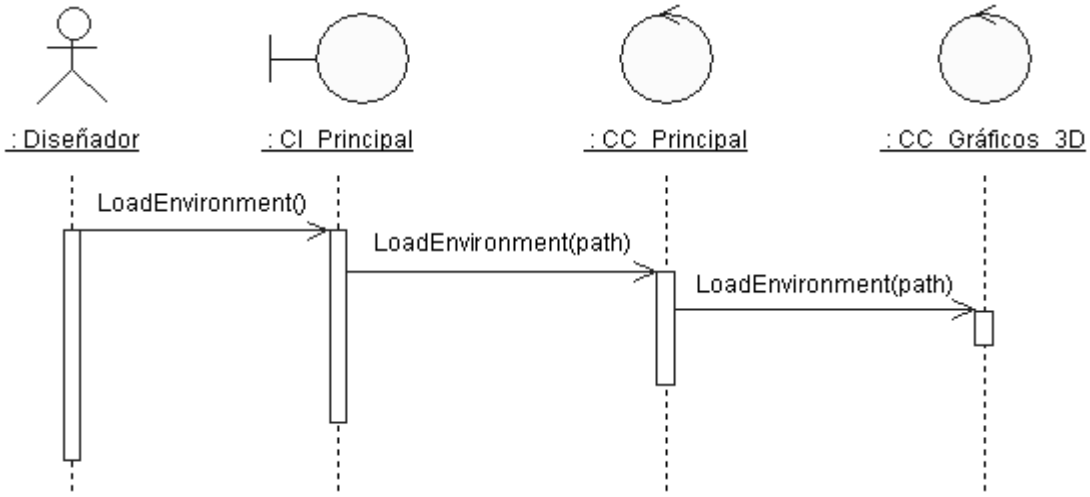


Fig. 26 Diagrama de secuencia del caso de uso "Cargar entorno".

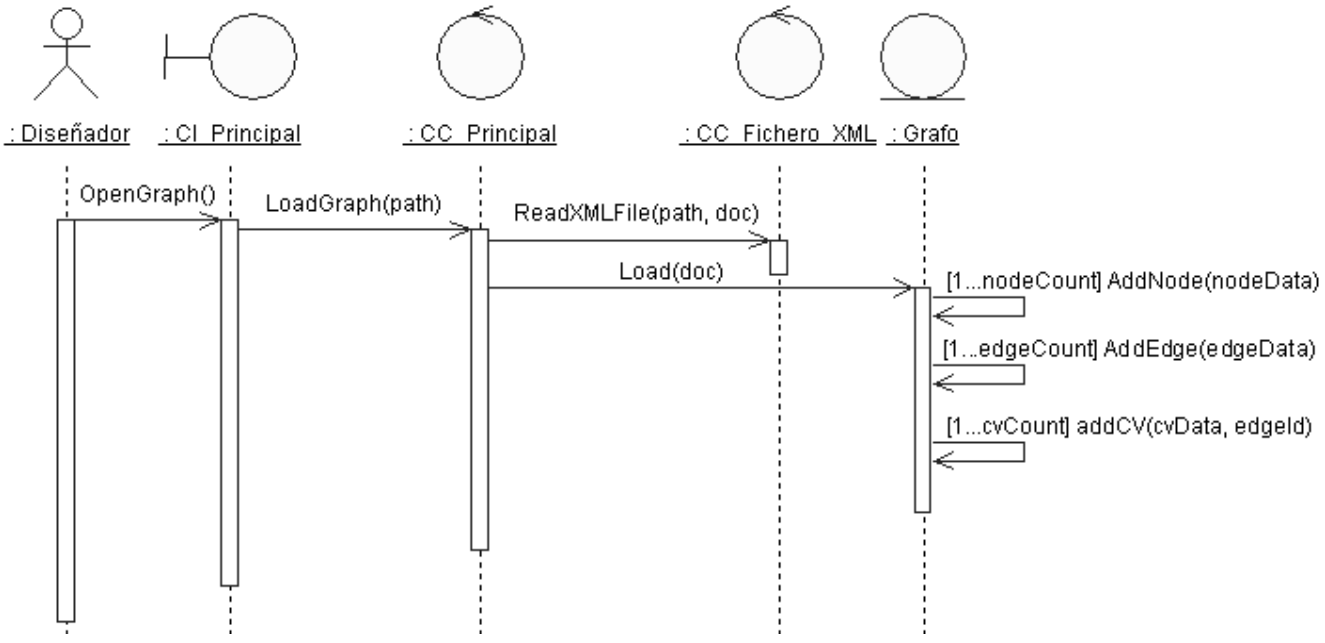


Fig. 27 Diagrama de secuencia del caso de uso "Cargar grafo".

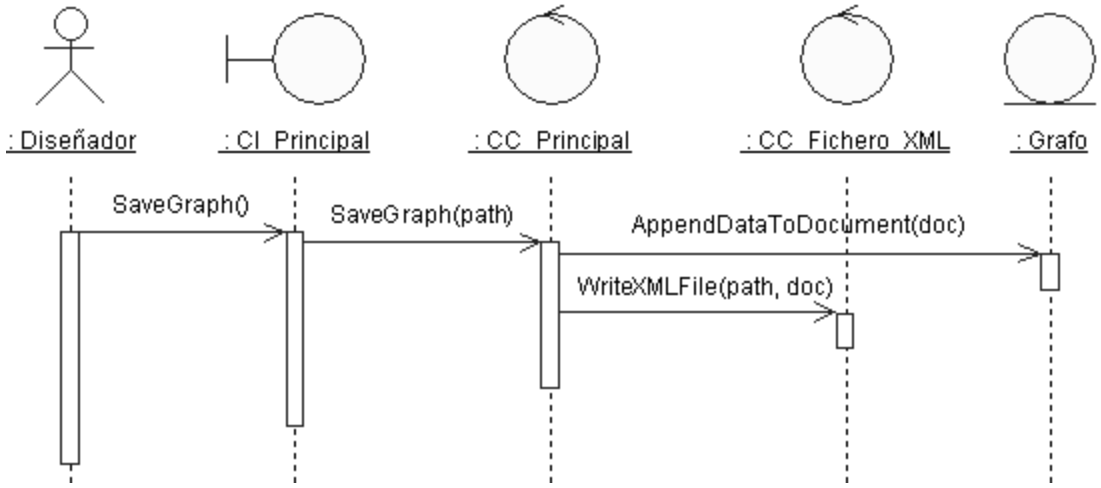


Fig. 28 Diagrama de secuencia del caso de uso "Salvar grafo".

Paquete "Planificación".

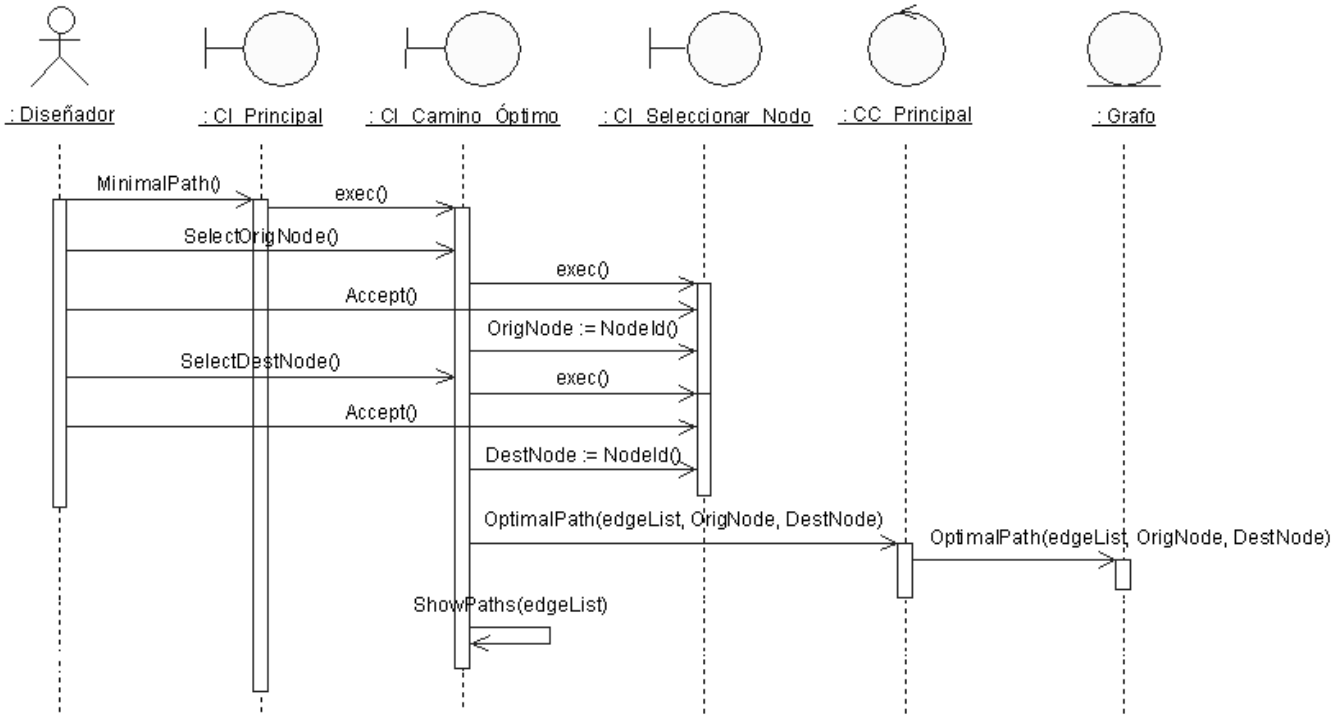


Fig. 29 Diagrama de secuencia del caso de uso "Consultar caminos óptimos".

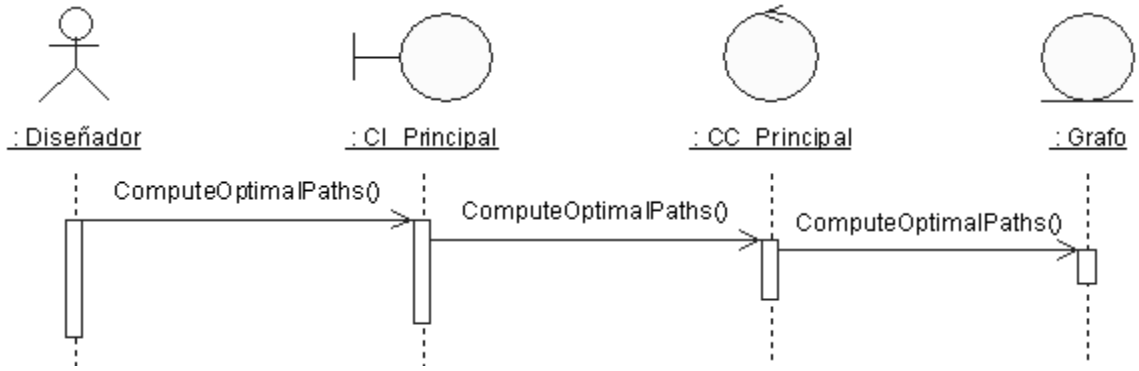


Fig. 30 Diagrama de secuencia del caso de uso "Calcular caminos óptimos".

Paquete "Gestión de grafos".

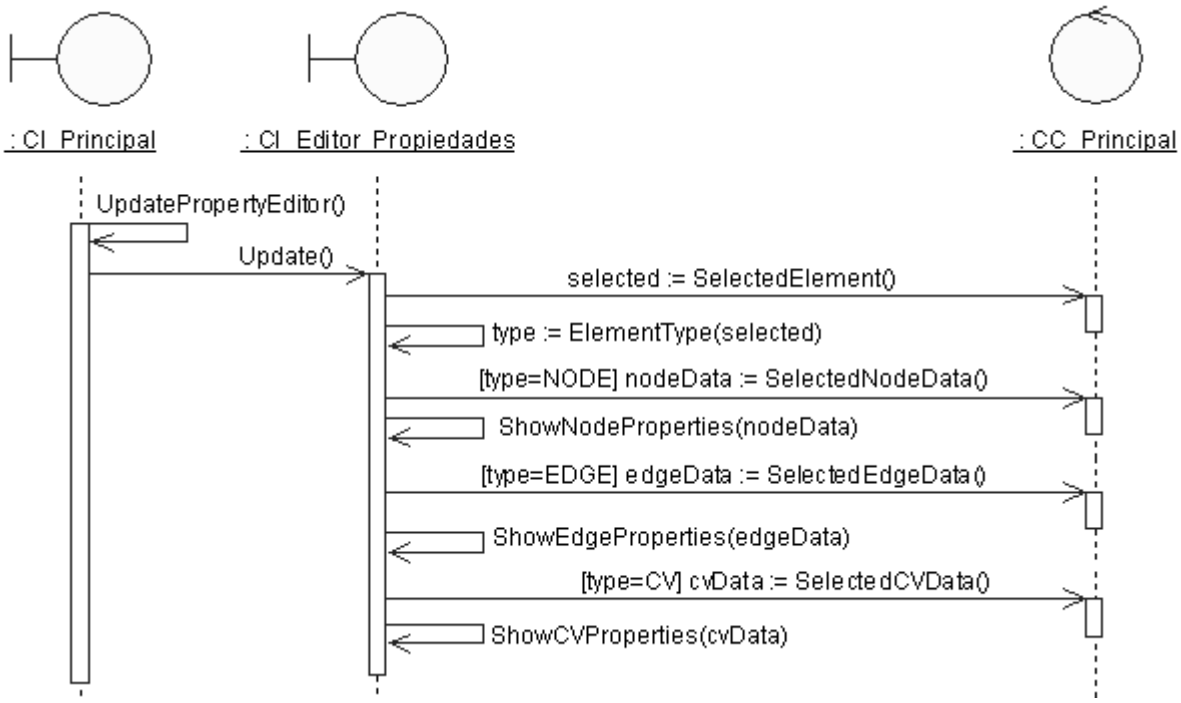


Fig. 31 Diagrama de secuencia del caso de uso "Mostrar propiedades".

Sub-paquete "Gestión de Nodos".

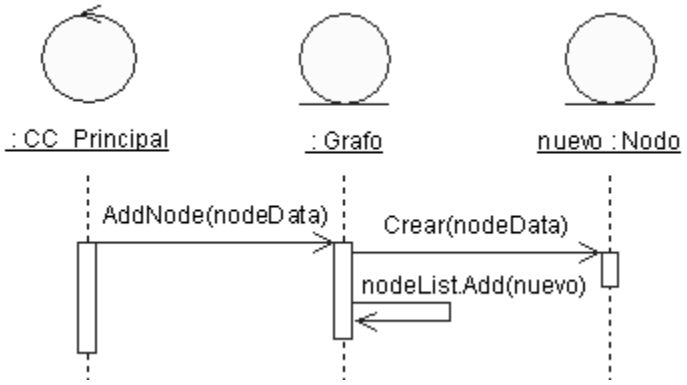


Fig. 32 Diagrama de secuencia del caso de uso incluido "Crear nodo".

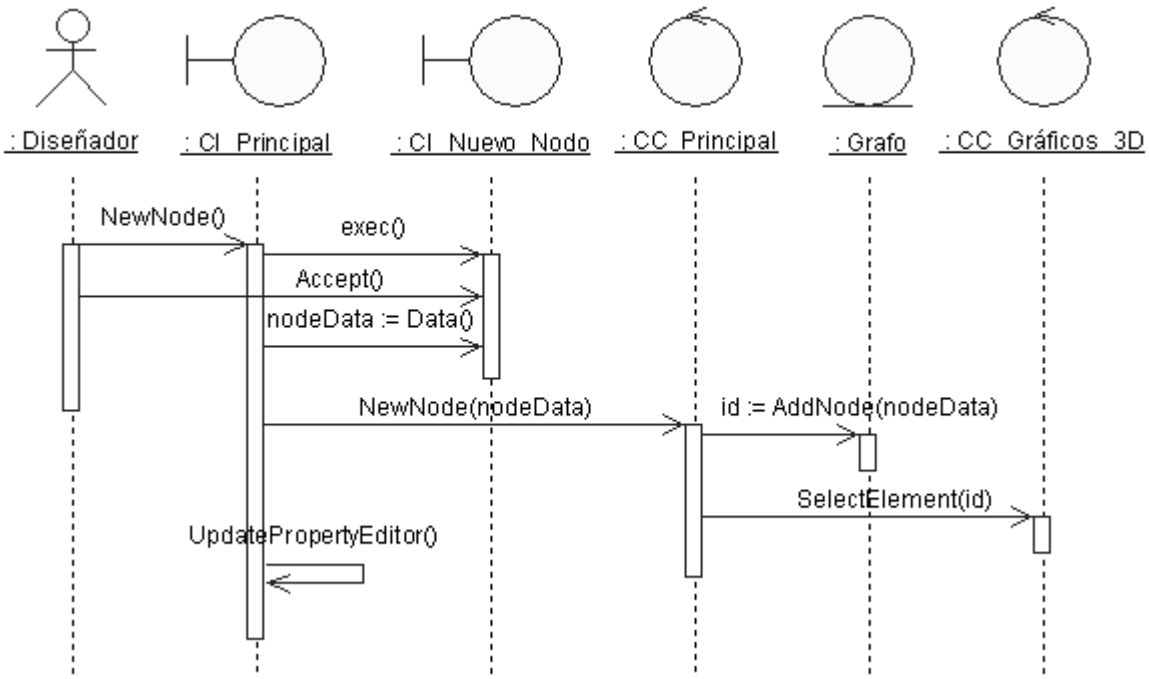


Fig. 33 Diagrama de secuencia del caso de uso "Crear nodo con asistente".

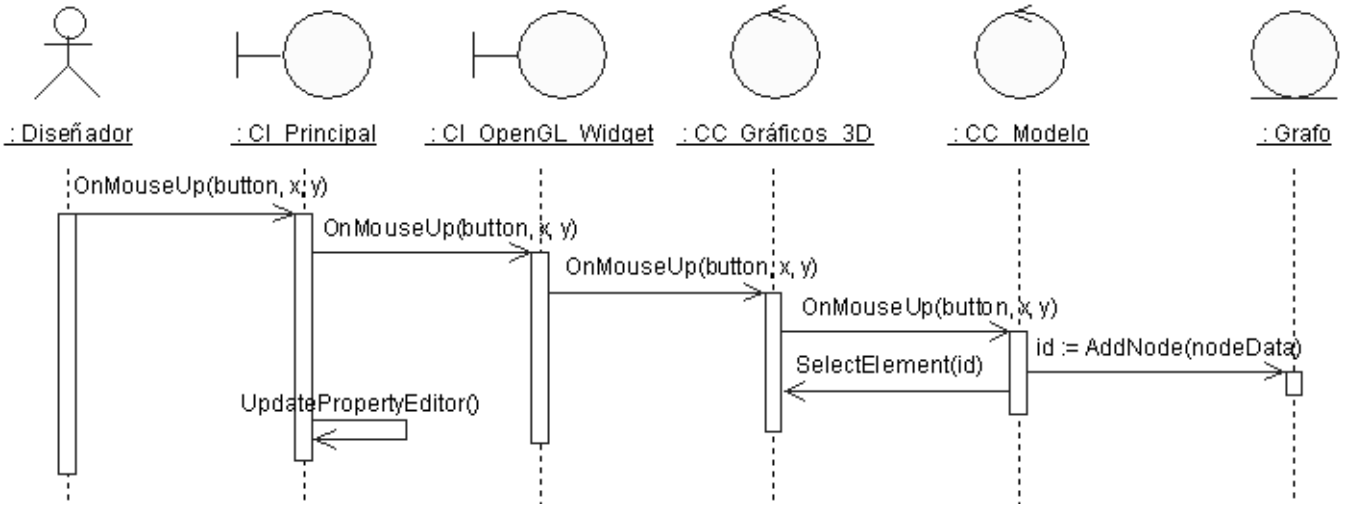


Fig. 34 Diagrama de secuencia del caso de uso "Crear nodo manualmente".

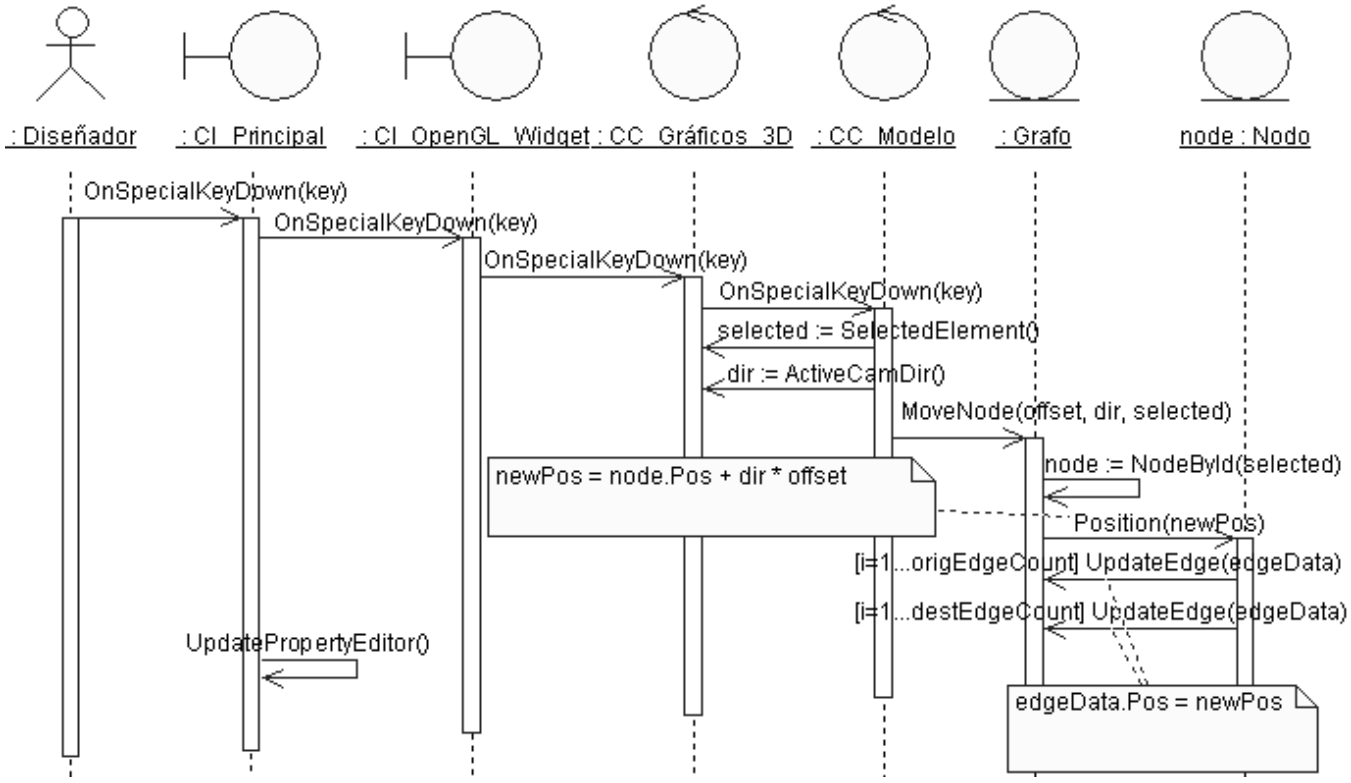


Fig. 35 Diagrama de secuencia del caso de uso "Desplazar nodo".

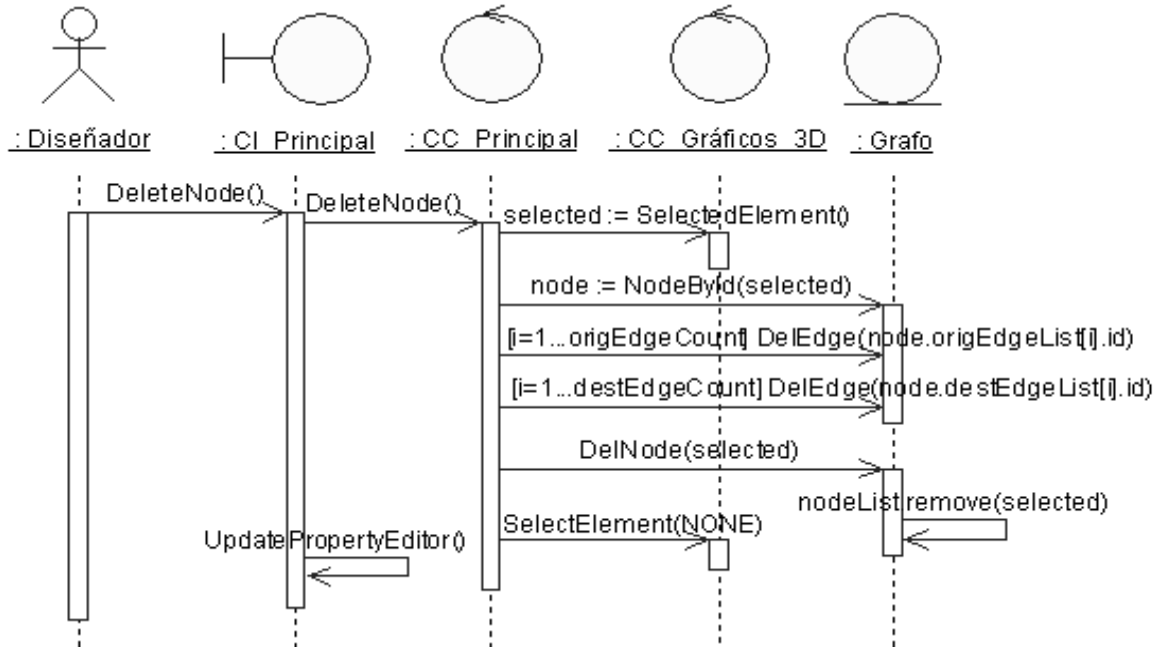


Fig. 36 Diagrama de secuencia del caso de uso "Eliminar nodo".

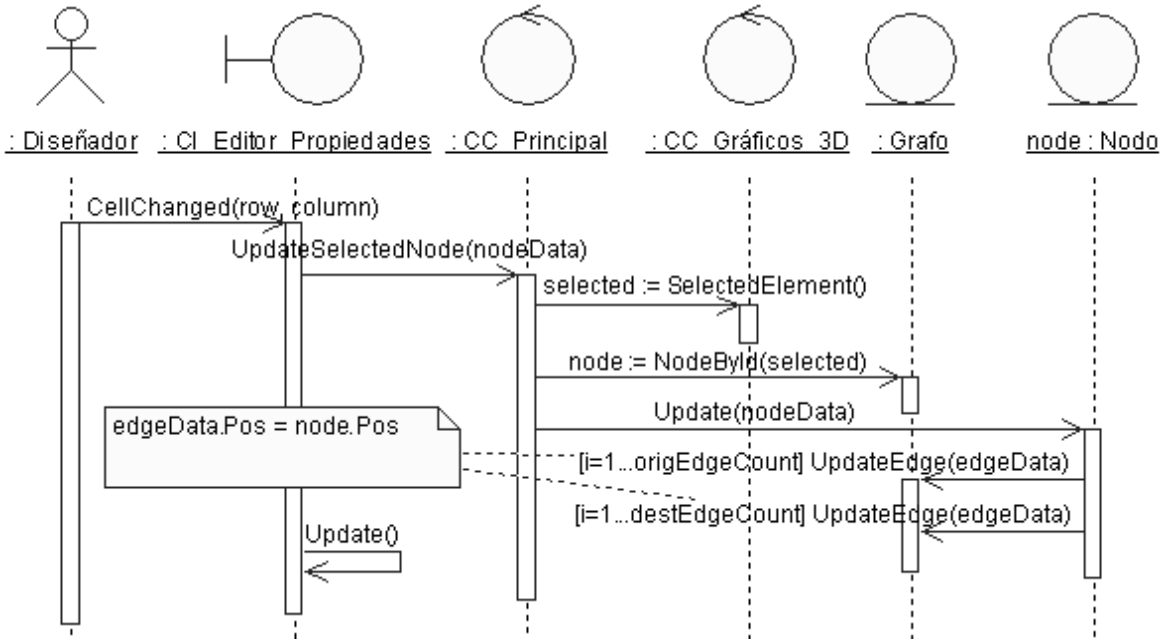


Fig. 37 Diagrama de secuencia del caso de uso "Modificar nodo".

Sub-paquete "Gestión de Aristas".

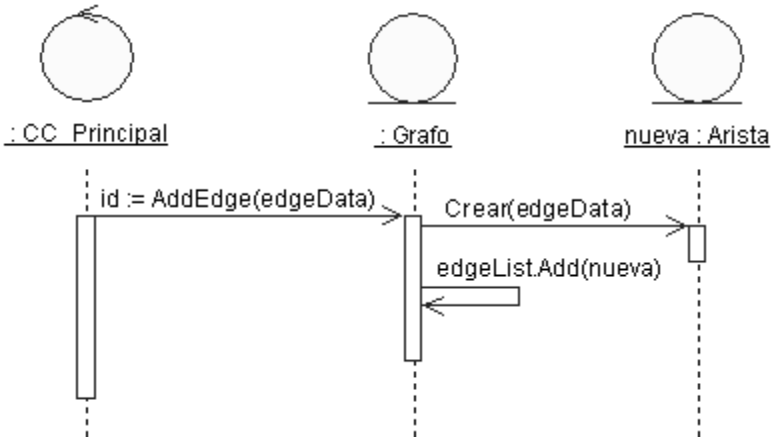


Fig. 38 Diagrama de secuencia del caso de uso incluido "Crear arista".

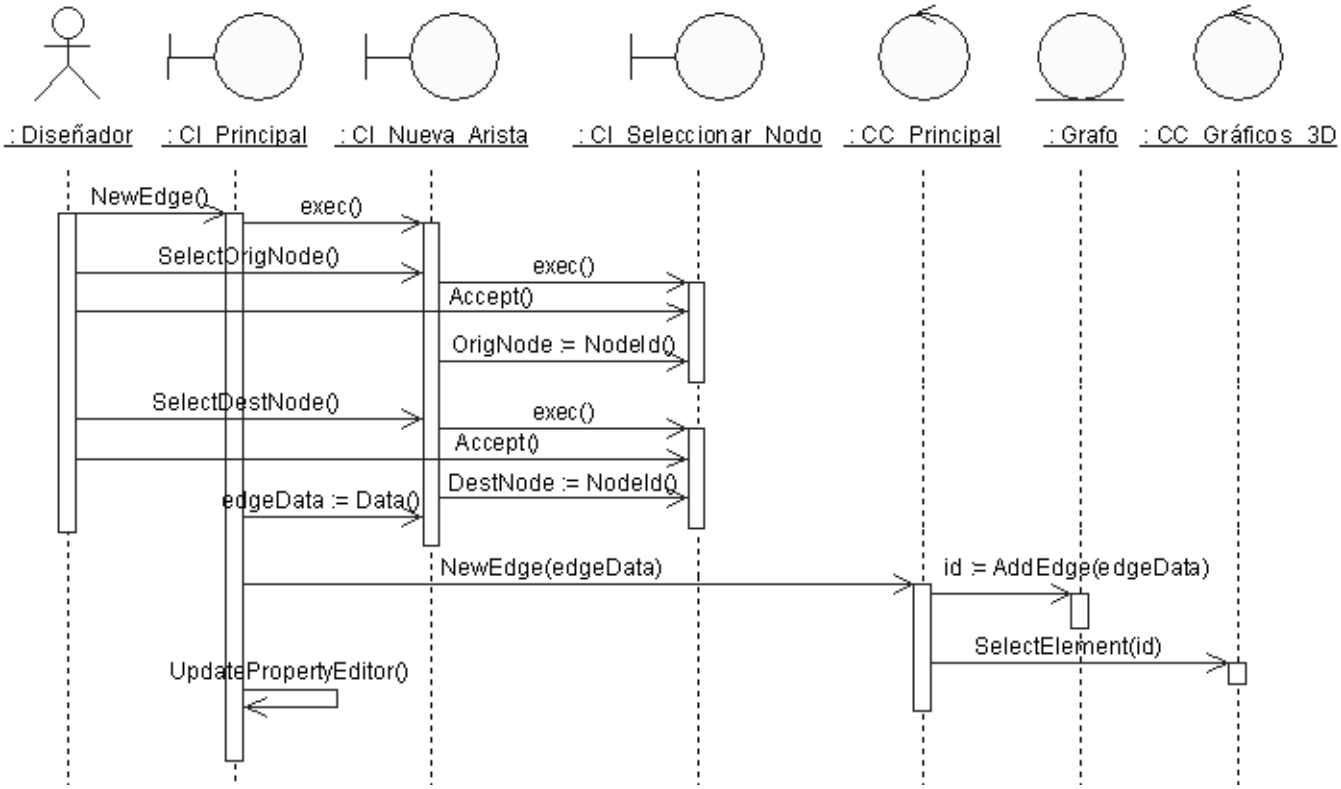


Fig. 39 Diagrama de secuencia del caso de uso "Crear arista con asistente".

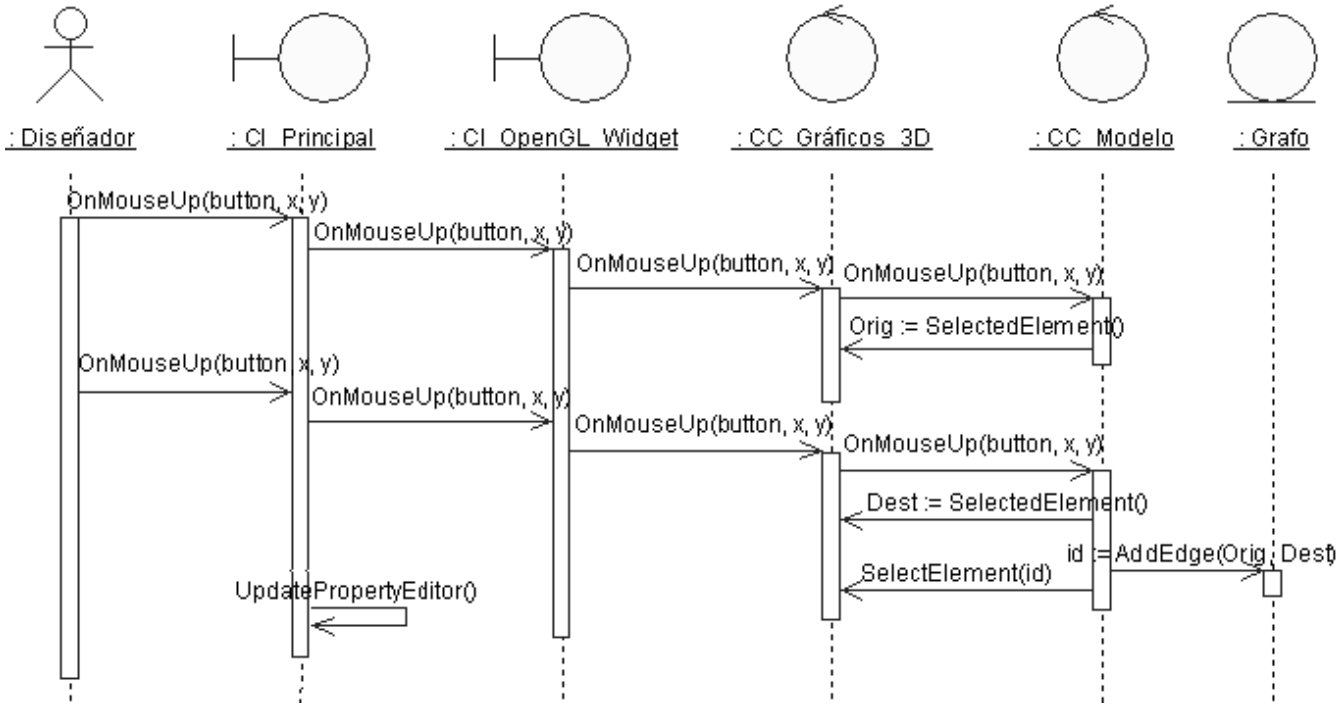


Fig. 40 Diagrama de secuencia del caso de uso "Crear arista manualmente".

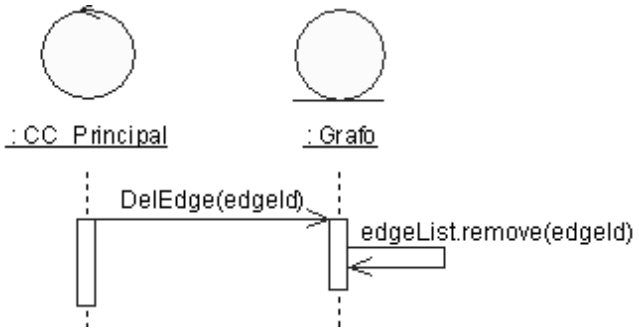


Fig. 41 Diagrama de secuencia del caso de uso incluido "Eliminar arista".

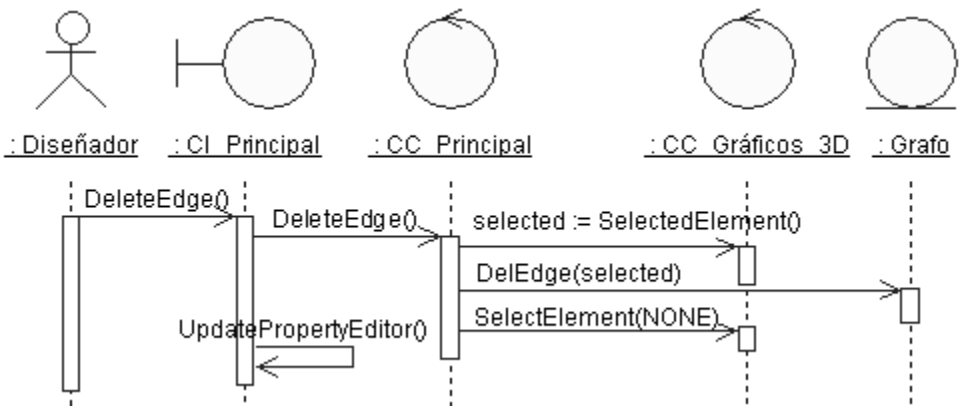


Fig. 42 Diagrama de secuencia del caso de uso "Eliminar arista manualmente".

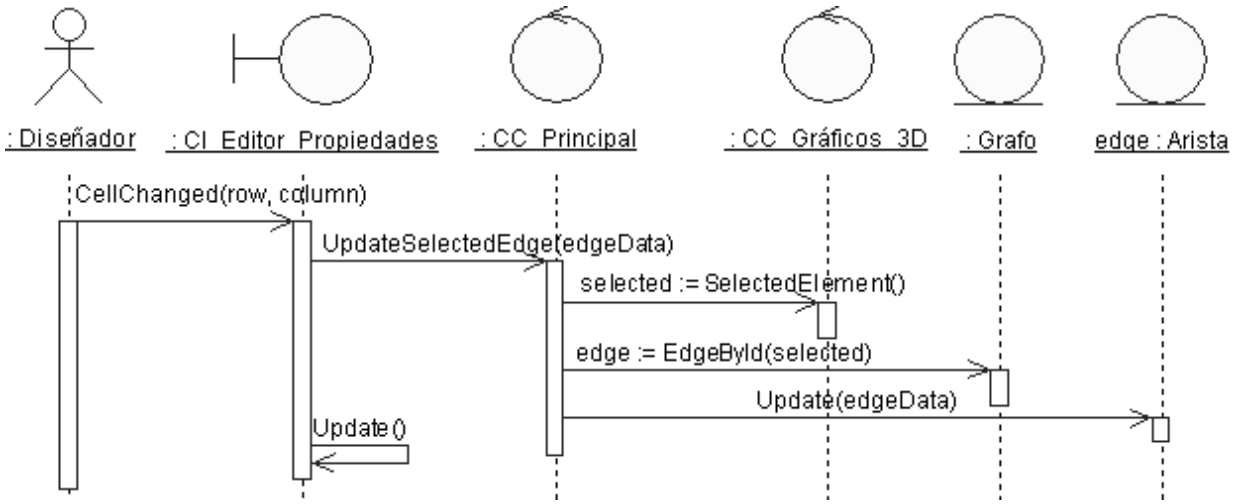


Fig. 43 Diagrama de secuencia del caso de uso "Modificar arista".

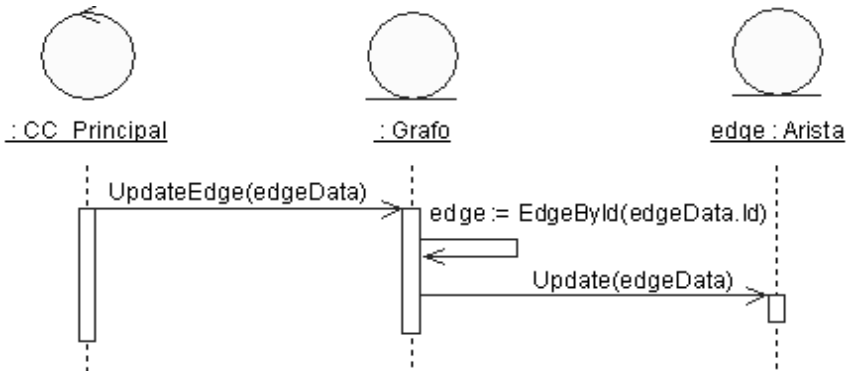


Fig. 44 Diagrama de secuencia del caso de uso incluido "Actualizar arista".

Sub-paquete "Gestión de vértices de control".

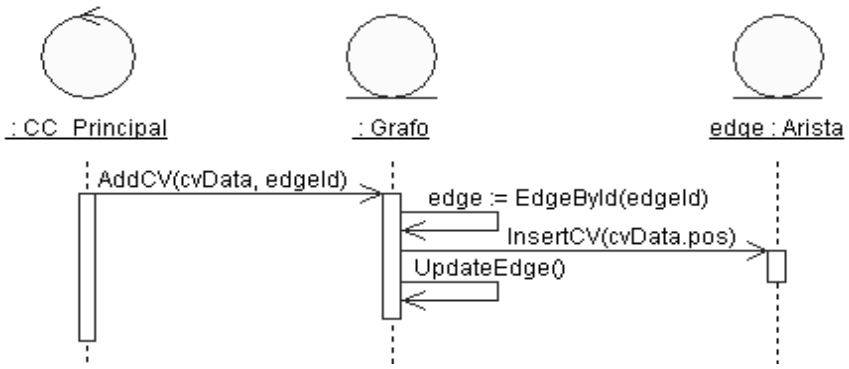


Fig. 45 Diagrama de secuencia del caso de uso incluido "Crear vértice de control".

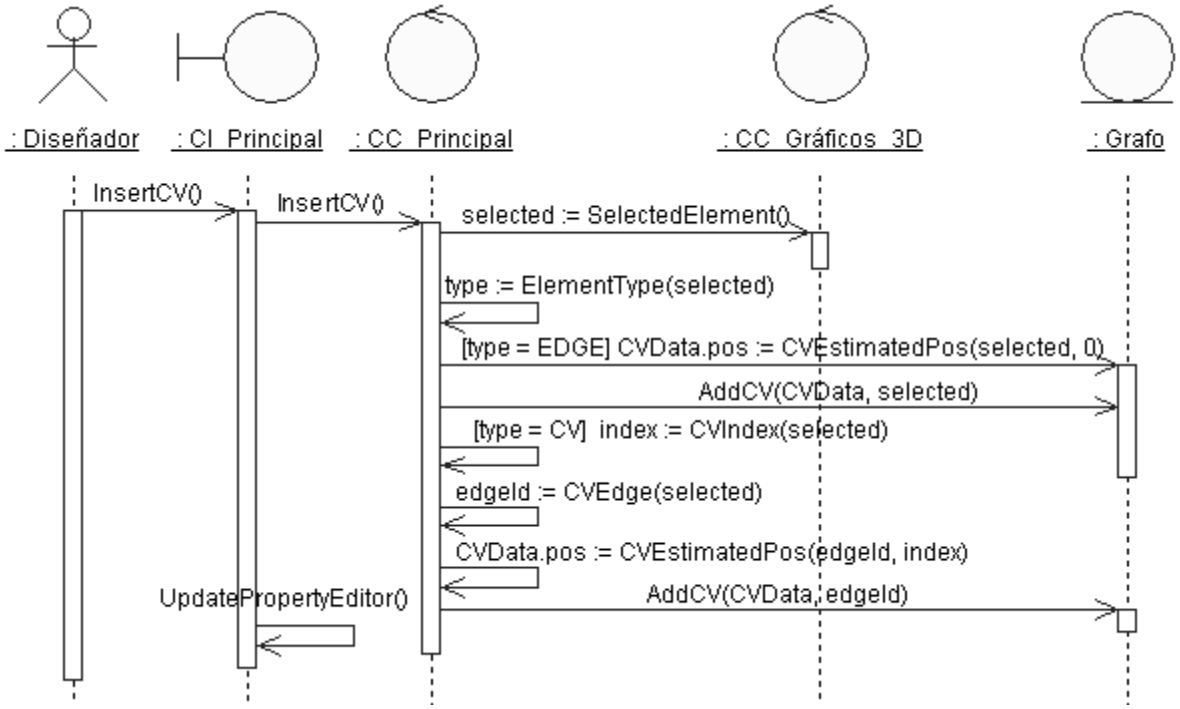


Fig. 46 Diagrama de secuencia del caso de uso "Crear vértice de control manualmente".

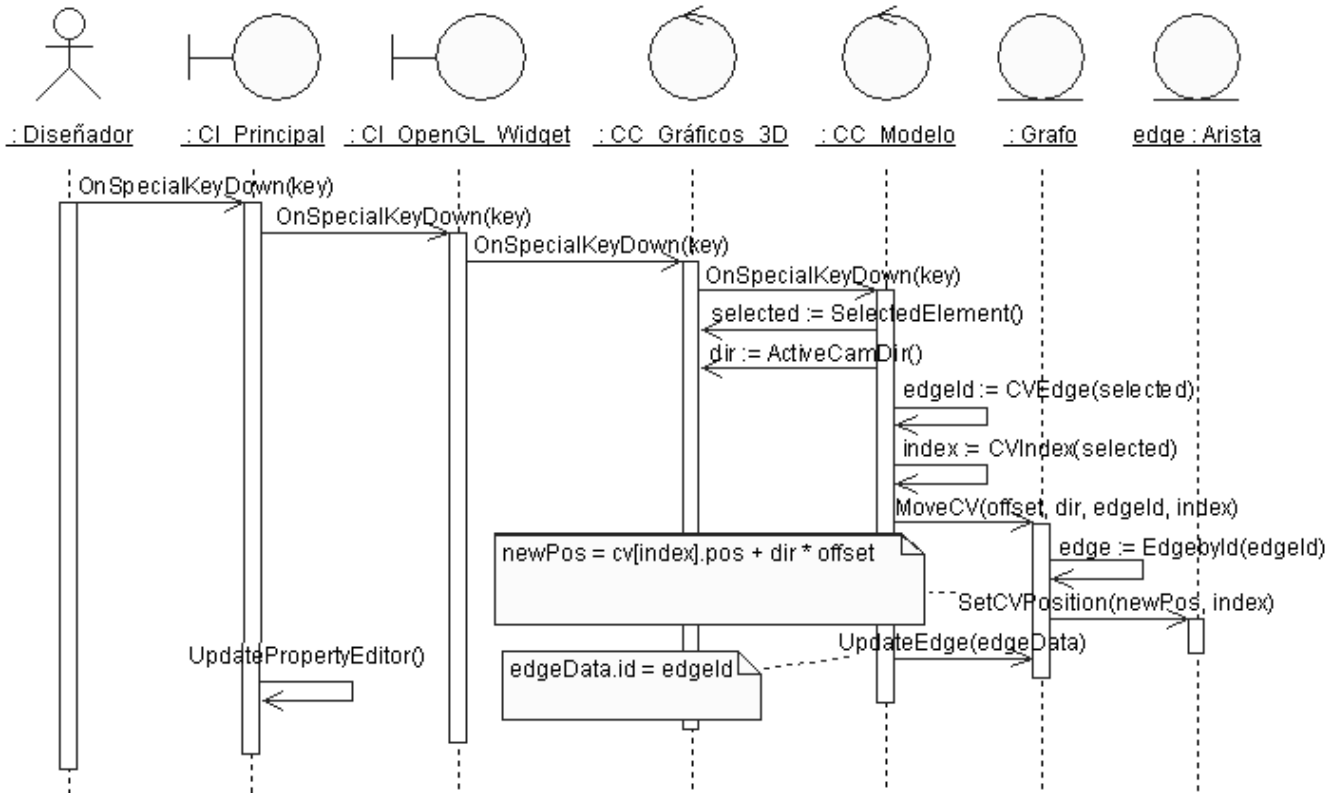


Fig. 47 Diagrama de secuencia del caso de uso "Desplazar vértice de control".

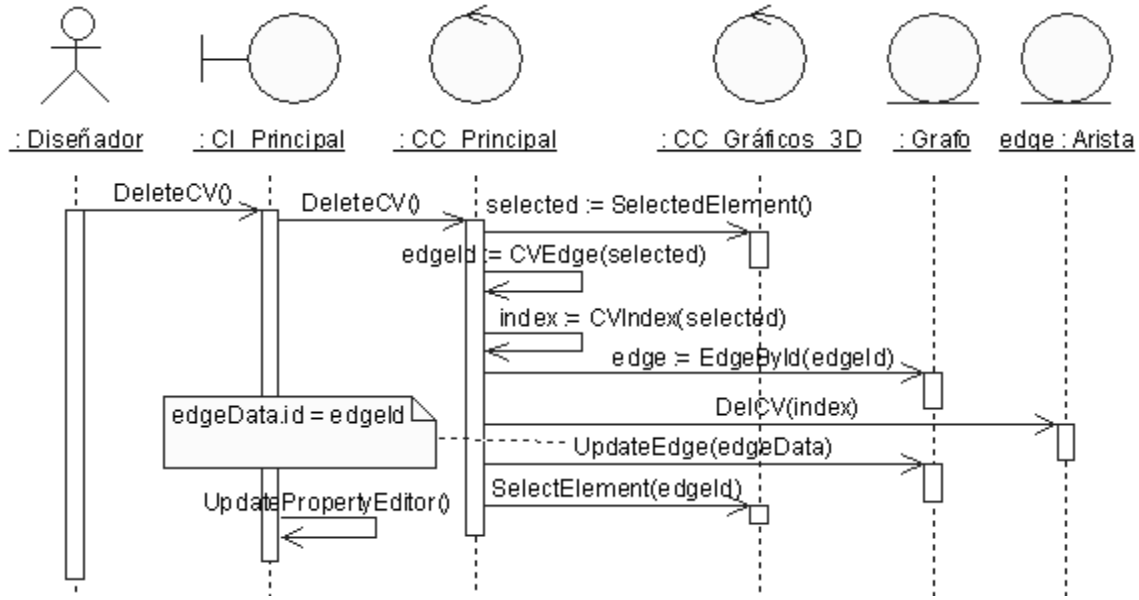


Fig. 48 Diagrama de secuencia del caso de uso "Eliminar vértice de control".

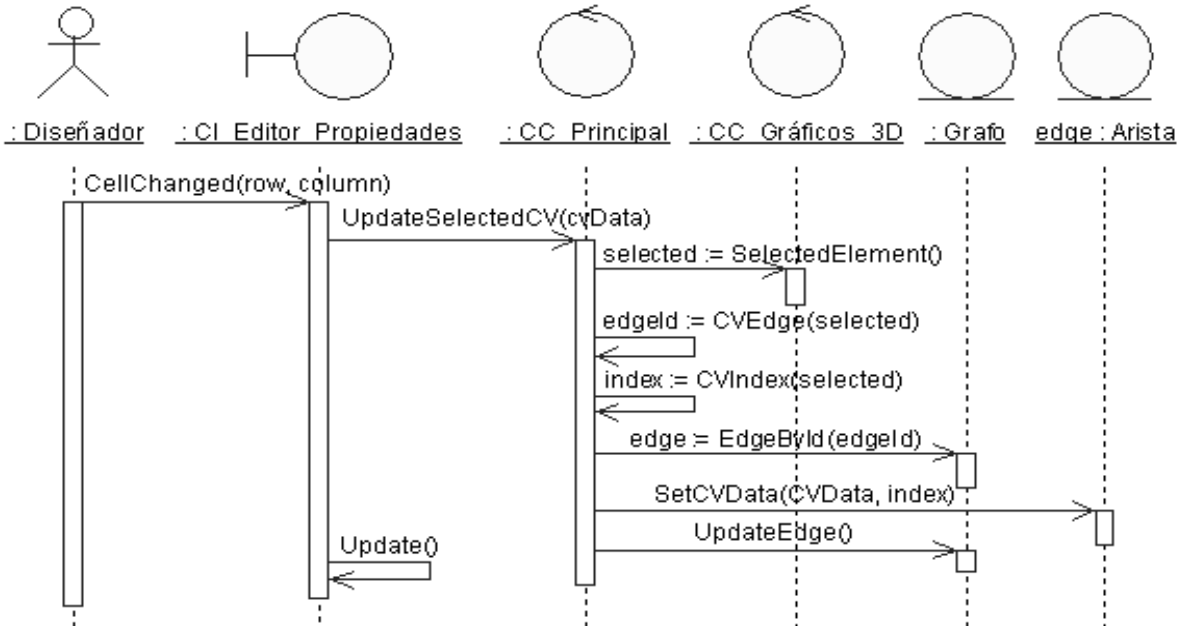


Fig. 49 Diagrama de secuencia del caso de uso "Modificar vértice de control".

Conclusiones

Al concluir este capítulo se tiene concebido detalladamente el diseño completo del sistema y la secuencia de pasos traducida a mensajes entre clases de los primeros casos de uso a desarrollar, con lo que se puede pasar a la etapa de implementación del proyecto.

CAPÍTULO 5: IMPLEMENTACIÓN Y PRUEBA

Introducción

Esta etapa del proyecto constituye el paso del diseño de clases a la creación de componentes físicos, que se traducen en ficheros .h y .cpp correspondientes a la implementación en C++. Además se elabora el diagrama de despliegue para el sistema.

5.1 Estándares de codificación.

A continuación se menciona la propuesta de estándar de codificación a utilizar para el desarrollo de la herramienta. El código seguirá el mismo estándar de la herramienta SceneToolkit, el cual está escrito en inglés por ser un idioma muy difundido en el mundo informático, respetando los estándares de codificación para C++. El conocimiento de los estándares de codificación reducirá perceptiblemente el riesgo de que los desarrolladores cometan errores; durante el desarrollo los estándares de codificación ayudan a los ingenieros a producir un código de alta calidad y aumentan considerablemente la capacidad de mantenimiento y reutilización del producto final a largo plazo.

Nombre de los ficheros:
Se nombrarán los ficheros .h y .cpp de la siguiente manera:
STKNameOfUnits.cpp

Constantes:
Las constantes se nombran con mayúsculas, utilizándose "_" para separar las palabras:
MY_CONST_ZERO = 0;

Tipos de datos:
Los tipos se nombrarán siguiendo el siguiente patrón:

Enumerados:
enum **E**MyEnum {**ME**_VALUE, **ME**_OTHER_VALUE};

Indicando con "**E**" que es de tipo enumerado. Nótese que las primeras letras de las constantes de enumerados son las iniciales del nombre del enumerado. Véase otro ejemplo:
enum **E**Node**T**ype{**NT**_GEOMETRYNODE,…};

Estructuras:
struct **S**MyStruct {…};

Indicando con "**S**" que es una estructura. Las variables miembros de la estructura se nombrarán igual que en las clases, leer más adelante.

Clases: class **C**ClassName;
Indicando con "**C**" que es una clase

Interfaces: **I**MyInterface
Indicando con "**I**" que es una interfaz.

Listas e iteradotes STD:
vector<> **T**Name**List**;
TNameList::iterator **T**Name**ListIter**;
map<> **T**Name**Map**;
TNameMap::iterator **T**Name**MapIter**;
multimap<> TNameMultiMap;
TNameMultiMap::iterator TNameMultiMapIter;

Declaración de variables:
Los nombres de las variables comenzarán con un identificador del tipo de dato al que correspondan, como se muestra a continuación. En el caso de que sean variables miembros de una clase, se le antepondrá el identificador "m_" (en minúscula), si son globales se les antepondrá la letra "g", y en caso de ser argumentos de algún método, se les antepondrá el prefijo "arg_".

Tipos simples:
bool bVarName;
int iName;

unsigned int uiName;
float fName;
char cName;
char* acName; // arreglo de caracteres
char* pcName; // puntero a un char
char** aacName; // bidimensional
char** apcName; // arreglo de punteros
bool m_bMemberVarName; //variable miembro
char gcGlobalVarName; //variable global, no se le antepone “”
short sName;

Instancias de tipos creados:
EMyEnumerated eName;
SMyStructure kName;
CClassName kObjectName;
CClassName* pkName; //puntero a objeto
CClassName* akName; //arreglo de objetos
CClassName* akName; // variable miembro de clase
IMyInterface* pIName; //puntero interfaces

Métodos:
En el caso de los métodos, se les antepondrá el identificador del tipo de dato de devolución, y en caso de no tenerlo (void), no se les antepondrá nada. Solamente los constructores y destructores omitirán el tipo de dato de retorno.

En el caso de los argumentos se les antepone el prefijo “arg_”.

Constructor y destructor:
CClassName (bool arg_bVarName, float& arg_fVarName);
~CClassName ();
Funciones:
bool bFunction1 (…);
int* piFunction2 (…);

CClassName* pkFunction3 (…);

Procedimientos:

void Procedure4 (…);

Métodos de acceso a miembros:

Los métodos de acceso a los miembros de las clases no se nombrarán “Gets” y “Sets”, sino como los demás métodos, pero con el nombre de la variable a la que se accede y sin “m_”:

```
int iMyVar; //variable
```

Obtención del valor:

```
int iMyVar()
{
    return iMyVar;
}
```

Establecimiento del valor:

```
void MyVar(char* arg_iMyVar)
{
    iMyVar = arg_iMyVar;
}
```

Obtención y establecimiento del valor:

```
int& iMyVar()
{
    return iMyVar;
}
```

5.2 Diagrama de despliegue.

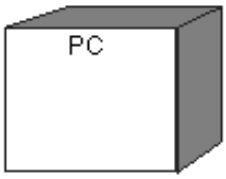


Fig. 50 Diagrama de despliegue.

El sistema se desplegará en una PC ya que no están involucradas las funciones de red de la STK.

5.3 Diagramas de componentes.

A continuación se muestran los diagramas de componentes de los distintos paquetes, estos diagramas muestran la distribución de las clases en ficheros de código fuente. Nótese la utilización de los estereotipos de Racional Rose para C++.

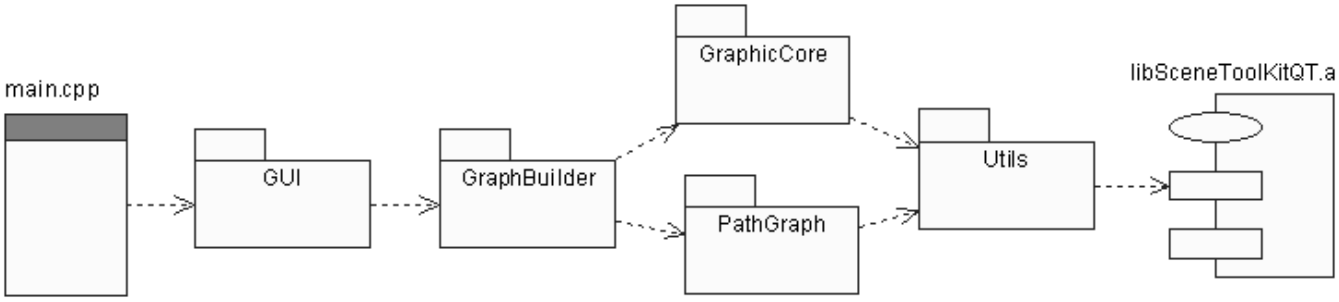


Fig. 51 Diagrama de paquetes de componentes.

En esta imagen se resalta la relación entre el componente main.cpp de tipo “Main Program” con los distintos paquetes. El paquete “libSceneToolkit.a” es la librería dinámica de SceneToolkit generada con el compilador GNU G++.

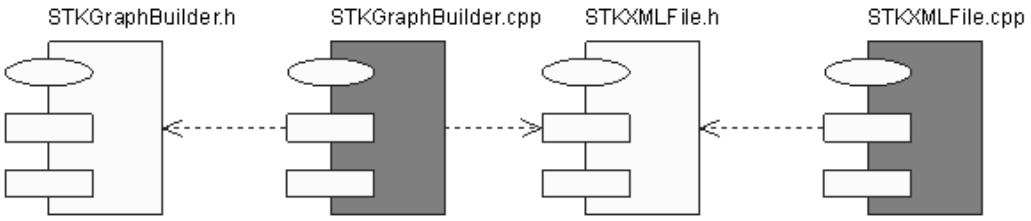


Fig. 52 Diagrama de componentes, paquete “Graph Builder”.

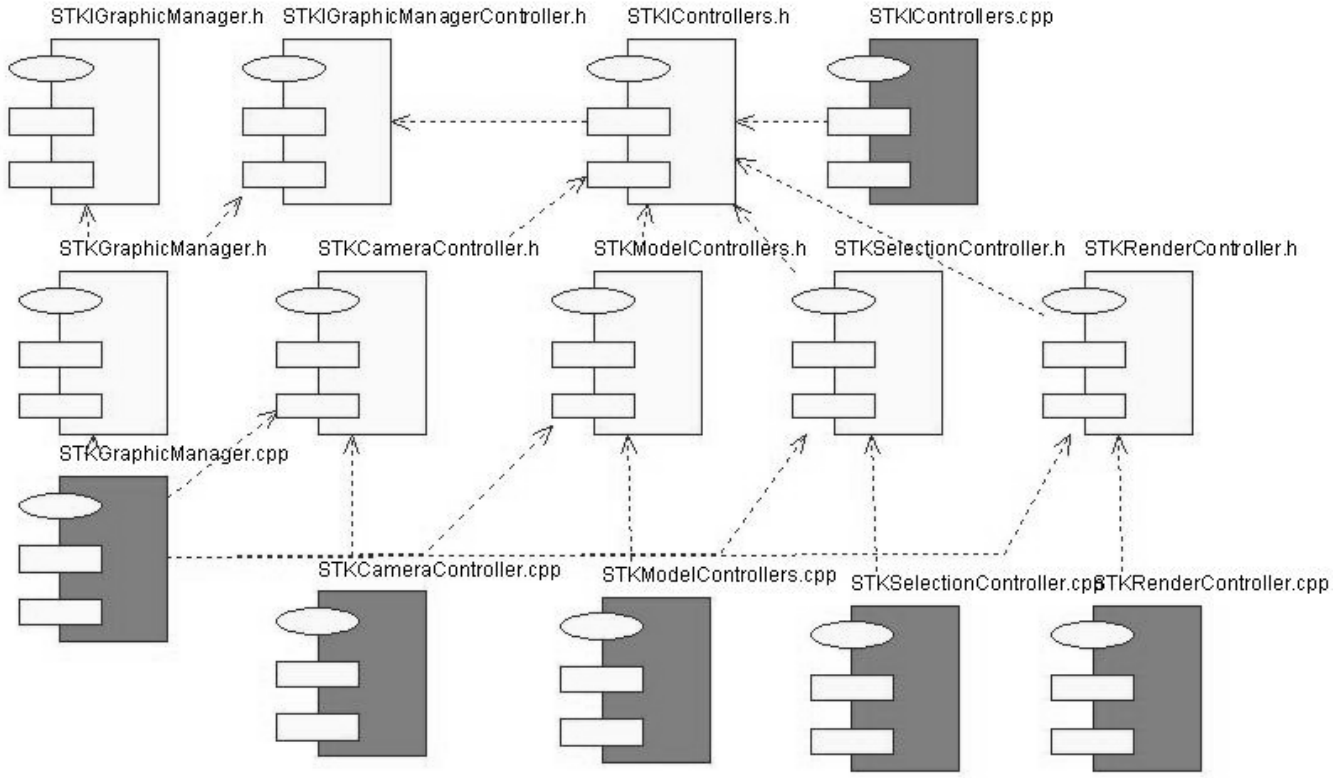


Fig. 53 Diagrama de componentes, paquete "Graphic Core".

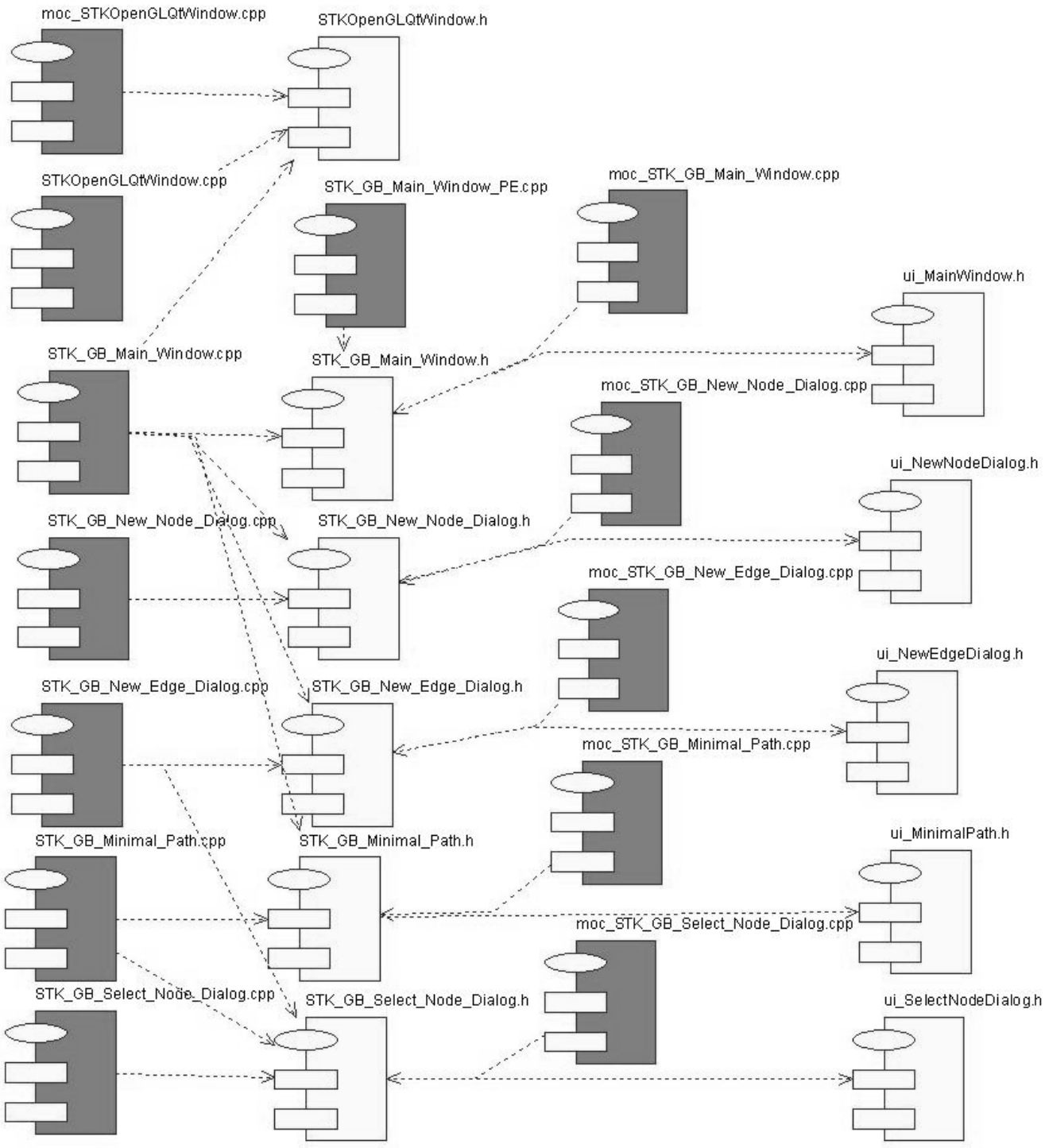


Fig. 54 Diagrama de componentes, paquete "GUI".

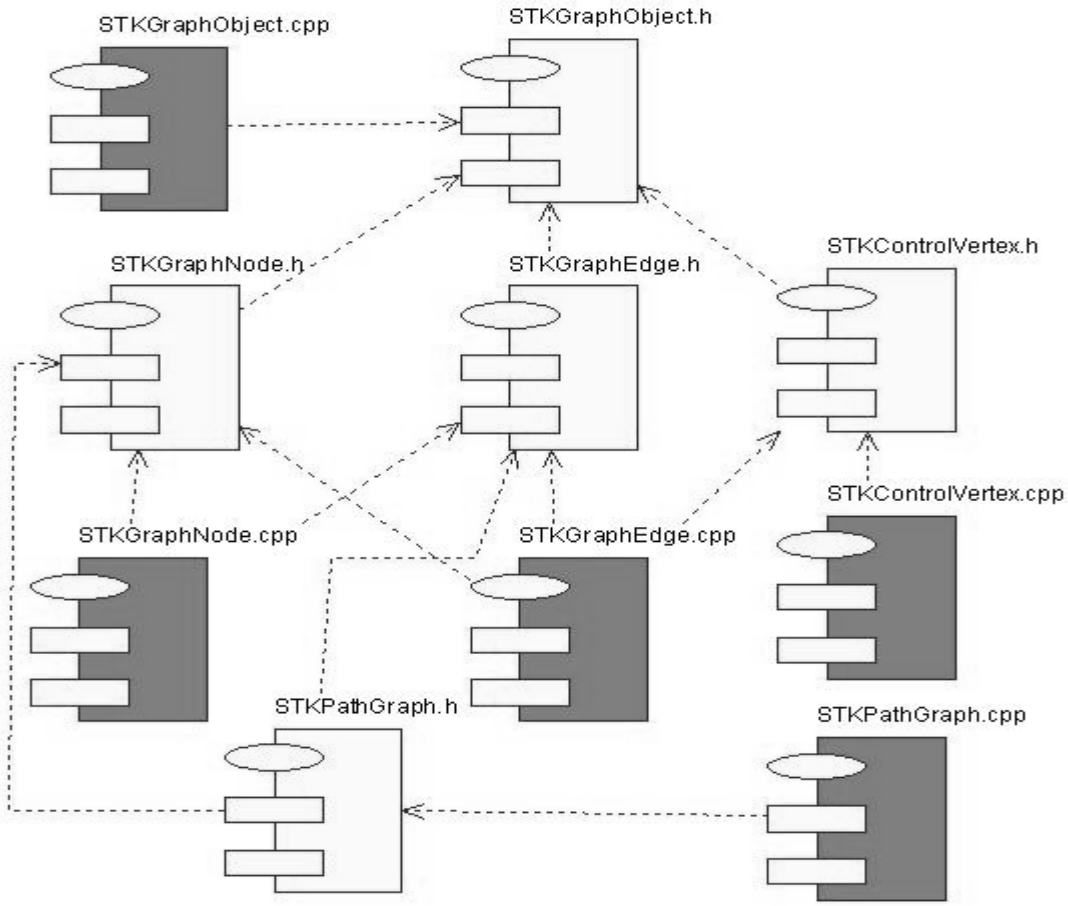


Fig. 55 Diagrama de componentes, paquete "Path Graph".

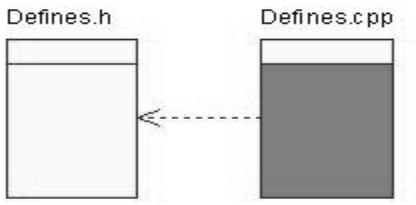


Fig. 56 Diagrama de componentes, paquete "Utils".

5.4 Descripción de los casos de usos de prueba.

En este epígrafe se recogen los casos de prueba utilizados para detectar mal funcionamientos del sistema. El tipo de pruebas utilizado es el de caja negra y entre otros aspectos se probó la respuesta del sistema ante valores extremos e insuficiencia de datos.

A continuación se especifica cada uno de los casos de prueba:

Tabla 27 Caso de prueba "Seleccionar mal camino de lectura o nombre de fichero 3DX".

Caso de uso	Cargar entorno.
Caso de prueba	Seleccionar mal camino de lectura o nombre de fichero 3DX.
Entrada	Se selecciona la opción "Cargar entorno".
Resultado	Se muestra un mensaje de aviso, no se importa ningún dato.
Condiciones	El fichero debe tener extensión 3DX.

Tabla 28 Caso de prueba "Seleccionar mal camino de lectura o nombre de fichero GraphML".

Caso de uso	Cargar grafo.
Caso de prueba	Seleccionar mal camino de lectura o nombre de fichero GraphML.
Entrada	Se selecciona la opción "Cargar grafo".
Resultado	Se muestra un mensaje de aviso, no se importa ningún dato.
Condiciones	El fichero debe tener extensión GrahpML.

Tabla 29 Caso de prueba "Seleccionar mal camino de escritura o nombre de fichero GraphML".

Caso de uso	Salvar grafo.
Caso de prueba	Seleccionar mal camino de escritura o nombre de fichero GraphML.
Entrada	Se selecciona la opción "Salvar grafo".
Resultado	Muestra un mensaje de aviso al usuario, no se guarda ningún dato.
Condiciones	El fichero debe tener la extensión GrahpML.

Tabla 30 Caso de prueba "Guardar fichero GraphML sin arista".

Caso de uso	Salvar grafo.
Caso de prueba	Guardar fichero GraphML sin arista.
Entrada	Se selecciona la opción "Salvar grafo".
Resultado	Muestra un mensaje de aviso al usuario, no se guarda ningún dato.
Condiciones	El grafo de caminos no debe tener aristas.

Tabla 31 Caso de prueba "Teclear descripción de nueva arista demasiado larga".

Caso de uso	Crear arista con asistente.

Caso de prueba	Teclear descripción de nueva arista demasiado larga.
Entrada	Se teclea en el campo descripción una cadena de más de 255 caracteres.
Resultado	Solamente acepta el número de caracteres establecido y crea la arista.
Condiciones	Mostrado formulario “Nueva arista”.

Tabla 32 Caso de prueba “Teclear nombre de nueva arista demasiado largo”.

Caso de uso	Crear arista con asistente.
Caso de prueba	Teclear nombre de nueva arista demasiado largo.
Entrada	Se teclea en el campo nombre una cadena de más de 31 caracteres.
Resultado	Solamente acepta el número de caracteres establecido y crea la arista.
Condiciones	Mostrado formulario “Nueva arista”.

Tabla 33 Caso de prueba “Entrar id de nodo origen de nueva arista inválido”.

Caso de uso	Crear arista con asistente.
Caso de prueba	Entrar id de nodo origen de nueva arista inválido.
Entrada	Se teclea en el campo nodo origen un id de un nodo que no exista.
Resultado	Muestra un mensaje de error y no se crea la arista.
Condiciones	Mostrado formulario “Nueva arista”.

Tabla 34 Caso de prueba “Entrar id de nodo destino de nueva arista inválido”.

Caso de uso	Crear arista con asistente.
Caso de prueba	Entrar id de nodo destino de nueva arista inválido.
Entrada	Se teclea en el campo nodo destino un id de un nodo que no exista.
Resultado	Muestra un mensaje de error y no se crea la arista.
Condiciones	Mostrado formulario “Nueva arista”.

Tabla 35 Caso de prueba “Entrar nivel de detalle de nueva arista inválido”.

Caso de uso	Crear arista con asistente.
Caso de prueba	Entrar nivel de detalle de nueva arista inválido.
Entrada	Se teclea en el campo nivel de detalle un valor que no sea válido, ej: “0”.
Resultado	Muestra un mensaje de error y no se crea la arista.
Condiciones	Mostrado formulario “Nueva arista”.

Tabla 36 Caso de prueba "Omitir campo de nueva arista".

Caso de uso	Crear arista con asistente.
Caso de prueba	Omitir campo de nueva arista.
Entrada	Se deja vacío alguno de los campos del formulario.
Resultado	Muestra un mensaje de error y no se crea la arista.
Condiciones	Mostrado formulario "Nueva arista".

Tabla 37 Caso de prueba "Teclear descripción de nuevo nodo demasiado larga".

Caso de uso	Crear nodo con asistente.
Caso de prueba	Teclear descripción de nuevo nodo demasiado larga.
Entrada	Se teclea en el campo descripción una cadena de más de 255 caracteres.
Resultado	Solamente acepta el número de caracteres establecido y crea el nodo.
Condiciones	Mostrado formulario "Nuevo nodo".

Tabla 38 Caso de prueba "Teclear nombre de nuevo nodo demasiado largo".

Caso de uso	Crear nodo con asistente.
Caso de prueba	Teclear nombre de nuevo nodo demasiado largo.
Entrada	Se teclea en el campo nombre una cadena de más de 31 caracteres.
Resultado	Solamente acepta el número de caracteres establecido y crea el nodo.
Condiciones	Mostrado formulario "Nuevo nodo".

Tabla 39 Caso de prueba "Omitir campo de nuevo nodo".

Caso de uso	Crear nodo con asistente.
Caso de prueba	Omitir campo de nuevo nodo.
Entrada	Se deja vacío alguno de los campos del formulario.
Resultado	Muestra un mensaje de error y no se crea el nodo.
Condiciones	Mostrado formulario "Nuevo nodo".

Tabla 40 Caso de prueba "Aceptar sin seleccionar nodo".

Caso de uso	Buscar nodo.
Caso de prueba	Aceptar sin seleccionar nodo.
Entrada	No se selecciona ningún nodo en la lista.

Resultado	Muestra un mensaje de advertencia.
Condiciones	Mostrado formulario “Seleccionar nodo”.

Tabla 41 Caso de prueba “Establecer nombre de arista demasiado largo”.

Caso de uso	Modificar arista.
Caso de prueba	Establecer nombre de arista demasiado largo.
Entrada	Se teclea en el campo nombre una cadena de más de 31 caracteres.
Resultado	Solamente acepta el número de caracteres establecido y actualiza la arista.
Condiciones	Arista seleccionada y sus propiedades mostradas en el editor de propiedades.

Tabla 42 Caso de prueba “Establecer descripción de arista demasiado larga”.

Caso de uso	Modificar arista.
Caso de prueba	Establecer descripción de arista demasiado larga.
Entrada	Se teclea en el campo descripción una cadena de más de 255 caracteres.
Resultado	Solamente acepta el número de caracteres establecido y actualiza la arista.
Condiciones	Arista seleccionada y sus propiedades mostradas en el editor de propiedades.

Tabla 43 Caso de prueba “Establecer nivel de detalle de arista inválido”.

Caso de uso	Modificar arista.
Caso de prueba	Establecer nivel de detalle de arista inválido.
Entrada	Se teclea en el campo nivel de detalle un valor que no sea válido, ej: “0”, valores no enteros, valores negativos o valores positivos mayores que 99.
Resultado	Repone el valor anterior a la modificación y no se actualiza la arista.
Condiciones	Arista seleccionada y sus propiedades mostradas en el editor de propiedades.

Tabla 44 Caso de prueba “Establecer nombre de nodo demasiado largo”.

Caso de uso	Modificar nodo.
Caso de prueba	Establecer nombre de nodo demasiado largo.
Entrada	Se teclea en el campo nombre una cadena de más de 31 caracteres.
Resultado	Solamente acepta el número de caracteres establecido y actualiza el nodo.
Condiciones	Nodo seleccionado y sus propiedades mostradas en el editor de propiedades.

Tabla 45 Caso de prueba "Establecer descripción de nodo demasiado larga".

Caso de uso	Modificar nodo.
Caso de prueba	Establecer descripción de nodo demasiado larga.
Entrada	Se teclea en el campo descripción una cadena de más de 255 caracteres.
Resultado	Solamente acepta el número de caracteres establecido y actualiza el nodo.
Condiciones	Nodo seleccionado y sus propiedades mostradas en el editor de propiedades.

Tabla 46 Caso de prueba "Establecer componente de posición de nodo fuera de rango".

Caso de uso	Modificar nodo.
Caso de prueba	Establecer componente de posición de nodo fuera de rango.
Entrada	Se teclea en uno de los campos posición (X, Y, Z) un valor menor que -9999999 o mayor que 99999999.
Resultado	No acepta el valor entrado, repone el valor anterior y no actualiza el nodo.
Condiciones	Nodo seleccionado y sus propiedades mostradas en el editor de propiedades.

Tabla 47 Caso de prueba "Establecer grado de libertad de nodo fuera de rango".

Caso de uso	Modificar nodo.
Caso de prueba	Establecer grado de libertad de nodo fuera de rango.
Entrada	Se teclea en el campo grado de libertad un valor menor que 0 o mayor que 999.
Resultado	No acepta el valor entrado, repone el valor anterior y no actualiza el nodo.
Condiciones	Nodo seleccionado y sus propiedades mostradas en el editor de propiedades.

Tabla 48 Caso de prueba "Establecer componente de posición de vértice de control fuera de rango".

Caso de uso	Modificar vértice de control.
Caso de prueba	Establecer componente de posición de vértice de control fuera de rango.
Entrada	Se teclea en uno de los campos posición (X, Y, Z) un valor menor que -9999999 o mayor que 99999999.
Resultado	No acepta el valor entrado, repone el valor anterior y no actualiza el vértice de control.
Condiciones	Vértice de control seleccionado y sus propiedades mostradas en el editor de propiedades.

Tabla 49 Caso de prueba "Establecer componente de peso de vértice de control fuera de rango".

Caso de uso	Modificar vértice de control.
Caso de prueba	Establecer componente de peso de vértice de control fuera de rango.
Entrada	Se teclea en uno de los campos peso (X, Y, Z) un valor menor que 1 o mayor que 99999.
Resultado	No acepta el valor entrado, repone el valor anterior y no actualiza el nodo.
Condiciones	Vértice de control seleccionado y sus propiedades mostradas en el editor de propiedades.

Tabla 50 Caso de prueba "Consultar camino óptimo sin haber calculado los caminos óptimos".

Caso de uso	Consultar camino óptimo.
Caso de prueba	Consultar camino óptimo sin haber calculado los caminos óptimos.
Entrada	Se selecciona la opción "Consultar caminos óptimos".
Resultado	Se muestra un mensaje al usuario preguntando si desea calcular los caminos óptimos.
Condiciones	Caminos óptimos no calculados.

Conclusiones

En este momento se encuentra todo preparado para pasar a la etapa de programación de los casos de uso desarrollados en el primer ciclo. Como posibilidad adicional que brinda la herramienta Rational Rose, es posible generar el código fuente de los componentes.

CONCLUSIONES

La posibilidad de disponer de herramientas para la creación y edición de caminos tridimensionales es vital para los motores gráficos de hoy día. Tanto es así, que los principales y más famosos de estos sistemas cuentan con algún tipo de facilidad de edición de curvas. En este trabajo, en cumplimiento con el objetivo inicialmente planteado, se construyó un sistema innovador que se adapta perfectamente a las necesidades requeridas por el motor gráfico SceneToolkit y brinda la posibilidad de crear, modificar y deformar visualmente los caminos del entorno, mediante un juego de herramientas intuitivas y fáciles de utilizar; además permite la aplicación de cálculos de camino mínimo entre nodos del grafo de caminos y guardar toda la información en fichero.

Para el cumplimiento de los objetivos de este proyecto, en concordancia con las exigencias hechas por el cliente, se requirió primeramente hacer un estudio de varios temas, tales como: planificación de movimientos, algoritmos de recorrido óptimo sobre grafos, formatos de fichero para almacenar información de grafos, técnicas de aproximación e interpolación, y principales características de la biblioteca SceneToolkit y del framework multiplataforma QT. A partir de esta investigación se proponen las características técnicas de la solución. Luego se realizó la captura de los requisitos funcionales y de los no funcionales, y la agrupación de los primeros en casos de uso del sistema. A continuación se procedió a las etapas de análisis, diseño e implementación utilizando los artefactos de RUP, donde surgieron y maduraron las clases, se “realizaron” los casos de uso y se creó el diagrama de componentes.

Se quiere destacar lo novedoso de varias de las decisiones más importantes tomadas en este proyecto, como son: la posibilidad de portar la herramienta a otros sistemas operativos debido a la utilización de funciones de la STL de C++, de la biblioteca de clases QT y del motor gráfico SceneToolkit, todos multiplataforma. Debido a la flexible arquitectura de la herramienta, el diseño es adaptable a posibles cambios, lo que posibilita que se puedan agregar nuevas funcionalidades en versiones posteriores sin hacer grandes modificaciones, en vistas a confeccionar un producto cada vez más completo y eficaz.

RECOMENDACIONES

Este trabajo se realizó con el objetivo de incorporar las técnicas de pathfinding y planificación de movimientos al motor gráfico SceneToolkit. Se recomienda continuar profundizando en estas ramas, ya que proporcionan más dinámica y realismo a las simulaciones en entornos virtuales. Como trabajos futuros se proponen las siguientes mejoras a la herramienta desarrollada:

- Incorporarle un inspector de objetos para mejorar la gestión de elementos del grafo de caminos por parte del usuario.
- Incorporarle funcionalidades para configurar distintas preferencias del usuario, por ejemplo: color de los distintos elementos (seleccionados o no), tamaño de los nodos y vértices de control, anchura y alisamiento (smooth) de las aristas, velocidades de traslación y rotación de la cámara. Además del almacenamiento de esta información en un fichero de configuración.
- Incorporarle soporte para más idiomas mediante las bondades que brinda QT Linguist.
- Agregarle más propiedades a los distintos elementos, de tal manera que permitan representar un conjunto más amplio de situaciones reales, como por ejemplo, el valor del peralte o inclinación transversal, que puede ser diferente para ambos lados de las carreteras.
- Incorporarle funcionalidades para hacer simulaciones, teniendo en cuenta el tipo de los caminos y de los elementos que los recorrerán.

En ese sentido, para llevar a cabo la última recomendación es necesario hacerle a la SceneToolkit las siguientes mejoras:

- Agregarle distintos controladores de caminos para los elementos que recorren los distintos tipos de arista definidos en este trabajo.
- Definirle la posibilidad de que los controladores de agentes terrestres también realicen seguimiento del terreno.
- Implementarle un módulo de inteligencia artificial o de pathfinding que haga uso de la información generada por la herramienta desarrollada.

BIBLIOGRAFÍA CITADA:

1. **CADAZZ.** CAD software - history of CAD CAM. [Online] 2004. [Cited: febrero 20, 2008.] http://www.cadazz.com/cad-software-history.htm.

2. **Gerald, Farin E.** *A history of curves and surfaces in CAGD.* 2001. p. 516.

3. **McNeel, Robert.** OpenNurbs initiative. [Online] 2007. [Cited: febrero 20, 2008.] http://www.opennurbs.org/.

4. **Software, nPower.** Power NURBS Pro. [Online] 2008. [Cited: febrero 22, 2008.] http://www.npowersoftware.com/.

5. **LaValle, Steven M**. Motion strategy library. [Online] 2008. [Cited: febrero 25, 2008.] http://msl.cs.uiuc.edu/msl/.

6. **Motion, Kineo Computer Aided**. Automatic motion and path planning software development kit. [Online] 2008. [Cited: febrero 25, 2008.] http://www.kineocam.com/kineoworks-library.php.

7. **PathEngine.** The PathEngine SDK. [Online] 2008. [Cited: febrero 27, 2008.] http://www.pathengine.com/contact.php.

8. **E., Aimée Vargas, Lien, Jyh-Ming and Amato, Nancy M.** *Vizmo++: A visualization, authoring, and educational tool for motion planning.* 2005. p. 12.

9. **Collazo, Mariela Nogueira and Madrigal, Omar Correa.** *Algoritmos para la manipulación eficiente de objetos dinámicos en escenas virtuales urbanas.* Universidad de las Ciencias Informáticas. La Habana : s.n., 2007. p. 112.

10. **LaValle, Steven M.** *Planning algorithms.* s.l. : Cambridge University Press, 2006. p. 842. 0-521-86205-1.

11. **Benkmann, Matthias S.** Motion planning using random networks. [Online] 2001. [Cited: marzo 4, 2008.] http://www.winterdrache.de/freeware/motionplan/index.html.

12. **M. S. Branicky, S. M. LaValle, K. Olson, L. Yang.** *Deterministic vs. probabilistic roadmaps***.** 2002. submitted to the IEEE Transactions on Robotics and Automation.

13. **Puebla, Benemérita Universidad Autónoma de.** *Teoría de grafos.* 2003. p. 17.

14. **Russell, Stuart and Norvig, Peter.** *Artificial intelligence, a modern approach.* 2nd. s.l. : Prentice Hall, 2002. p. 1132.

15. **Pearl, Judea.** *Heuristics: intelligent search strategies for computer problem solving.* s.l. : Addison Wesley, 1984.

16. **Dechter, Rina and Pearl, Judea.** *Generalized Best-first search strategies and the optimality of A*.* 1985. Journal of the ACM (JACM). Vol. 32, pp. 505 - 536. 0004-5411 .

17. **Torres, Juan Carlos.** Diseño de curvas y superficies. *Diseño asistido por ordenador. 4º Curso Ingeniería Informática.* 2003, p. 22.

18. **J., Fernández Cañavate F., Pedro, Company Calleja and P., Martí Montrull.** *Aplicación de las curvas CAGD al diseño óptimo de forma.* 1996. p. 12.

19. **Rondo, Ing. José Manuel Saavedra.** *Computación gráfica tridimensional. Curvas y superficies.* 2006.

20. **UMA, Universidad de Málaga.** *Sistemas avanzados de diseño industrial.* 2005. p. 14.

21. **Alberich, Jordi.** *Las flores de Bézier. Elasticidad e inestabilidad en el grafismo digital interactivo.* 2004. p. 12. 1695-5951.

22. **Ribó, Ramon.** *Descripción matemática de las NURBS.* 2003. p. 34.

23. **B.A., Barsky.** *Computer graphics and geometric modelling using Beta-Splines.* s.l. : Springer-Verlag, 1988. p. 64.

24. **Himsolt, Michael.** GML - graph modelling language. [Online] 1997. [Cited: marzo 15, 2008.] http://infosun.fmi.unipassau.de/Graphlet/GML/.

25. **W3C.** Extensible Markup Language (XML) 1.0. [Online] 1998. [Cited: marzo 20, 2008.] http://www.w3.org/TR/REC-xml.

26. **RPI.** XGMML (eXtensible Graph Markup and Modeling language). [Online] 2000. [Cited: marzo 30, 2008.] http://www.cs.rpi.edu/~puninj/XGMML/draft-xgmml-20010628.html .

27. **I. Herman, M.S. Marshall.** *GraphXML - An XML based graph interchange format.* 2000. p. 24. 1386-3681.

28. **Richard C. Holt, Andy Schürr, Susan Elliott Sim, Andreas Winter.** GXL: A graph-based standard exchange format for reengineering. [Online] 2002. [Cited: abril 2, 2008.] http://www.gupro.de/GXL.

29. **W3C.** GraphML Primer. [Online] 2001. [Cited: abril 5, 2008.] http://graphml.graphdrawing.org/primer/graphml-primer.html.

30. **Trolltech.** Qt: cross-platform rich client development framework. [Online] 2008. [Cited: abril 8, 2008.] http://trolltech.com/products/qt.

31. **Medina, Yasmany Cubela and Palau, Leonardo Nieblas.** *Interfaz Visual para la configuración de Entornos Virtuales desarrollados con la Herramienta SceneToolkit.* Universidad de las Ciencias Infrmáticas. La Habana : s.n., 2008. p. 137.

BIBLIOGRAFÍA CONSULTADA:

1. **Artz, B.E.** *An analytical road segment terrain database for driving simulation.* 1995. pág. 84. 2-87717-051-9.

2. **G., Greiner y H.P., Seidel.** *Automatic modelling of smooth spline surfaces.* 1997. pág. 35.

3. **Andersson, Fredrik.** *Bezier and B-Spline technology.* 2003. pág. 58.

4. **Bourke, Paul.** *Bezier surface in 3D.* 2006. pág. 79.

5. **López, Fernando Jiménez y Román, Yanoski Rogelio Camacho.** *Biblioteca gráfica para sistemas de realidad virtual.* 2004.

6. **Josuttis, Nicolai M.** *The C++ Standard Library: A tutorial and reference.* s.l. : Addison Wesley Longman, 1999. pág. 640. 0-201-37926-0.

7. **V.B, Anand.** *Computer graphics and geometric modelling for engineers.* s.l. : John Wiley & Sons, 1993. pág. 456.

8. **J.D., Foley, y otros.** *Computer graphics. Theory and practice.* s.l. : Addison-Wesley, 1992.

9. **León, Mauricio López.** *Curvas de Bézier.* 2004. pág. 7.

10. **Parejo, José Cortés y Valle, Juan Manuel Cordero.** *Curvas y superficies para modelado geométrico.* 2002. pág. 464.

11. **Farin, Gerald E.** *Curves and surfaces for computer aided geometric design. A practical Guide.* s.l. : Academic Press, 1988. pág. 47.

12. **Toumazet, J.P.Colinot y J.J.** *Evariste: genesis of a road databases editor.* 1997. 2-87717-063-2.

13. **L., Piegl.** *Fundamental developments of Computer-Aided Geometric Design.* s.l. : Academic Press, 1993.

14. **Diestel, Reinhard.** *Graph theory.* 3. New York : s.n., 2005. pág. 408.

15. **Bayarri, S.** *An instructional aid system for driving schools based on visual simulation.* 1998. pág. 63.

16. **W., Piegl L. y Tiller.** *The NURBS book.* 2nd. New York : Springer-Verlag, 1996. pág. 35.

17. **Prat, Vincent.** *NURBS curves and surfaces tutorial.* 2001. pág. 13.

18. **Virtual, Proyecto Herramientas de Desarrollo para Sistemas de Realidad.** *SceneToolKit: herramienta básica para desarrollo de sistemas de realidad virtual v2.3.* 2007. pág. 252.

19. **Bayarri, S.** *Segmented database and dynamic management algorithms for combined driving simulation.* 1996. pág. 66.

20. **Macri, Dean.** *Using NURBS surfaces in real-time applications.* 2000. pág. 15.

21. **Fernández., S. Bayarri y M.** *Virtual reality in driving simulation.* 1997. pág. 90.

GLOSARIO

API: conjunto de funciones y procedimientos que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.

Blender: programa multiplataforma, dedicado especialmente al modelado y creación de gráficos tridimensionales.

Callback: técnica en que aplicaciones clientes pasan "punteros a funciones" a las aplicaciones servidoras para ser notificadas acerca de algún evento.

DirectX: es una colección de APIs creadas para facilitar las complejas tareas relacionadas con multimedia, especialmente programación de juegos y vídeo en la plataforma Microsoft Windows.

Engine: componente principal de un video juego u otra aplicación interactiva con gráficos en tiempo real.

Entornos virtuales o mundos virtuales: se trata de la simulación de mundos o entornos, denominados virtuales, en los que el hombre interacciona con la máquina en entornos artificiales semejantes a la vida real.

Escalabilidad: es la propiedad deseable de un sistema que indica su habilidad de estar preparado para hacerse más grande sin perder calidad en los servicios ofrecidos. Capacidad del sistema informático de cambiar su tamaño o configuración para adaptarse a las circunstancias cambiantes.

FPS: (frames per second) cantidad de imágenes que renderiza un motor gráfico en un segundo.

Framework: estructura de soporte definida, en la cual otro proyecto de software puede ser organizado y desarrollado.

Maya: Programa para modelar y diseñar gráficos en 3D de altas prestaciones, desarrollado inicialmente por la empresa Alias System Corporation (adquirida por Autodesk en 2005)

OpenGL: es una especificación estándar que define una API multilenguaje y multiplataforma para escribir aplicaciones que produzcan gráficos 2D y 3D.

Path: camino, curso, ruta, senda, sendero, trayecto, vereda; (informática: trayecto de búsqueda).

Plugin: aplicación informática que interactúa con otra para aportarle una función o utilidad específica.

Realidad Virtual: es un sistema o interfaz informático que genera entornos sintéticos en tiempo real, representación de las cosas a través de medios electrónicos o representaciones de la realidad, una realidad ilusoria, pues se trata de una realidad perceptiva sin soporte objetivo, existe sólo dentro del ordenador.

Render: proceso de generar una imagen desde un modelo. Los medios por los que se puede hacer un renderizado van desde lápiz, pluma, plumones o pastel, hasta medios digitales en dos y tres dimensiones.

Rhinoceros: herramienta de software para modelado en 3D basado en NURBS.

Signals y slots: mecanismo de comunicación entre objetos, exclusiva del framework Qt.

Simulador: es un aparato capaz de reproducir un sistema, los simuladores nos hacen vivir sensaciones que en realidad no están sucediendo.

STL: subconjunto de la librería estándar de C++ que posee los contenedores, algoritmos, iteradores, funciones objeto, etc.; aunque algunas personas utilizan el término STL indistintamente con la librería estándar de C++.

Pathfinding: conjunto de rutinas de programación que ayudan a un personaje controlado por el ordenador a encontrar el camino más corto entre dos puntos del mapa, así como a actualizar la ruta ante la aparición de obstáculos inesperados.

Viewport: define las dimensiones de la ventana de una superficie de destino de representación hacia la que se proyecta un volumen 3D.

XML: Extensible Markup Language, lenguaje de marcas extensible.

3D Max Studio: es un programa de creación de gráficos y animaciones 3D desarrollado por Autodesk Media & Entrtaiment (formalmente conocido como Discreet y Kinetix).

3DX: formato de fichero para almacenar información de los entornos virtuales tratados por SceneToolkit.

Índice de Figuras

Índice de Tablas