



UNIVERSIDAD DE JAÉN
Escuela Politécnica Superior (Jaén)

Trabajo Fin de Máster

SIG 3D PARA LA GESTIÓN DE INFRAESTRUCTURAS SUBTERRÁNEAS

Alumno/a: Soria Tirado, Gregorio

Tutor/a: Lidia M^a Ortega Alvarado
Francisco R. Feito Higuera

Dpto.: Informática

Julio, 2017

Índice general

1. Introducción	4
1.1. Estado del arte	7
1.1.1. Avances históricos	8
1.1.2. Comparación entre los distintos SIG	10
1.1.3. Almacen de datos	12
1.1.4. Otros proyectos con Realidad Aumentada	14
1.2. Objetivos del proyecto	17
1.3. Resultados esperados	19
1.4. Justificación de los sistemas escogidos	20
1.4.1. Almacen de datos	20
1.4.2. Hardware de Realidad Aumentada	23
1.4.3. Desarrollo	28
2. Redes subterráneas	30
2.1. Red de saneamiento	30
2.2. Red de abastecimiento	35
2.3. Red de comunicaciones	41
2.4. Red eléctrica	42
3. Planificación y diseño	45
3.1. Análisis y diseño	47
3.1.1. Lista de características	48
3.1.2. Diagrama de caso de uso	48

3.1.3. Dividir el problema	51
3.1.4. Clases	52
3.1.5. Base de datos	74
3.2. Planificación	78
3.3. Presupuesto	80
4. Iteraciones	83
4.1. Iteración 1	83
4.2. Iteración 2	86
4.3. Iteración 3	95
4.4. Iteración 4	106
4.5. Iteración 5	111
4.6. Iteración 6	113
4.7. Iteración 7	115
4.8. Iteración 8	117
4.9. Iteración 9	119
4.10. Iteración 10	124
5. Resultados	131
5.1. Pruebas	132

Capítulo 1

Introducción

En este proyecto nos centraremos en crear una nueva herramienta capaz de gestionar el conjunto de las infraestructuras subterráneas. En la actualidad desde un punto de vista informático, existen multitud de sistemas informáticos para la gestión de éstas. El manejo de estos sistemas se ha convertido en un punto importante a la hora de trabajar en tareas de gestión sobre ellos. Estas redes soportan la mayor parte de servicios que se suministran a los ciudadanos: electricidad, comunicaciones, gas, agua, saneamiento, etc. Habitualmente estas infraestructuras consisten en amplias redes de conductos o túneles que se encuentran debajo del pavimento de las calles y de los edificios, y por lo tanto cuentan con una peculiaridad importante: no son directamente visibles.

A diferencia de la gestión actual de otras infraestructuras urbanas, este tipo de redes presentan limitaciones para ser gestionadas por un sistema de información geográfico (en adelante SIG) en dos dimensiones o por una herramienta CAD. Aunque en estos momentos es el sistema de manejo más extendido, posee un alto grado de abstracción. Es normal que en cualquier labor de mantenimiento o mejora que conlleve realizar una zanja de acceso se cometa un error y acabe produciendo la rotura de otra red cercana. Esto sería fácilmente evitable si se incluyera una tercera coordenada que indicase la profundidad del objeto con el que se quiere trabajar.

Actualmente, es la administración pública la encargada de gestionar el subsuelo pero son las empresas privadas quien se encargan de las tareas de mantenimiento de las redes. Por lo general, estas empresas no cuentan con información sobre las redes de

otras empresas, por lo que es habitual que una empresa indemnice a otra por rotura de sus infraestructura. Estos temas podrían ser solucionados si hubiera un sistema común de almacenaje e intercambio de datos entre ellas. Cada entidad almacena sus datos en su formato preferido, utilizando como anteriormente nombramos un SIG o cualquier herramienta CAD.

Respecto a la forma de trabajo actual de las empresas encargadas, lo más usual es que una persona en la oficina se encargue del diseño de los nuevos elementos o de la actualización de los existentes. Al ser simples planos cargados en herramientas CAD o SIG, ofrecen una pobre experiencia de navegación e interacción con los elementos. Por lo que se deben buscar alternativas a ésta forma de trabajar con los datos. Y aquí es donde entra el estudio de los SIG 3D. Un SIG 3D no es más que un SIG 2D el cual es capaz de cargar, visualizar e interactuar datos en tres dimensiones. Esta forma de trabajar ayuda de una manera más intuitiva al usuario.

Pero ya sea trabajando sobre un SIG 2D, 3D o una herramienta CAD, una vez que se realiza una obra, el operario de campo solo posee un formulario (digital o impreso) donde se muestran las tareas a llevar a cabo. Con la potencia que poseen los dispositivos ubicuos de hoy en día, nos damos cuenta de que se están desaprovechando recursos de estos. Estos aparatos cuentan con multitud de sensores como son cámaras, acelerómetros, gps, etc. que facilitarían la labor en la calle de los técnicos. Una solución a esto es el uso de técnicas de realidad aumentada. Este tipo de funcionalidad consiste en superponer objetos virtuales sobre la realidad que apreciamos a través de la pantalla de un dispositivo. Aunque ya existen avances en este tema, todavía son varios los problemas a resolver. Aun así parece la solución más aconsejable.

En temas relacionados con la realidad aumentada usada para la visualización del subsuelo ya existen algunos avances, los cuales veremos en la Sección 1.1. Este no es solo un problema de visualización de elementos, sino que deberemos posicionar al usuario dentro de un mundo no controlado y deberemos establecer la orientación inicial de hacia donde está mirando el dispositivo en ese momento.

Vista esta introducción donde se hace un repaso a los términos en los que profundizaremos a lo largo de esta documentación, haremos una breve introducción de los objetivos

que buscaremos a lo largo del proyecto. Posteriormente serán extendidos en la Sección 1.3:

- Creación de un sistema de gestión de infraestructuras del subsuelo, con capacidades asociadas a un SIG 3D
- Seguir los consejos de la OGC
- Mejorar la problemática asociada a los actuales SIG
- Creación de un sistema de realidad aumentada que permita trabajar a pie de obra

Hoy en día contamos con multitud de sistemas de almacenamiento e intercambio. Se debería buscar un sistema estándar donde esté situada toda la información y además fuera fácilmente accesible por el resto de empresas.

En 1994 se creó la organización llamada "Open Geospatial Consortium" u OGC, la cual vela por crear estándares sobre el acopio y la distribución de datos georeferenciados. Se han valorado ciertos sistemas para solucionar el problema que conlleva la gestión de infraestructuras urbanas. Como forma de guardar los datos se barajan varias opciones. Una de ellas es la opción del uso de bases de datos espaciales. Este tipo de base de datos no es más que una extensión de sistemas SQL, donde a través de la creación de nuevos tipos de datos y añadiendo funcionalidades geométricas y topológicas se resuelve gran parte de la problemática asociada a otros sistemas como los datos almacenados en formato CAD. De igual manera, este organismo aconseja la utilización de CityGML como formato para compartir información geográfica. Éste estándar es una extensión de GML destinado a infraestructuras urbanas. La gran ventaja de un fichero de este tipo es que es fácil de leer ya que sigue el estándar XML. Por lo tanto, en este proyecto nos guiaremos por las decisiones de la OGC.

El resto de este capítulo será dedicado a una revisión sobre como se ha trabajado hasta ahora y nuevas tendencias de este tema, seguiremos con una expansión de los objetivos comentados anteriormente. Una vez que los objetivos estén claros, especificaremos cuales serán los resultados que esperamos conseguir al termino del proyecto.

Terminaremos dando una visión de las distintas posibilidades existentes para la realización de éste proyecto y tomaremos la que sea más acorde tanto en presupuesto como en facilidad de uso.

1.1. Estado del arte

En esta sección buscaremos información sobre como han funcionado las empresas encargadas del subsuelo desde los inicios hasta nuestros días. Seguiremos con una comparación entre los distintos SIG(2D y 3D) existentes analizando sus puntos fuertes y débiles. Discutiremos cual será la mejor forma de guardar los datos. En la parte final se analizarán otros sistemas innovadores implementados en otros proyectos, que intentan solucionar los problemas sobre la gestión del subsuelo.

1.1.1. Avances históricos

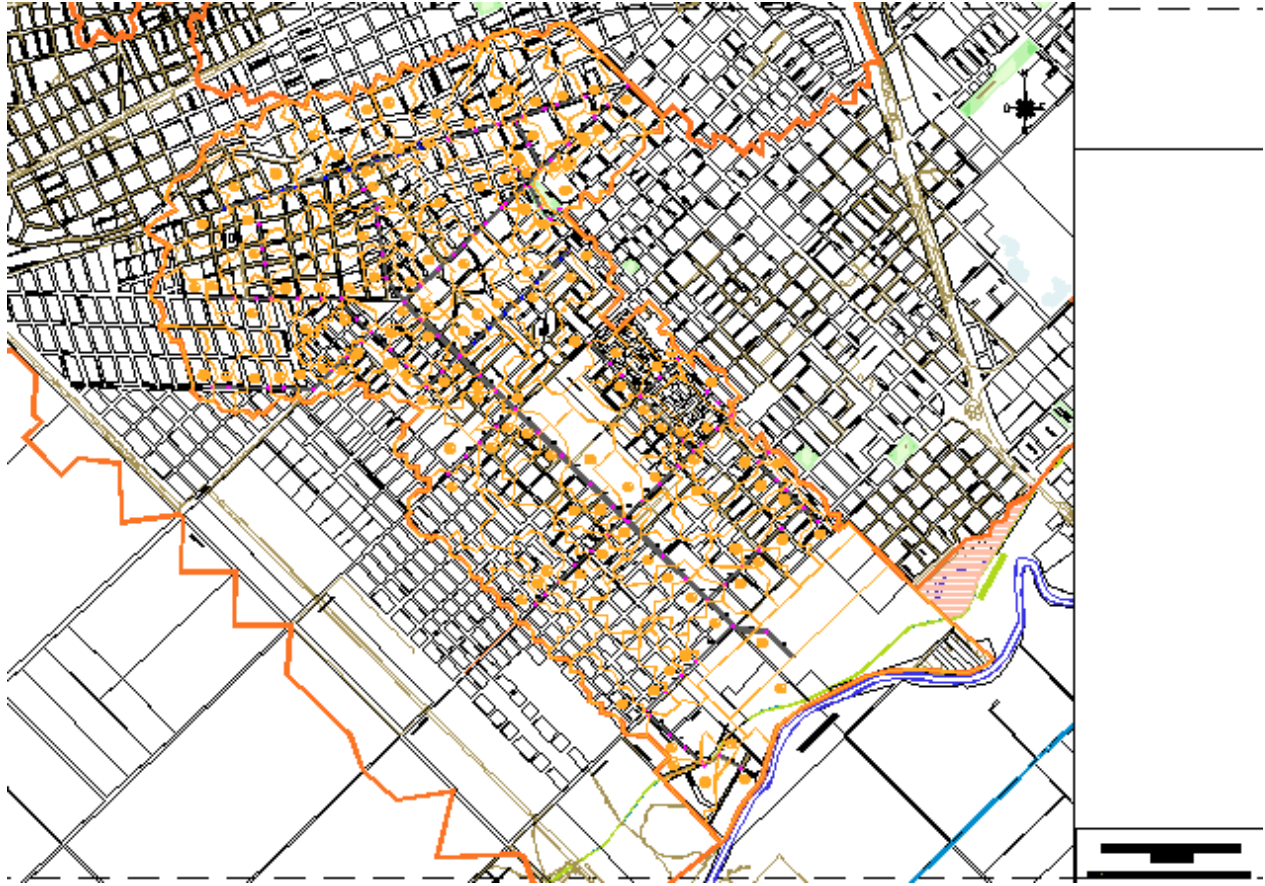


Figura 1.1: Plano de la infraestructura subterránea de una ciudad.

En un principio las empresas encargadas de la gestión de las infraestructuras subterráneas documentaban sus redes a través de planos en papel como podemos ver en la Figura 1.1. El problema de este tipo de técnica radica en la actualización del plano ya que cualquier modificación sobre un elemento podría suponer la réplica de todo el conjunto de componentes en un nuevo esbozo. Como podemos imaginar, la georeferenciación de estas redes no es precisa, por lo que dificulta cualquier tarea de mantenimiento.

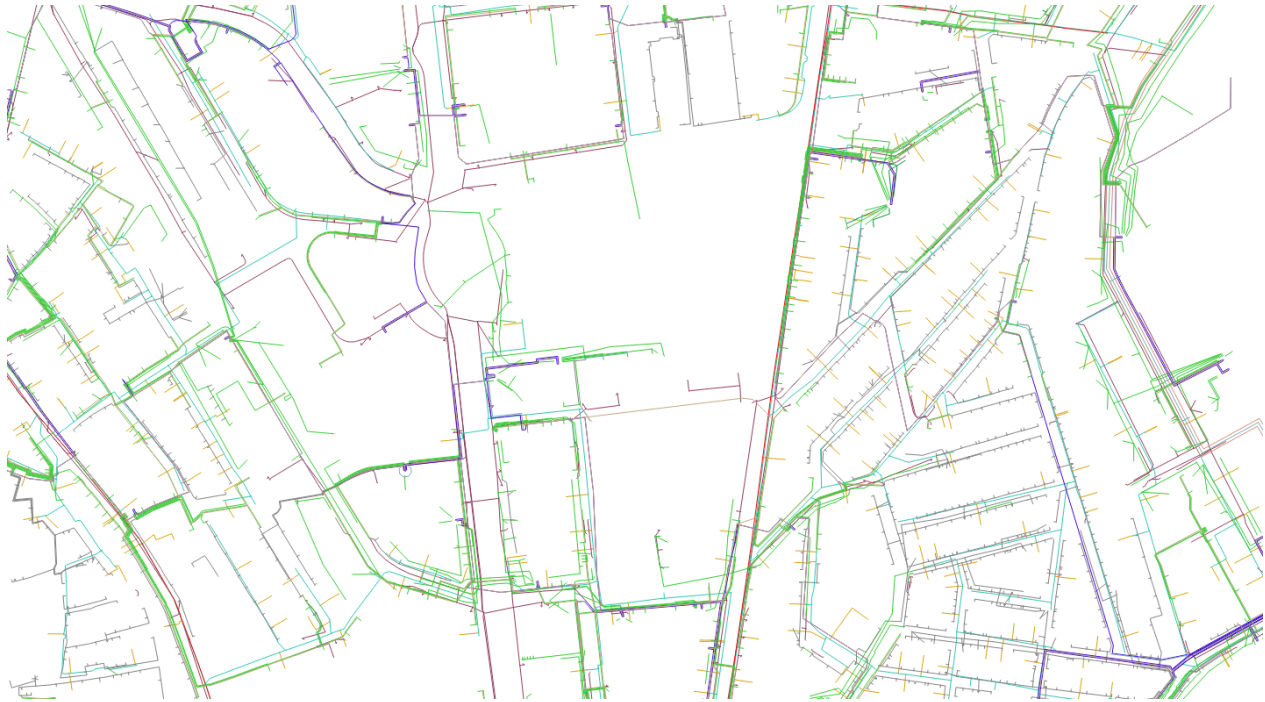


Figura 1.2: Plano en formato CAD de los elementos del subsuelo.

Una vez que se extendió el uso del ordenador, se estandarizó el uso de herramientas CAD como forma de trabajar con los datos. Como vemos en la Figura 1.2, el formato es similar al usado anteriormente. Con esta mejora, se consiguió que la actualización de los planos fuera más eficiente. Cada dato también está referenciado en torno a un sistema de coordenadas, por lo que la precisión a la hora de mantener la infraestructura es mayor. Aún así este sistema también posee ciertas debilidades que hacen que no sea un entorno de trabajo adecuado. Como podemos ver, cada elemento queda representado por un símbolo, por lo que si queremos añadir un nuevo elemento deberemos insertar la misma representación. Esto puede suponer un problema a la hora de cambiar cualquier signo, ya que deberíamos actualizar manualmente el resto de elementos. Pero la gran desventaja está reflejada en que no se posee información de profundidad a simple vista. A la hora de gestionar las redes del subsuelo, esta información es muy importante ya que puede suponer un ahorro de recursos y evita que haya incidentes con otras redes que estén situadas por encima de la que se desea administrar.

Hasta nuestros días los avances de los SIG solo han sido a nivel de implementación de nuevos algoritmos y optimizaciones de los ya existentes. Estas herramientas propor-

cionan un marco de trabajo donde se dibujan puntos, polilíneas o polígonos en él. Al pinchar sobre algún objeto apreciamos una vista en forma de formulario donde vemos sus características. A día de hoy los más utilizados son ArcGIS, QGIS, MapInfo y algunas de sus variaciones. Aunque se ha estandarizado su uso entre multitud de empresas, en el caso de las empresas que nos suministran datos siguen utilizando software de tipo CAD. Los problemas de edición de geometría y la búsqueda de mayor realismo no han sido resueltos todavía.

Los primeros SIG3D, no son más que una extensión de los SIG2D a los que se les incorpora una tercera coordenada. Entre los más importantes está ArcGIS 3D. Aún así no han triunfado entre las empresas, por lo que cabe preguntarse si es por la dificultad de utilizarlos, por sus contras o si no interesa al público al que va destinado.

Al ser numerosas las peticiones de la introducción de una tercera coordenada, las empresas dedicadas a crear software del tipo CAD han empezado a crear los primeros CAD 3D. Hoy en día, el más potente es Civil 3D. En él se pueden editar y visualizar los datos junto a su profundidad. Cada objeto está parametrizado entorno a sus propiedades, dotando a los datos de un gran realismo. Como cualquier CAD, no es capaz de realizar operaciones con los datos.

1.1.2. Comparación entre los distintos SIG

En esta Subsección analizaremos los SIG, tanto 2D como 3D, más importantes actualmente, sus pros y sus contras.

QGIS

Es la continuación del conocido Quantum GIS que tanto fue usado antes de 2002. Es un software de código libre y multiplataforma, incluso funciona bajo dispositivos móviles. Permite manejar datos tanto en formato ráster como vectorial a través de bibliotecas como GDAL y OGR. Como punto fuerte es que soporta multitud de formatos de ficheros, por lo que hace que sea muy útil para la visualización de cualquier información. Es capaz de conectar con bases de datos espaciales. Su representación se basa en puntos, líneas

y polígonos según los datos a cargar. No es capaz de mostrar la tercera dimensión por si mismo por lo que es necesario acudir a algún plug-in para poder usar esta funcionalidad. Pero estos añadidos tampoco están en un nivel avanzado ya que no son capaces de extraer la tercera coordenada de los datos, limitandose a proporcionar la misma profundidad a todos los objetos cargados dentro de QGIS. Tampoco es capaz de modificar la geometría de los datos, por lo QGIS queda como un software de visualización e intercambio entre distintos formatos.

GRASS

Es el GIS "open source" más usado en estos momentos debido a su gran potencia para realizar análisis de datos. Aunque su aprendizaje es costoso, la multitud de algoritmos que soporta lo hacen un GIS casi perfecto. Su tridimensionalidad solo llega a través de ficheros ráster, por lo que no es la mejor opción para nuestro manejo de datos. Tampoco es capaz de editar geometría. Por último, cabe destacar que al basarse también en bibliotecas como GDAL, sirve como base para muchos GIS, incluido QGIS.

ArcGIS

ArcGIS es el GIS más importante y más usado en el mundo en estos momentos. Es un GIS que soporta todo tipo de datos y de formatos. Está producido y comercializado por la empresa ESRI. Contiene edición de datos, tratamiento y análisis. Además cuenta con una aplicación móvil para la captura y gestión de información en campo. Aunque esta no contiene realidad virtual, si ayuda a trabajar in situ gracias a la geolocalización. ArcGIS no está muy especializado en tratamiento de datos 3D. Únicamente en visualización de superficies y alguna simbología 3D. Para todo esto utiliza la extensión 3D Analyst. Las aplicaciones que realizan visionado 3D serían ArcScene y ArcGlobe (forma parte de Analyst 3D). La diferencia fundamental entre una y otra es que, mientras que la primera muestra la información en forma de plano, la segunda la muestra en forma de "virtual globe". 3D Analyst es un toolbox que aumenta las capacidades de ArcGIS en cuanto análisis de superficies. Proporciona un conjunto de herramientas para crear y analizar superfi-

cies representadas como raster, terrenos, TIN y LAS. Los datos que soporta Analyst 3D son básicamente superficies. Existen herramientas que usan ArcGIS para el tratamiento de información del subsuelo como GeosoftTarget. Esta herramienta se encuentra tanto como software independiente como en una extensión para ArcGIS. Es un software geológico para visualizar, analizar y manipular proyectos de perforaciones o prospecciones geológicas. ArcGIS ha sido usado para el modelado de elementos del subsuelo como tuberías o válvulas, como el modelado de estructuras subterráneas de la ciudad de Las Vegas usando Civil 3D (AutoDesk), ArcGIS, Navisworks y Autodesk Infrastructure Modeler (Infraworks).

1.1.3. Almacen de datos

En la presente Subsección buscaremos referencias bibliográficas que nos ayuden a decidir donde almacenar los datos con los que trabajamos y como podríamos compartirlos con otros usuarios para que puedan visualizarlos en otras plataformas.

Nos centraremos en el artículo de [BZ11] donde se muestra la historia de los distintos métodos de almacenaje de datos geoespaciales y de forma más específica, en las bases de datos espaciales. Al principio vemos las distintas estructuras de datos que se han utilizado a lo largo del tiempo. Todo empezó utilizando la estructura espacial Quadtree [FB74] ya que no se contemplaba la utilización de datos en tres dimensiones. Posteriormente se avanzaría con el uso de estructuras de datos arbóreas, usando R-tree [Gut84], R*-tree [BKSS90] y otras [HNP95]. Hasta 1989 [Ege89] no se estableció el uso de la topología en objetos. Esto supuso un gran avance ya que se podían hacer operaciones y análisis de los datos. Gracias a que las bases de datos espaciales avanzaban entorno a conseguir mejoras, quedaba por determinar el modelado de los datos dentro de las bases de datos espaciales.

Los datos a representar son heterogéneos y las aplicaciones de la información llega desde la visualización hasta el análisis. Por lo tanto se debía buscar la transformación de los modelos geográficos para sus distintos usos hacia un tipo de objeto georeferenciado apto para su almacenaje dentro de una base de datos espacial. En 2004 [ZHC]

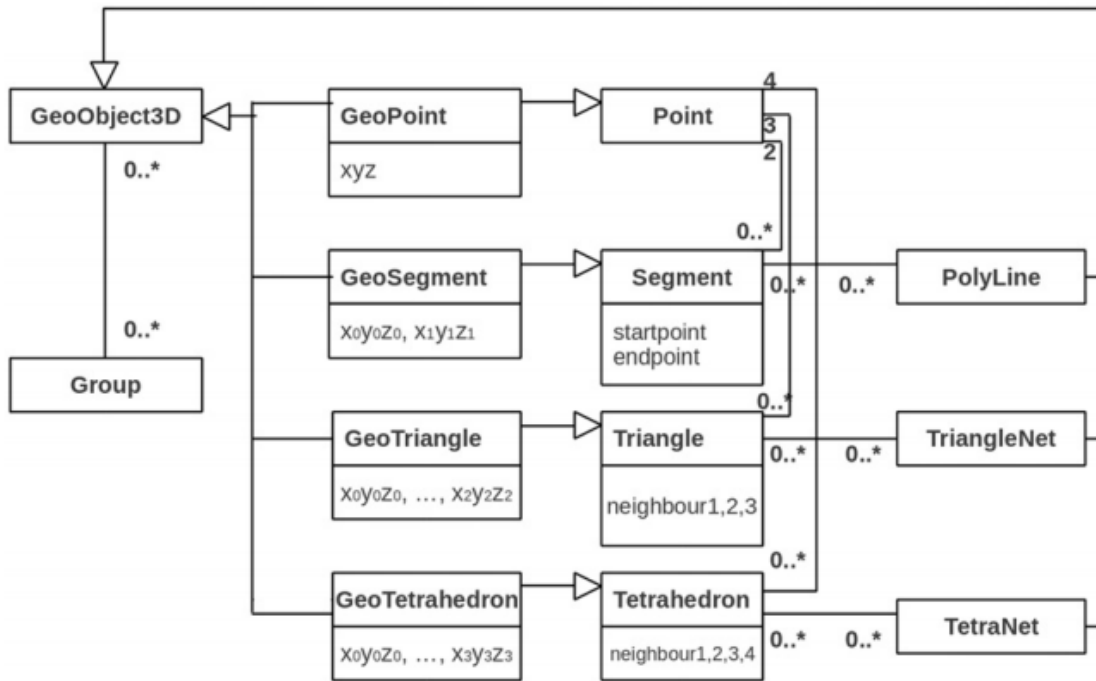


Figura 1.3: Modelado de los datos de una base de datos espacial.

descubrió que no existía un modelo relacional en las bases de datos tradicionales para la información georeferenciada. Fue en 2010 cuando la OGC estandarizó la solución a estos problemas [OGC10]. Propusieron el uso de SQL como forma de acceso a los datos, e inventaron la extensión GML de XML. Con esto, todas las bases de datos espaciales utilizarían los mismos tipos de datos para guardar datos geográficos. El resultado podemos observarlo en la Figura 1.3. Analizándola, vemos como todos los objetos son tridimensionales. Existe un tipo de dato base que es el punto. Los segmentos, triángulos y tetraedros lo utilizan para ser definidos. Por último, a alto nivel, se agrupan los segmentos en polilíneas, los triángulos en mallas y los tetraedros en redes.

Pero la definición de una base de datos espacial no queda definida todavía ya que falta la indexación espacial de los datos, que es la clave de la rapidez de las operaciones realizadas sobre una base de datos espacial. Desde los años 80 se ha investigado sobre el mejor método para el manejo de la información ([Tam82], [Gut84], [NHS81], [Sam05]). Pero ninguno ha conseguido la eficiencia del uso del R-Tree y su variante R*-Tree. Dado que se ha convivido con distintas estructuras arbóreas, se decidió usar el método llamado

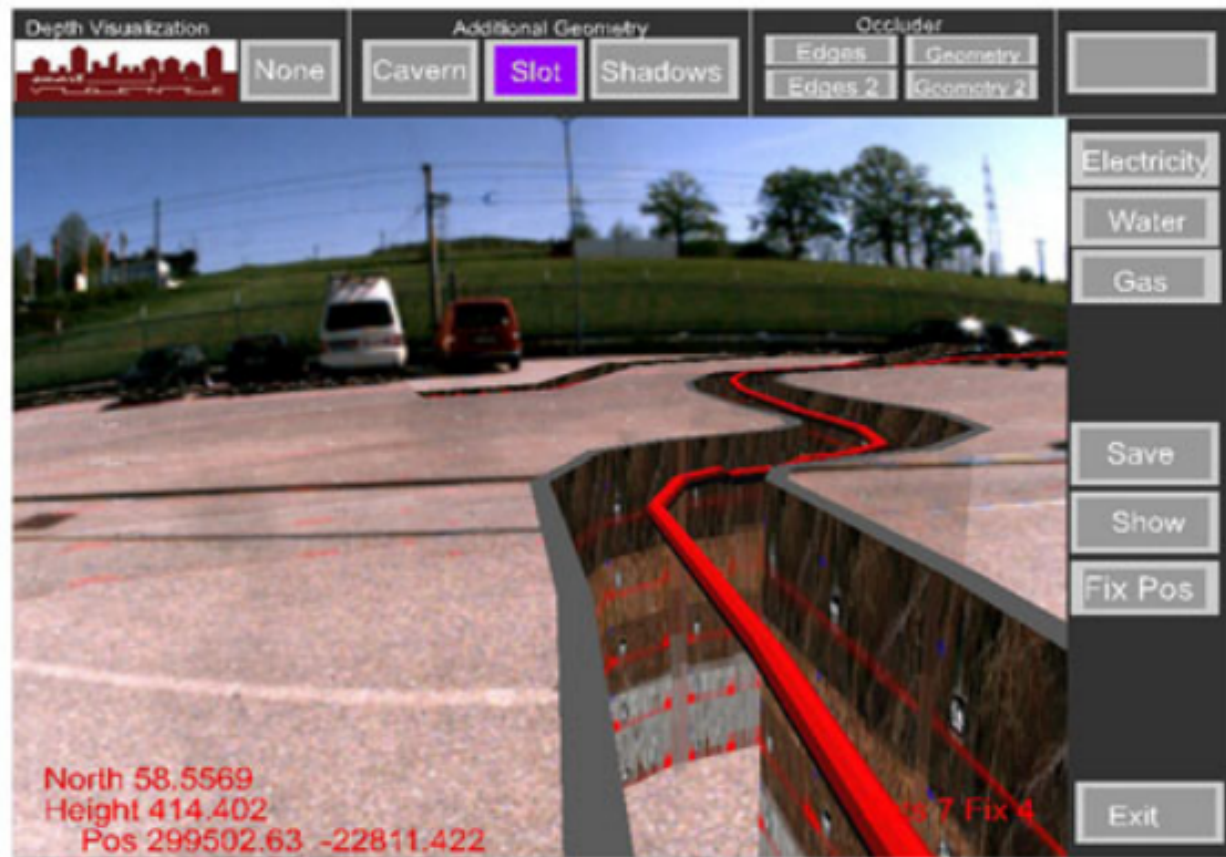


Figura 1.4: SmartVidente.

"Generalised Search Tree" (GiST) propuesto por [HNP95].

Con esto queda claro que la mejor forma de almacenar los datos es con el uso de una base de datos espacial.

1.1.4. Otros proyectos con Realidad Aumentada

En la introducción dimos una pincelada del objetivo de nuestro proyecto para los trabajadores situados a pie de obra. El uso de un dispositivo con capacidad para la utilización de técnicas de realidad aumentada ayudaría a llevar a cabo tareas de mantenimiento. Existen multitud de trabajos que han intentado solucionar esta incognita y que no han conseguido dar solución a algunos de los problemas que tiene el uso de la realidad aumentada en lugares no controlados.

El primer avance lo realizó [SZR12]. En este proyecto llamado "SmartVidente" se creó

un hardware compuesto por una tablet, una cámara, una unidad de medida inercial, un láser de medida y una antena GPS de alta precisión. Los datos están almacenados en una base de datos espacial, son trasladados al sistema de realidad aumentada creado y tras realizar unas operaciones, se pueden ver la representación de la información en forma de objetos virtuales superpuestos con la realidad. Pero antes de iniciar la aplicación de realidad aumentada se debe hacer un proceso de posicionamiento y otro de orientación. Dado que usan un GPS con correcciones diferenciales sobre puntos de control totalmente georeferenciados, consiguen una precisión de posicionamiento casi exacta. De igual manera, la orientación del dispositivo se consigue situando objetos virtuales sobre puntos de control georeferenciados con una precisión menor a un centímetro. Sus resultados pueden observarse en la Figura 1.4.

En 2015 se siguieron haciendo estudios en este campo de mano de [ZHHL15]. En este artículo se usa un framework basado en ARGIS. Se utilizan 13 tipos de redes subterráneas en las que solo se muestran las tuberías, almacenadas en formato Shape. Posteriormente se importan en una base de datos espacial. Los datos son recibidos en formato JSON a través de un rectángulo definido por el rango de visión del teléfono. Para visualizar los datos hacen grandes cálculos con redes neuronales para realizar el posicionamiento y el cálculo de movimiento.

En [BDK15] se muestra un sistema de realidad aumentada donde el objetivo es dotar de una oclusión correcta al sistema. Para ello se usa un sistema ARMOR (Augmente Reality Mobile Operation platform), (Figura 1.5), para procesar las imágenes. Este sistema está compuesto por un GPS submétrico con correcciones diferenciales, una videocámara, una pantalla para visión estereoscópica, un ordenador portátil, un controlador Nintendo Wii Mote, una batería externa y una mochila para transportar el sistema. A través de éste aparataje es posible ver información sobre las tuberías con una precisión centimétrica y recibir información sobre las mismas mediante el controlador. El resultado del uso de este proyecto podemos verlo en la Figura 1.6.

La investigación realizada por [LSP15] llamada BoreholeAR, tiene como resultado una aplicación donde se muestra información de los pozos subterráneos de una ciudad. Como hardware utiliza un iPad Air. Empiezan creando una base de datos espacial donde se



Figura 1.5: Estructura de un sistema ARMOR.

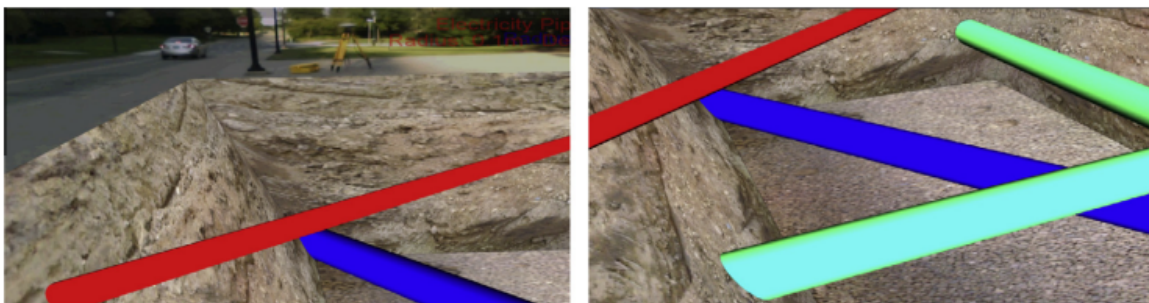


Figura 1.6: Resultados usando ARMOR.

almacenan los pozos. Para la locación utilizan el GPS básico del dispositivo. En este proyecto no se consigue ver una representación tridimensional de los pozos, limitandose a mostrar información sobre su identificador. Mostramos los resultados en la Figura 1.7.

Posteriormente, [ZHHL16] como continuación a [ZHHL15], pasó de utilizar redes neuronales para almacenar datos sobre el movimiento del usuario a través de los sensores. Cada lectura de los sensores determina una "pose". Los pasos intermedios entre dos poses son interpolados, dando así una precisión casi exacta del hacia donde está mirando el teléfono en cada momento. El único requisito para que la aplicación sea iniciada correctamente es que el teléfono debe estar apuntando al norte y la cámara debe estar apuntando hacia arriba (Figura 1.8). Respecto a la visualización, no se mejoró nada. A

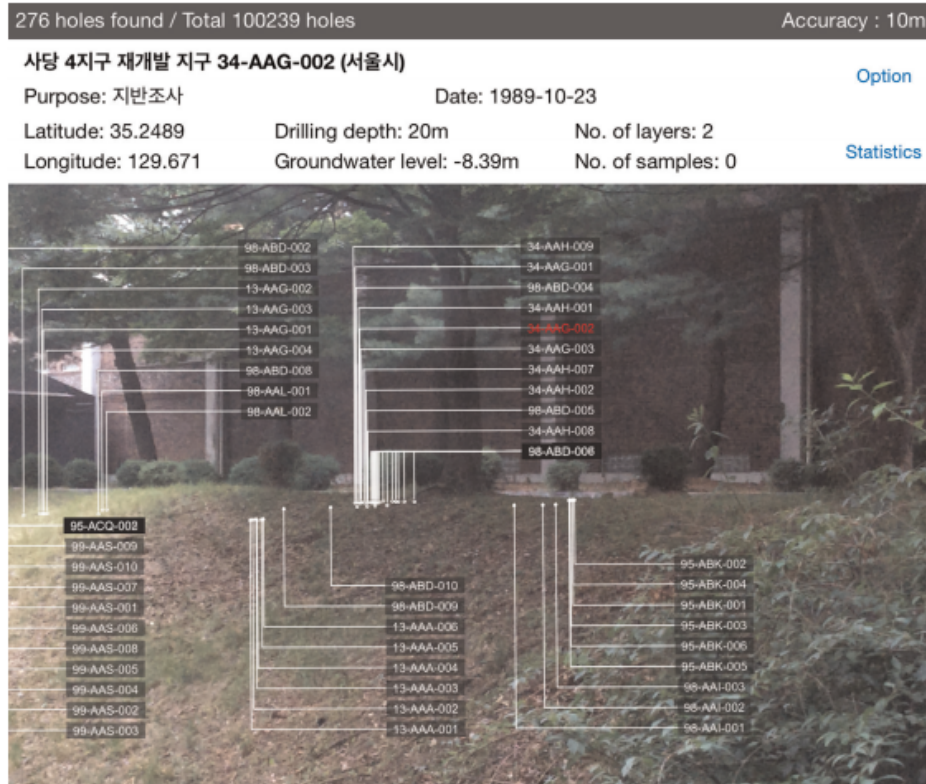


Figura 1.7: Resultados de BoreholeAR.

partir de este momento ya se podían ver todos los elementos del subsuelo y no solamente tuberías.

Por último, [ZPV16] utiliza métodos de posicionamiento y orientación para etiquetar edificios. Aunque este proyecto no está relacionado con el subsuelo, puede servir como referencia sus métodos propuestos. Los buenos resultados de esta aplicación se consiguen gracias a la realización de un "matching" entre los edificios que capta el dispositivo y los datos almacenados en una base de datos espacial.

1.2. Objetivos del proyecto

En esta sección trataremos de establecer los objetivos que deberemos conseguir a la finalización del proyecto. Para ello buscaremos los puntos fuertes y débiles vistos en la Sección 1.1. Trataremos de aprovecharnos de las ideas que proponen y que son a priori correctas, e intentaremos no reproducir las debilidades que algunos proyectos poseen.

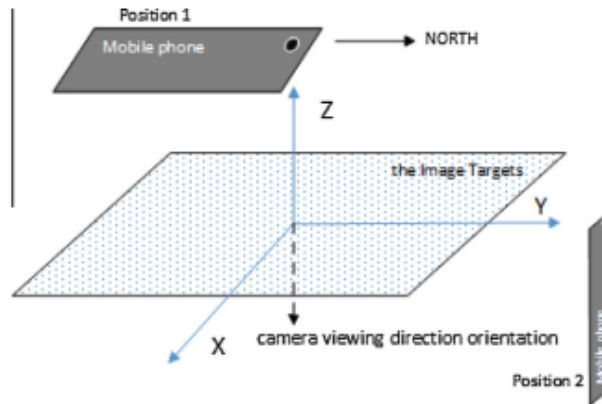


Figura 1.8: Posición inicial del dispositivo en ARGIS.

Nuestros objetivos serán los siguientes:

- Integración de los datos en un sistema de almacenaje:
 - La forma de almacenar los datos que actualmente está establecidos no es la más adecuada, los datos no tienen una georeferenciación
 - Los datos deben ser extraídos de forma simple para su intercambio
 - Los ficheros en formato CAD son poco escalables
- Visualización de los datos en formato GIS, introduciendo la coordenada de altura
 - Los GIS 3D actuales no tienen en cuenta las propiedades intrínsecas de los datos
 - Los datos en 2D deben tener también cabida dentro de nuestro software
 - La aplicación debe ser única, no debe depender del formato de los datos
 - Incorporación de la topología
- Operaciones
 - Integrar la edición de objetos, tanto a nivel geométrico como sus propiedades temáticas
 - Esta modificación debe ser natural, sencilla y adaptada a la naturaleza de los datos

- La modificación debe estar estandarizada, sin importar en que dispositivo se esté usando
- Visualización con Realidad Aumentada
 - El hardware usado no debe ser incomodo para el usuario
 - Debe proporcionar distintos métodos de posicionamiento, por si alguna opción no estuviera disponible en ese momento
 - Permitir la edición y actualización de los datos
 - Cumplir su función en cualquier parte del mundo sin tener que realizar labores costosas antes de poder usar el software

1.3. Resultados esperados

En la Sección hicimos una pequeña introducción sobre los resultados que debemos conseguir para que este proyecto quede en una fase bastante avanzada. En esta sección ampliaremos estos puntos.

Por tanto, los resultados esperados son los un sistema que integre estas capacidades:

- Base de datos espacial
 - Los elementos de las infraestructuras subterráneas deben estar clasificados según su naturaleza
 - Debe contener toda la información básica que parametrizan los datos
 - Contener topología entre los distintos elementos de las infraestructuras subterráneas
 - Fácilmente exportable a otros formatos
 - Con capacidad de análisis sobre los datos espaciales
- Aplicación de gestión
 - Debe tener capacidades asociadas a un SIG 3D

- Debe permitir utilizar distintas capas, según seleccione el usuario
 - Un sistema de carga edición y actualización
 - Permitir operaciones de modificación
 - Dado que los datos están topológicamente conectados, cualquier operación de inserción, borrado o movimiento de un objeto debe afectar a los objetos conectados
 - Diferentes tipos de visualización
- Aplicación de realidad aumentada
 - Ofrecer al usuario varios métodos de posicionamiento
 - Un sistema de orientación fácil y entendible para el usuario
 - Incluir todo lo que incluye la aplicación de gestión

1.4. Justificación de los sistemas escogidos

1.4.1. Almacén de datos

Como hemos visto en la Sección 1.1, los mejores resultados se han conseguido a través de una base de datos espacial. Por lo tanto, creemos que no debe haber discusión alguna sobre cual sistema usar. A continuación, explicamos los beneficios que supone el uso de una base de datos espacial:

- No requiere un software especial para leer los datos ya que sigue el estándar SQL
- Las operaciones sobre los datos están ya programadas
- Cualquier sistema suele ser compatible con las bases de datos espaciales
- Los datos están unificados en una tabla, por lo que no es necesario utilizar varios ficheros

- Uso de estructuras de datos e índices espaciales que hacen más rápido el acceso a la información
- Extensión de funcionalidades mediante el uso de funciones
- Datos georeferenciados
- Cambio de proyección al vuelo
- Capacidad de almacenamiento de datos tradicionales junto a datos de tipo ráster o vectoriales
- Análisis espacial y operaciones

Ya que estamos convencidos de que una base de datos espacial es la mejor opción, debemos decidir cual usar entre las tantas que existen.

Oracle Spatial

Es la opción de manejo de datos espaciales de Oracle [Ora]. Posiblemente es la opción más potente que existe en la actualidad en el mercado ya que son numerosas las opciones que permite. Su interfaz no es amigable, por lo que debe ser controlado por un especialista. Otro punto en contra es su alto coste, ya que esta extensión cuesta 304 euros a parte de los 156 euros del software principal. Por lo tanto, está más destinado a grandes empresas.

Microsoft SQL Server

Es la herramienta más potente en torno a bases de datos no espaciales y en estos momentos es una de las más usadas en el mundo. Pero no es así en lo que a datos espaciales se refiere, ya que la versión gratuita no incluye funciones espaciales avanzadas, por lo que si se desea un sistema completo deberá realizar una inversión más costosa que la ofrecida por Oracle. [Mic]

MySQL

Esta herramienta está muy limitada. En ella solo se pueden realizar algunas operaciones básicas. No tiene tantos tipos de datos como el resto de herramientas.[MyS]

SpatialLite

Es una herramienta que está empezando a madurar. Es Open Source. Empieza a incluir funciones para el tratamiento de datos que pueden ser interesantes. Su mejor cualidad es su facilidad de uso. [Spa]

PostGIS

PostGIS es una extensión de PostgreSQL [Posb]. Si miramos la Sección 1.1, observamos que todos los proyectos utilizan este sistema de base de datos. No solo por ser libre y gratuito, sino porque es la herramienta más madura del mercado. Es compatible con los estándares marcados por la OGC. En su amplia documentación encontramos más de mil funciones espaciales. Otro punto a favor es la importación y exportación a otros formatos a través de herramientas que se instalan junto a él. La gran mayoría de SIG funcionan como clientes de este servicio. Es modular, por lo que los usuarios pueden crear sus propias extensiones y compartirlas con el resto del mundo. Por todos estos motivos, han hecho de PostGIS[Posa] el sistema a usar por multitud de agencias gubernamentales.

Elección

Entre las posibles opciones que tenemos, usaremos PostGIS. Además de que es una herramienta fiable y utilizada por millones de usuarios, al ser gratuita nos abarata los costes del proyecto.

1.4.2. Hardware de Realidad Aumentada

Empezamos observando las plataformas usadas en la Sección 1.1 y nos damos cuenta de que no es conveniente crear una herramienta de tipo ARMOR. Sus dimensiones y peso hace que un trabajador se sienta incómodo a la hora de trabajar en la calle. Debemos buscar sistemas que más adecuados para el trabajo. En el campo del hardware con realidad aumentada se han realizado multitud de avances en los últimos años, por lo que buscaremos cuales son los mejores valorados y más potentes y sobre estos seleccionaremos el que más se adecue a nuestras necesidades.

iPad



Figura 1.9: iPad.

iPad (Figura 1.9) es la tablet comercial de la marca Apple. En cualquiera de sus versiones ha sido una de las tablets más vendidas por su gran potencia y renombre. Cuenta con sensores GPS, brújula, giroscopio, acelerómetro y una cámara. Contiene una GPU bastante poderosa, por lo que se acoplaría perfectamente a nuestro proyecto. Como defecto,

no cuenta con un sensor tipo "unidad inercial de medición", por lo que si lo utilizáramos en nuestro proyecto deberíamos programar el seguimiento del usuario nosotros mismos. Teniendo en cuenta que estos sensores no están preparados para realizar un "motion tracking" del usuario, éste fallaría a menudo. Por este motivo y por su alto precio, descartamos la opción de usar este dispositivo en nuestro proyecto.

Tablet y smartphone Android



Figura 1.10: Tablet con el sistema operativo Android.

Como sabemos, existen multitud de tablet (Figura 1.10) y smartphones (Figura 1.11) que funcionan con el sistema operativo Android. Por éste motivo existen varias gamas sobre estos dispositivos. Cualquier dispositivo móvil del mercado incluye los mismos sensores que un iPad, diferenciando una de otra solo por la potencia de sus componentes. Queda claro que si queremos usar nuestra aplicación, debemos encontrar alguno que tenga una buena GPU y un buen procesador. Pero la situación sería la misma que con un iPad, deberíamos crear el seguimiento. Por lo tanto, descartamos este tipo de dispositivo.



Figura 1.11: Smartphone con el sistema operativo Android.

Microsoft Hololens



Figura 1.12: Gafas con RA Microsoft Hololens.

Estas gafas de realidad aumentada (Figura 1.12) es la gran apuesta de Microsoft. En ellas podemos superponer objetos virtuales a la realidad e interactuar con ellos con nuestras manos. En este caso, incluye una unidad inercial de medición, por lo que el seguimiento del usuario es bastante aproximado a la realidad. El único sensor que falta es un GPS, por lo que si damos la posibilidad de que el usuario elija su posición en un mapa, podríamos llevar a cabo el proyecto con esta tecnología. El mayor inconveniente de este hardware es que algunos usuarios dicen que sufren mareos y nauseas, no se deberían aceptar estas situaciones mientras el usuario está trabajando.

Google Tango



Figura 1.13: Tablet con RA Google Tango.

Google Tango (Figura 1.13) es la apuesta de Google por la realidad aumentada. A simple vista no es más que una tablet normal, pero si observamos la parte trasera encontramos sensores preparados para ofrecer al usuario la experiencia que la realidad aumentada ofrece. Cuenta con dos cámaras, una unidad de medida laser y una unidad inercial de medición. Estos componentes unidos es lo que Google llama el "proyecto Tango". Cualquier dispositivo que contenga estos sensores y pueda utilizar Android como sistema operativo, podrá recibir aplicaciones con tecnología de realidad aumentada. Su peso y tamaño es igual que cualquier tablet del mercado, por lo que es fácil de transportar.

El proyecto Tango proporciona las siguientes características a un software:

- **Rastreo de movimiento:** El dispositivo es capaz de seguir su posición y su orientación. Si el usuario camina y a la vez mira alrededor a través de la pantalla, el sistema sabrá en todo momento donde está de forma relativa al punto donde comenzó a rastrear y la orientación. Esto es posible gracias a la unidad de medición inercial que proporciona la posición y la orientación del dispositivo con seis grados de libertad. Cada combinación de una posición y una orientación es almacenada como una "pose" que son trasladadas a la cámara virtual de la escena en forma de posición y rotación de la misma.
- **Aprendizaje del área:** Usando solo el rastreo de movimiento, el dispositivo no re-

cordará por dónde ha estado moviéndose y hacia donde ha estado mirando. Si además de esta habilidad, dotamos de memoria al sistema, la siguiente vez que el usuario pasee por el mismo lugar, será detectado por el smartphone lo que conlleva una mejora de la precisión de la trayectoria actual y la capacidad de orientarse y posicionarse de una forma más efectiva sin tener que acceder a los servicios en la nube que Google ofrece. Para este reconocimiento el sistema debe ver y recordar las principales características del entorno, como son las esquinas o las aristas. Posteriormente esto es almacenado a través de una descripción matemática del mundo.

- **Percepción de profundidad:** Proporciona la posibilidad de reconocer los objetos de sus alrededores y su posición en el espacio. Para ello es necesario un proyector infrarrojo y una visión estereoscópica. Con esto se puede conseguir interactuar con los objetos de alrededor, medir áreas o crear una nube de puntos con color de lo que se está viendo. Como algunos scanners, Tango también es afectada por la falta o exceso de luz, igual que por objetos que la reflejen.

Leyendo sus características, hacen que Google Tango sea el dispositivo más adecuado para trabajar en nuestro proyecto.

Lenovo Phab 2 y Asus ZenPhone 3

Tanto Lenovo como Asus han querido unirse al proyecto Tango sacando al mercado dos smartphones, Phab 2 (Figura 1.14) y ZenPhone 3 (Figura) respectivamente. Su GPU es de mayor potencia que Google Tango, aunque su tamaño de pantalla es menor. El uso de estos dispositivos supondría llevar a buen puerto a nuestro proyecto.

Elección

Queda decidido que usaremos un dispositivo implicado en el proyecto Tango. Entre las tres posibles opciones descartamos los dos smartphones dado que, aunque sean más potentes entorno a la GPU, la de Google Tango soporta nuestros gráficos. Pero el



Figura 1.14: Smartphone con RA Lenovo Phab 2.

principal motivo es el tamaño de la pantalla. Un trabajador se sentirá más cómodo con una pantalla mayor, ya que podrá apreciar mejor los elementos superpuestos a la realidad y la información que les acompaña.

Si entramos en la web del proyecto Tango, se nos ofrece varias formas para crear aplicaciones con realidad aumentada. Por lo que se abre otro debate, como vamos a programar la aplicación.

1.4.3. Desarrollo

El proyecto Tango ofrece tres sistemas para crear una aplicación, las cuales comentamos a continuación:

Lenguaje C

Esta programación se hace difícil ya que tendríamos que implementar toda la interacción con los objetos, su caja envolvente, geometría, shaders y demás. No es un sistema nativo para Android, por lo que el compilador debería pasar desde C hacia Java, haciendo la aplicación más lenta y menos eficiente.



Figura 1.15: Smartphone con RA Asus ZenPhone 3.

Java

Es la forma natural para programar un software que corre bajo un sistema operativo Android. Contamos con los mismos problemas que con C, deberíamos desarrollar nosotros todo lo referido a los gráficos, haciendo que el proyecto sea más largo.

Unity

Unity es un potente motor de videojuegos. En el ya están programadas las funciones gráficas de cualquier objeto virtual, por lo que ahorramos tiempo en tareas laboriosas. Con el fin de acelerar el desarrollo del proyecto y centrarnos exclusivamente en nuestros objetivos, usaremos Unity para desarrollar nuestra aplicación.

Capítulo 2

Redes subterráneas

En este capítulo analizaremos los tipos de redes subterráneas, sus características y los elementos que la componen.

2.1. Red de saneamiento

Una red de saneamiento o también llamada red de alcantarillado es aquella que tiene como objetivo el recoger agua residual, conducirla fuera de los núcleos urbanos para evitar que suponga un peligro para la salud humana y tras un procesamiento de reducción del nivel de contaminación es devuelta al medio acuático natural. Este agua contaminada puede llegar desde edificios, fuentes públicas, incluso desde las precipitaciones.

Este tipo de redes normalmente funcionan por presión atmosférica, es decir, por gravedad, no es usual que se necesiten elementos que trabajen en aumentar la presión del agua y si en algún momento son necesarios tienen que ser en pequeños tramos.

El trazado de este tipo de redes normalmente sigue con el trazado de la red viaria y deben estar separados al menos 1.5 m del bordillo, en medio de los carriles de circulación y deberán estar al menos a 2 metros de los edificios ya que esta red puede afectar de forma negativa a los cimientos de estos. En cuanto a la profundidad, esta red deberá estar a más de un metro del pavimento.

En general, el trazado de estas redes está compuesto por tuberías y nodos, llamados pozos que conectan tramos de conducciones. Estas tuberías suelen recibir nombres



Figura 2.1: Arqueta

distintos según su diámetro, su longitud y su objetivo dentro de la red.

En España existen multitud de empresas que se encargan de las redes de saneamiento y esto depende principalmente de la población, siendo el ayuntamiento de la localidad quien pone a concurso la gestión de la red. Algunas de estas empresas son Emasesa (Empresa Metropolitana de Abastecimiento y Saneamiento de Aguas de Sevilla S.A.), Somajasa (Sociedad Mixta del Agua Jaén S.A.) o Aqualia. Ésta última es la encargada de este tipo de red en Jaén capital, siendo Somajasa la encargada en otras poblaciones de Jaén.

Los elementos subterráneos que habitualmente encontramos en este tipo de redes son los siguientes:

Arqueta

Una arqueta o también llamada acometida es un pequeño depósito destinado a recibir agua no potable proveniente de edificios y a través de conductos, llevar el agua hacia un pozo donde posteriormente se incorporará a las conducciones. Estas pueden ser tanto de ladrillo como prefabricadas en hormigón o plásticos. Suelen estar enterradas e incorporará una tapa desde la que se puede acceder a ella para realizar trabajos de reparación

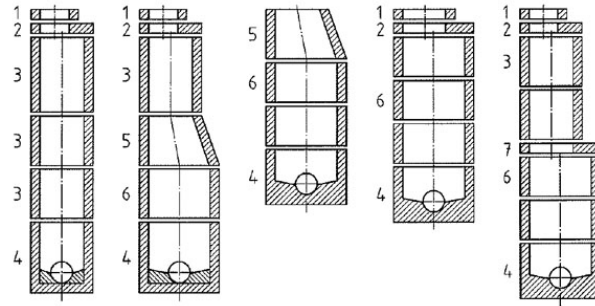


Figura 2.2: Esquema de como puede ser construido un pozo



Figura 2.3: Pozo

o de mantenimiento. Normalmente las arquetas tienen una dimensión variante entre 0.60 y 1 metro cúbico, siendo posible la incorporación de suplementos para aumentar la altura de ésta. Podemos ver una imagen en la Figura 2.1.

Pozo

Los pozos, junto a las conducciones, son los elementos más importantes de este tipo de red, ya que se encargan de almacenar y repartir el agua entre las conducciones que están interconectados a él. Su diámetro interior está entorno a los 1.2 metros, permitiendo que operarios puedan trabajar en el interior en tareas de inspección o limpieza. Normalmente están fabricados con hormigón armado y tienen sección regular. Su colocación es obligatoria en los cambios de dirección, en la confluencia de dos o mas colectores, al principio de cada colector, en los cambios de sección de las conducciones y en tramos

rectos de la red cada cierta distancia.

Un pozo puede estar compuesto por distintas piezas:

1. Módulo de ajuste: Son reducciones que se realizan a los pozos en los que se colocan las tapas por donde posteriormente se accederá.

2. Losa de cierre: Elementos que reducen las dimensiones del pozo hasta el módulo de ajuste.

3. Módulo de recrecido alto: Constituyen la altura del pozo, estas piezas se instalan para conseguir una mayor profundidad del mismo. Cada pieza suele ser de un metro de altura.

4. Módulo de base: Es la parte de el pozo que recibe las canalizaciones entrantes y salientes. Siempre están situadas en la parte inferior del pozo.

5. Módulo cónico: Es un elemento cuyo objetivo es adaptar dos piezas de distinto diámetro dentro del pozo con el fin de agrandar el diámetro.

6. Módulo de recrecido bajo: Aquellas partes que son colocadas debajo de otras piezas con el fin de aumentar el diámetro de pozo.

7. Losa reductora: Es un elemento que reducen el diámetro del pozo.

En la Figura 2.2 podemos ver varios esquemas de pozos que siguen la numeración anterior, de igual manera, en la Figura 2.3 podemos ver un pozo.

Imbornal

Un imbornal o también llamado sumidero son los elementos cuya función es recoger el agua pluvial y conducirlo a la red de saneamiento a través de los conductos. Normalmente el modelo usado es el de rejilla. Sus dimensiones suelen ser reducidas, pudiendo rebosar en caso de grandes precipitaciones. Estos elementos se instalan a nivel de los bordillos del acerado. Podemos ver un imbornal en la Figura 2.4.

Conducto

Los conductos son llamados en ocasiones colectores terciarios y se encargan de conducir el agua desde las arquetas y los imbornales hasta los pozos. Su sección es de



Figura 2.4: Imbornal

tamaño pequeño ya que la cantidad de agua que transporta no es excesiva. En la Figura 2.5 podemos ver una arqueta de ladrillo, la cual tiene una salida en forma de conducto.

Conducción

Una conducción es una tubería de mayor sección que interconecta pozos. Debido a su tamaño pueden ser visitables para realizar tareas de mantenimiento. Este tipo de tubería aumenta su diámetro, variando entre los 0.1 y 1.4 metros, según el caudal que recauda y la cercanía al destino final. Debido a la importancia de estas deben ser cuidadosamente fabricadas, debiendo resistir a impactos, elementos químicos y biológicos y a la abrasión. Para ello se usan materiales como el hormigón armado, el gres, la fundición dúctil y el policloruro de vinilo rígido. Podemos ver un pozo y un par de conducciones de hormigón armado en la Figura 2.6.

Una vez vistos los componentes de la red de saneamiento podemos verificar como



Figura 2.5: Conducto



Figura 2.6: Conducción

funciona esta red (Figura 2.7). Las arquetas(3) e imbornales(1) proporcionan agua no potable a los pozos(4) a través de los conductos(2), una vez en los pozos(4) a través de conducciones(5) atraviesan otros pozos(4) y dirigen el agua hacia otras conducciones(5) de mayor tamaño para termina en una depuradora de agua, la cual devuelve el agua al medio acuático más cercano.

2.2. Red de abastecimiento

La red de abastecimiento está definida como aquella red que es capaz de suministrar agua potable a los edificios, la industria o al servicio público, como pueden ser las fuentes.

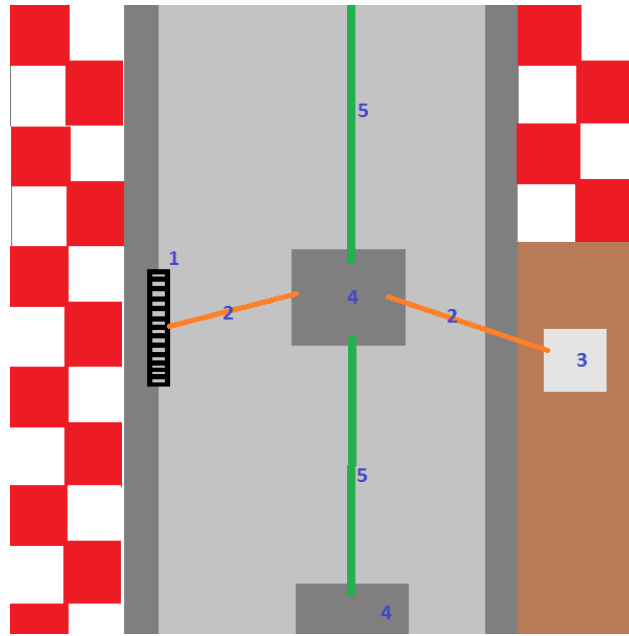


Figura 2.7: Esquema de una red de saneamiento

Estas agua provienen de distintos lugares, como puede ser el mar, lagos, ríos, aguas manantiales y demás. Antes de ser distribuida debe ser potabilizada.

Esta red se divide en dos partes, la parte que transporta el agua a través de la población, también llamada “red arterial” y la red secundaria, la cual se encarga de conectar la red arterial con las derivaciones hacia los edificios, bocas de riego, y tomas contra incendios.

Cabe destacar que igual que pasa con la red de saneamiento, una vez que las acometidas llegan a los edificios, la empresa encargada de la instalación de aguas no es responsable de la instalación de un particular.

La profundidad mínima de esta red es de mayor de 0.8 metros si discurre por el acera y de mayor de 1 metro si discurre por la calzada. Deberá estar separada más de 0.4 metros del resto de redes de servicios y siempre por encima de las redes de alcantarillado para evitar la contaminación de las aguas. Respecto a los edificios, las tuberías deberán estar a más de 0.8 metros de cualquier fachada, cimientos u otras instalaciones subterráneas similares.

Uno de los aspectos más importantes de la red de abastecimiento es la velocidad del agua, por lo que esta debe de quedar limitada entre dos valores para que la red funcione



Figura 2.8: Boca de riego

correctamente. En caso de una velocidad excesiva se producen elevadas pérdidas de carga y sobrepresiones que se traducirían en la rotura de las conducciones. Si la velocidad es baja, puede provocar que se formen depósitos de materias en suspensión que puedan provocar obstrucciones. Por lo tanto la velocidad deberá estar entre los 0.3 y 2 m/s.

Los elementos subterráneos de esta red son los siguientes:

Boca de riego

Son elementos cuyo uso exclusivo es para la limpieza de las calles y en casos especiales, para el servicio contra incendios, también reciben el nombre de hidrante. Deberán estar colocadas a nivel del pavimento de la calle, en las aceras y estarán alimentadas por derivaciones de la red general. Podemos ver una en la Figura 2.8.

Caudalímetro

Es un elemento de medición del caudal circulante por los conductos de la red de abastecimiento. Existen dos tipos, electromagnéticos y ultrasónicos. El primero se basa en el principio de Faraday, el cual dice “Si un conductor se mueve en el interior de un campo magnético se induce una fuerza electromotriz en el conductor, proporcional a su velocidad y perpendicular a la dirección del movimiento del campo”, y el segundo basado en la



Figura 2.9: Caudalímetro

medición de la diferencia de tiempos de propagación de una onda acústica impulsional, emitida alternativamente, en el sentido o en dirección contraria al paso del fluido.

Podemos ver un caudalímetro ultrasónico en la Figura 2.9.

Contador

Es un elemento de medición que se sitúa entre la entrada de un abonado y la red de abastecimiento. Este aparato cuenta el agua que pasa a través de él y un operario lo revisará cada cierto periodo para determinar cuanto debe pagar un cliente. Esta instalación la debe realizar la misma empresa que gestiona la red. Podemos ver uno en la Figura 2.10.

Desagüe

Es un componente de la red que permite el vaciado de ésta. Consiste en una derivación en la parte inferior de la tubería controlado por válvulas de seccionamiento y un



Figura 2.10: Contador



Figura 2.11: Desagüe

tramo de tubería que termina en la red de alcantarillado. En la Figura 2.11 encontramos un desagüe.

Regulador de presión

Son elementos hidromecánicos que provocan de forma automática una pérdida de carga tal que la presión del agua no supere un valor máximo. En general no se debe reducir la presión más de un 50 % en un mismo regulador, por lo que si es necesario se instalarán mas reguladores en serie. Vemos un regulador de presión en la Figura 2.12.



Figura 2.12: Regulador de presión



Figura 2.13: Tubería

Tubería

Una tubería es un conducto que cumple la función de transportar agua u otros fluidos. Se suele elaborar con materiales muy diversos. También sirven para transportar materiales que, si bien no son propiamente un fluido, se adecuan a este sistema: hormigón, cemento, cereales, documentos encapsulados, etcétera.



Figura 2.14: Canalización

2.3. Red de comunicaciones

Las redes de comunicaciones son aquellas que proporcionan servicios de conexión entre líneas telefónicas, internet y televisión por cable. En un mismo municipio pueden existir numerosas empresas que tengan instalaciones subterráneas, por ejemplo, Vodafone y Movistar. Esta red se aconseja que esté separada 20 cm de las redes de baja tensión y 25 de las de alta tensión, igualmente deberá estar a 30 cm del resto de redes subterráneas.

Al contrario que en otras redes, solo existen dos tipos de elementos subterráneos en esta red:



Figura 2.15: Cámara

Canalizaciones

Es una obra de trazado lineal formado por un conjunto de conductos por los que pasa los cables y dan soporte a los elementos del exterior. Vemos una canalización en la Figura 2.14.

Cámara

Es una arqueta de mayor dimensión por donde cruzan los conductos, cubierta por una tapa superior y con escalera por donde puede acceder un operario para trabajar sobre los cables. En la Figura 2.15 vemos a un operario trabajando en una cámara.

2.4. Red eléctrica

Esta red es la que proporciona electricidad a los consumidores. Para ello es necesario producir la electricidad, esto se hace en una planta generadora a través de combustibles fósiles y no fósiles. Una vez producida se distribuye a lo largo de kilómetros a través de cables de alta tensión, ya sean subterráneos o aéreos, hasta llegar al núcleo de una población donde se transformará en electricidad de tensión media y será transformada de nuevo en los centros de distribución para llegar a los edificios en forma de cables de baja

tensión.

Aunque existen multitud de empresas que proporcionan electricidad a los usuarios, la más influyente en Andalucía es Endesa.

Esta red debe seguir un trazado lo más rectilíneo posible y si se puede, de forma paralela a las líneas de las fachadas y bordillos. Además se debe tener en cuenta que la curvatura de los cables debe ser mínima.

Las instalaciones subterráneas deberán ir dentro de un tubo no conductor y reforzado con un segundo tubo y un recubrimiento de hormigón en los casos de cruces de calzadas. Sobre la profundidad, estos cables deberán estar por debajo de los 0.9 metros en caso de ir por el acerado y de 1.1 metros en caso de seguir la calzada. Esta red no tiene una especificación de la distancia hacia otras redes, por lo que deberá respetar los requisitos de las demás redes.

Los elementos subterráneos que componen esta red son los siguientes:

Cables de alta tensión

Son los cables que se encargan de transportar la electricidad desde la planta generadora hasta el inicio de la población. Para corriente alterna, la cantidad de energía que conducen estos cables es de 1000 voltios y de 1500 en caso de corriente continua. Estos cables pueden pasar de estar en un medio subterráneo a torres de alta tensión, también llamados tramos aéreos.

Cables de media tensión

Estos cables inician en la linde de la población y se llevan a través de conductos hasta centros de distribución donde se transforma en baja tensión. Igual que en los cables de alta tensión, también pueden salir al exterior. En algunos tramos estos cables están trenzados para reducir las interferencias, esto es decisión de la empresa que hace la instalación. Soportan una tensión que varía entre 1 y 36 kV.

Cables de baja tensión

Son los cables que salen desde el centro de distribución hacia los hogares de los abonados. La tensión se reduce hasta los 400v.

Arqueta

Son elementos similares a los usados en la redes de saneamiento por donde cruzan cables de media y de baja tensión. En este caso, la arqueta debe de ser construida con hormigón armado.

Capítulo 3

Planificación y diseño

Nuestro proyecto será construido a través de un desarrollo iterativo e incremental. El proyecto está dividido en varias fases, que se repiten iterativamente cada vez que se produce una nueva mejora o funcionalidad en el sistema, comparándolo con la última versión del proyecto. Cada vez que el ciclo está terminado, se dice que se ha conseguido un nuevo incremento y deberemos volver al primer punto hasta que el proyecto esté listo para la fase de entrega y mantenimiento.

Para conseguir un buen resultado, estableceremos tres directivas que siempre debemos seguir:

- El software debe hacer lo que el cliente quiere que haga
- Aplicar los principios básicos de la programación orientada a objetos para añadir flexibilidad
- Esforzarse por un diseño reusable y mantenible

Una vez definida la metodología a usar, indicaremos cuales serán nuestras fases de desarrollo del software. Podemos ver los pasos que seguiremos en la Figura 3.1. Procedemos a explicar cada una de las fases a continuación:

- **Lista de características:** Una vez que el cliente nos proporciona la información necesaria sobre lo que quiere de su producto, debemos buscar las características de éste. Una característica no es más que una descripción a alto nivel de algo que

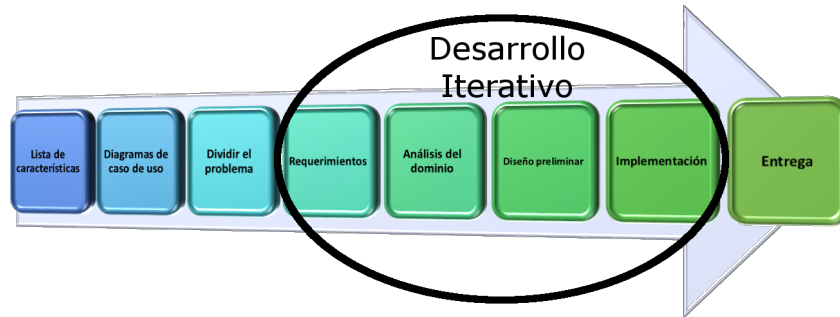


Figura 3.1: Fases del proceso de desarrollo del proyecto.

el sistema debe hacer. Buscaremos las características a través de las palabras del cliente y las pondremos en una lista.

- **Diagramas de caso de uso:** Un diagrama de caso de uso representa a través de nodos y de actores el uso del sistema. Para realizarlo nos debemos referir a la lista de características y crear un nodo equivalente por cada característica escrita.
- **Dividir el problema:** Analizando los casos de uso deberemos partir el problema en pequeñas piezas de funcionalidad, consiguiendo así un sistema modular, donde cada funcionalidad tendrá una sola responsabilidad. En este momento ya se podrán usar algunos patrones de diseño.
- **Requerimientos:** Escogeremos un nodo del diagrama de caso de uso, lo dividiremos en una serie de pasos en lenguaje formal y lo escribiremos en una lista, esta vez de forma concreta y especificando como funcionará el sistema, estableciendo entradas y salidas.
- **Análisis del dominio:** Buscaremos entre los requisitos nombres y verbos. Los nombres serán candidatos a ser clases del sistema, mientras que los verbos serán los métodos que las componen.
- **Diseño preliminar:** Creamos el diagrama de clases de lo extraído anteriormente, donde mostraremos las clases, sus métodos y atributos y cómo se relacionan cada una de ellas con el resto.

- **Implementación:** Pasamos a código el diagrama de clases creado anteriormente. Este es el momento de realizar pruebas a través de lo desarrollado.
- **Entrega:** El producto está terminado y puede ser entregado al cliente.

Dado que el tamaño de esta documentación sería inmenso si especificáramos todas las fases en cada iteración, se ha decidido mostrar el resultado final del diseño preliminar y una explicación de lo abordado en cada iteración.

Ya que hemos explicado como vamos a trabajar, podemos empezar a resolver el problema planteado.

3.1. Análisis y diseño

Empezaremos explicando brevemente en que consistirá el proyecto para conseguir mayor entendimiento de este apartado.

Utilizaremos una base de datos espacial para alojar la información. Esta información será almacenada en forma de polilíneas y de puntos. Dado que en la vida real, los objetos que representan estos datos están conectados, también guardaremos esta relación. Estas conexiones determinan la topología de la red.

Para mostrar la información de la base de datos, crearemos un sistema de información geográfico en tres dimensiones. Éste consultará a la base de datos y extraerá puntos y polilíneas. Los puntos serán tratados tal cual hemos recibido la información. Las polilíneas serán separadas en segmentos para poder realizar operaciones de actualización sobre cada punto de cada una de ellas. Posteriormente los representaremos en un mapa. Cada tipo de objeto tendrá una representación tridimensional que, incluyendo propiedades temáticas, conseguiremos un mayor nivel de realismo. Al software se le incorporará una interfaz de usuario. Con esto hacemos que el usuario se sienta más cómodo a la hora de usar nuestra aplicación.

Por último crearemos otro SIG3D, pero esta vez le añadiremos técnicas de realidad aumentada. Una vez posicionado y orientado el usuario podrá andar por la ciudad y observar las tuberías, arquetas, pozos y todo tipo de objetos que le rodean. Ésta parte

incluirá todas las opciones que posee el sistema de información geográfico comentado anteriormente.

3.1.1. Lista de características

Como anteriormente explicamos, debemos leer las instrucciones del cliente y crear una lista de características a alto nivel de lo que el sistema debe hacer.

SIG 3D Subterráneo

Lista de características

1. Los datos podrán ser visualizados mediante un sistema ubicuo.
2. Se deben visualizar los datos mediante realidad aumentada.
3. Se podrán realizar operaciones de creación, lectura, actualización y borrado de datos.
4. Existirán operaciones de análisis de los elementos subterráneos.
5. Edición de geometría.

3.1.2. Diagrama de caso de uso

Ya que tenemos la lista de características establecida, debemos diseñar uno o varios diagramas de caso de uso que represente todos los puntos que anteriormente comentamos.

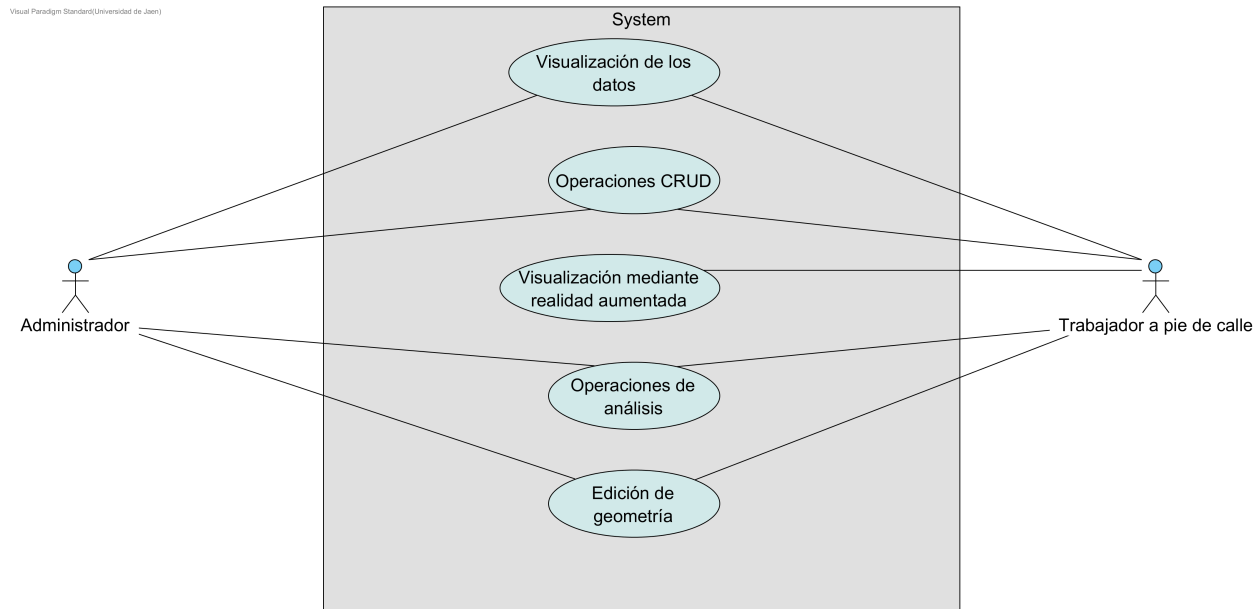


Figura 3.2: Raíz de los diagramas de caso de uso.

En la Figura 3.2 podemos ver que existen dos tipos de usuarios en el sistema. El primero será el administrador el cual podrá visualizar los datos, realizar operaciones de inserción, lectura, actualización y borrado de éstos, realizar operaciones de análisis sobre los datos y editar la geometría. Mientras tanto, el segundo actor de nuestro sistema será un trabajador a pie de calle, el cual, además de poder realizar las operaciones propias del administrador, también podrá visualizar los datos mediante realidad aumentada.

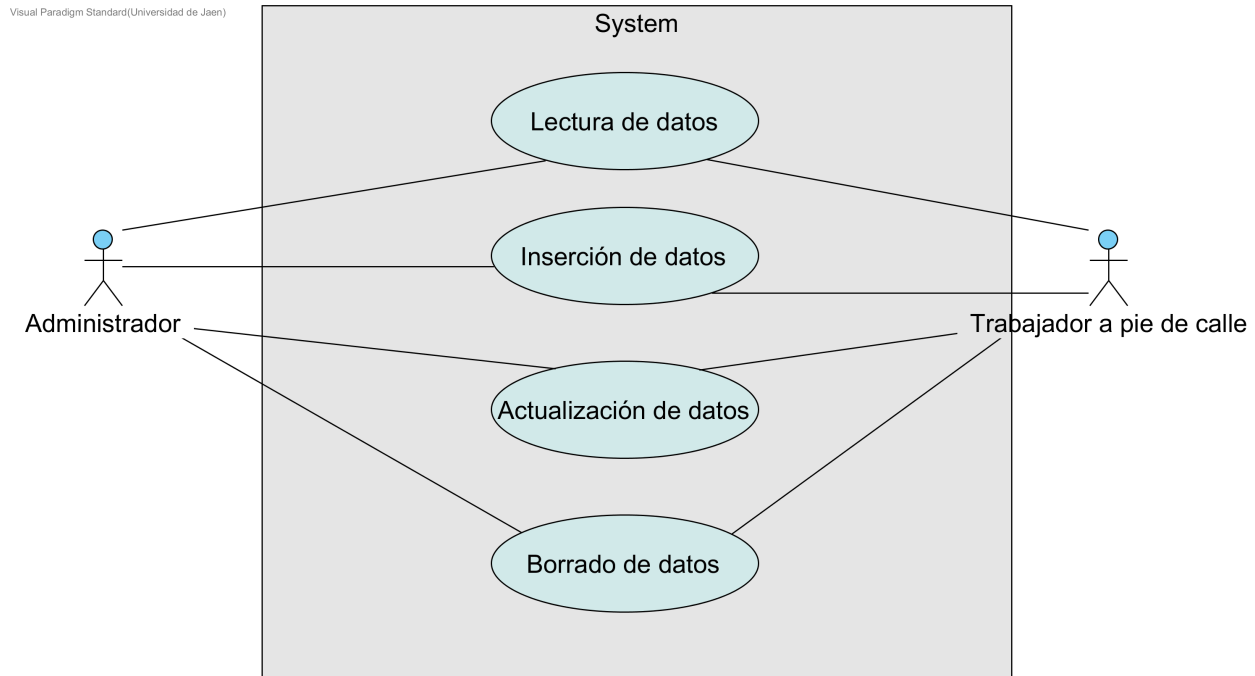


Figura 3.3: Diagrama de caso de uso "Operaciones CRUD".

El caso de uso identificado como "Operaciones CRUD" será dividido en varias partes, con el fin de dividir la operaciones y que se realicen como incrementos, en la Figura 3.3 vemos como quedaría desglosado este caso de uso.

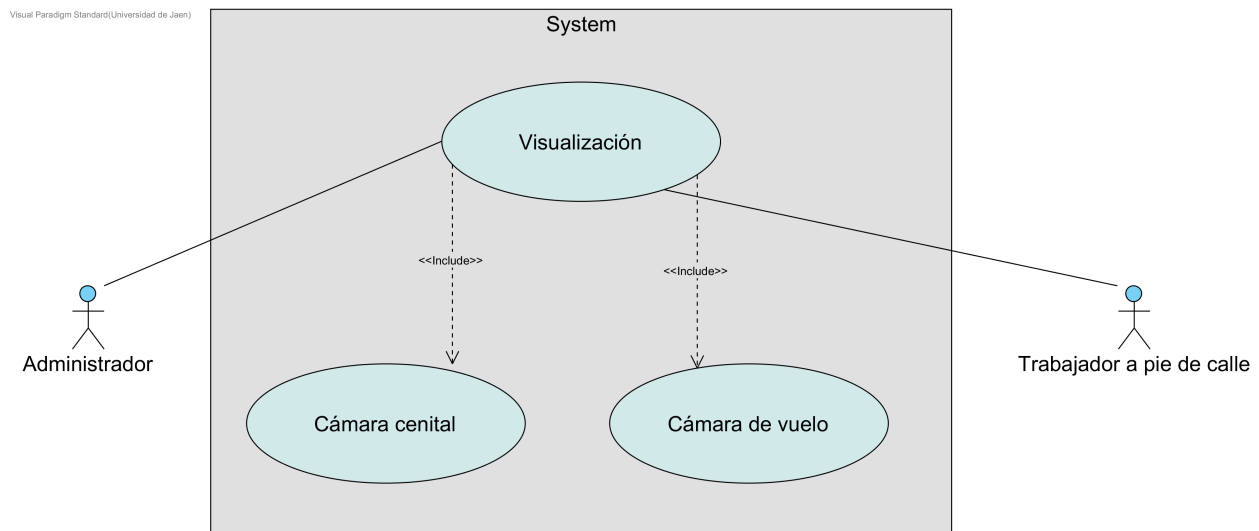


Figura 3.4: Diagrama de caso de uso "Visualización de los datos".

De igual manera, se ha decidido subdividir el caso de uso denominado «Visualización de los datos». Éste caso de uso estará compuesto de la visualización de los datos y de la

creación de las cámaras que hacen posible moverse a través de la escena. Vemos este caso de uso en la Figura 3.4.

3.1.3. Dividir el problema

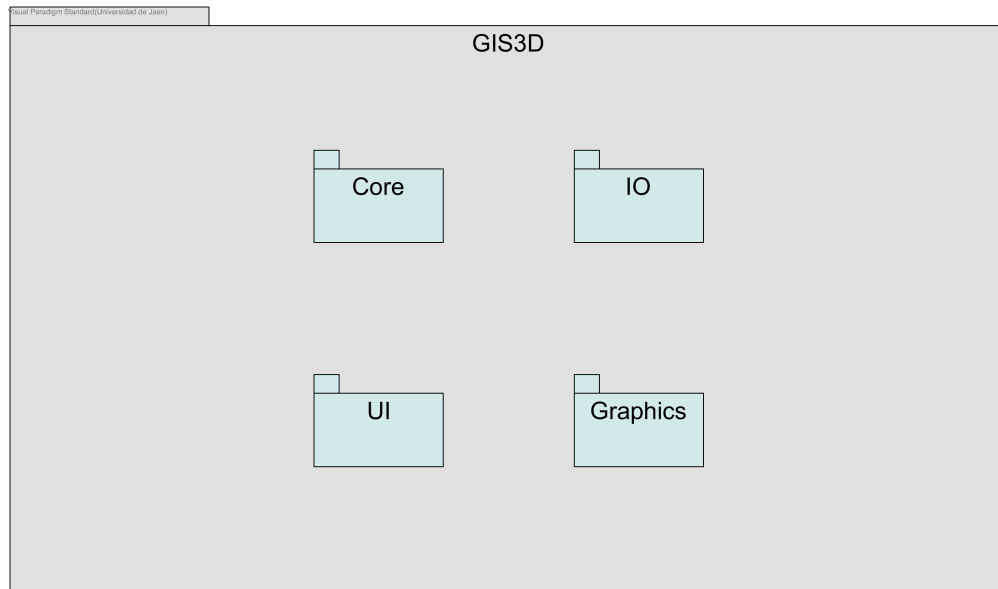


Figura 3.5: Diagrama de paquetes.

Ahora es el momento de dividir nuestro gran problema en partes más pequeñas. Con esto conseguimos modularidad en nuestra aplicación, haciendo así que los incrementos sean pequeños pero asegurándonos de que funcionen como el cliente quiere que lo haga.

Como podemos ver en la Figura 3.5, hemos creado cuatro módulos dentro de nuestra aplicación:

- **GIS3D:** Este módulo es el encargado de comunicar unas partes de la aplicación con el resto.
- **Core:** En él se encuentra la funcionalidad básica de todo SIG como son las capas y los modelos.
- **Graphics:** Es el módulo encargado de la visualización de los datos establecidos en el módulo Core.

- **IO:** Se encarga de acceder a la base de datos para realizar las operaciones CRUD.
- **UI:** Engloba las clases que definen la funcionalidad de cada panel de la interfaz de usuario.

Tras esta fase podremos empezar a iterar sobre nuestro problema y solucionandolo con cada pequeño incremento.

3.1.4. Clases

Como anteriormente se especificó, las clases creadas estarán divididas en paquetes. En la Figura 3.6 vemos el diagrama de clases completo. En el se muestran las clases que se han creado para la consecución del proyecto y sus relaciones. Debido a su gran tamaño se ha decidido explicar estas clases dividiendolas en sus paquetes. Cada paquete mostrará las clases que están contenidas. Haremos una explicación del paquete en general, mostraremos un diagrama de menor tamaño que el diagrama global y definiremos cada una de las clases junto a sus atributos.

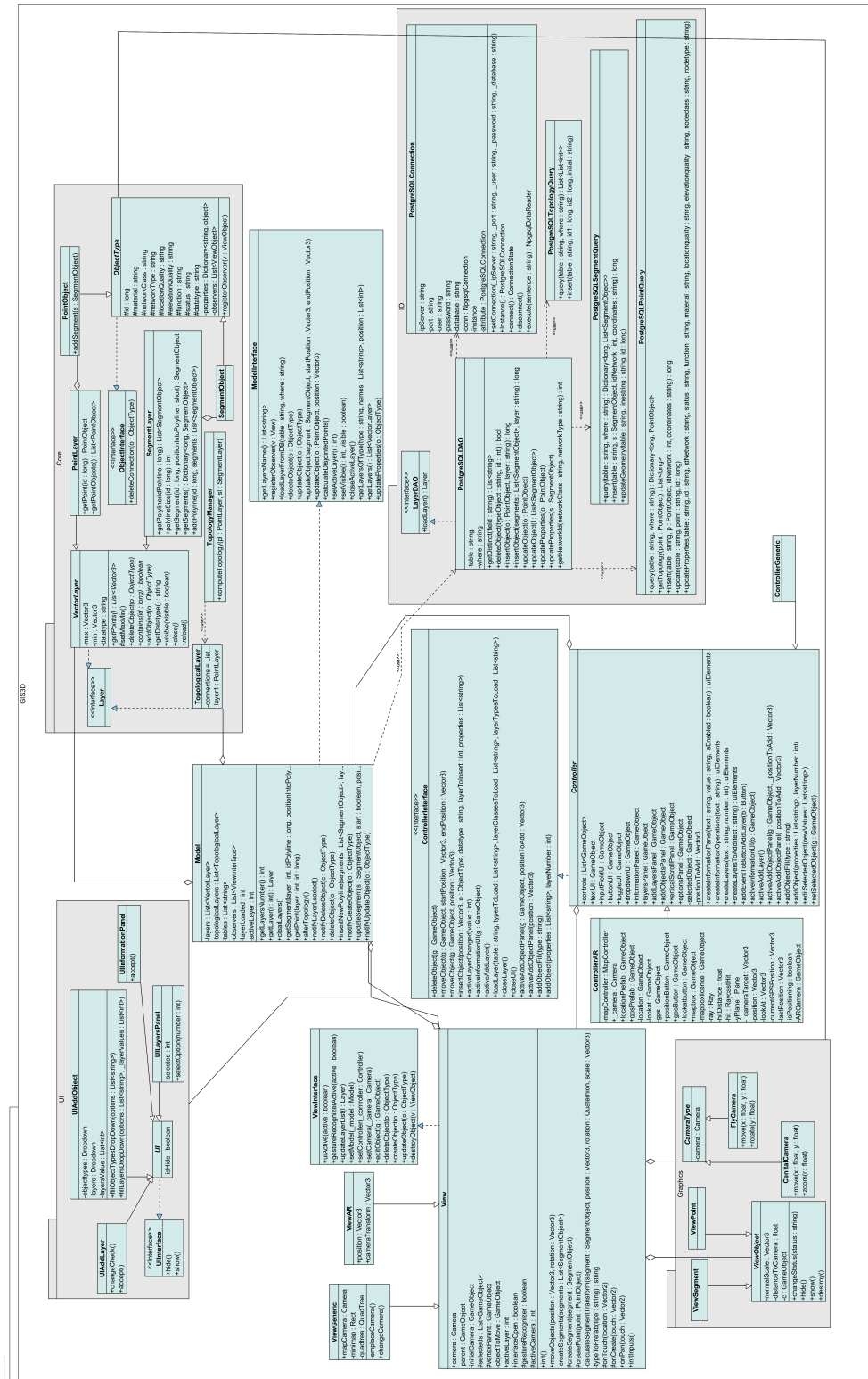


Figura 3.6: Diagrama de clases.

GIS3D

Este paquete es el base para todas las clases y paquetes de nuestro software. En él encontramos todo lo necesario para la creación de un SIG 3D. Dentro de él también nos encontramos a las clases que forman el patrón de diseño Modelo-Vista-Controlador. Estas clases serán las encargadas de tratar con los datos y se comunicarán para realizar las acciones seleccionadas por el usuario. El diagrama de clases se muestra en la Figura 3.7. A continuación definimos cada una de estas clases:

- **ControllerInterface**: Interfaz que define los métodos que estrictamente debe de tener un controlador de esta aplicación.

- Métodos:

- **void deleteObject**(*GameObject g*): Método invocado por la vista tras la decisión del usuario de eliminar un objeto.
- **void moveObject**(*GameObject g, Vector3 startPosition, Vector3 endPosition*): Método invocado por la vista tras la decisión del usuario de cambiar la geometría de un objeto de tipo segmento.
- **void moveObject**(*GameObject g, Vector3 position*): Método invocado por la vista tras la decisión del usuario de cambiar la posición de un objeto de tipo puntual.
- **void insertObject**(*Vector3 position, ObjectType o, string datatype, int layerToInsertList<string> properties*): Crea un objeto en la posición indicada, dentro de la capa proporcionada por parámetro.
- **void activeLayerChanged**(*int value*): Activa el panel de las capas abiertas, pasando por parámetro la capa activa en ese momento.
- **void activeInformationUI**(*GameObject g*): Activa el panel de información, mostrando los atributos y operaciones a realizar sobre el objeto seleccionado por el usuario.
- **void activeAddLayer**(): Activa el panel de carga de una nueva capa.

- **void loadLayer**(*string table, List<string> typesToLoad, List<string> layerClassesToLoad, List<string> layerTypesToLoad*): Formatea el condicionante a través del contenido de las listas *typesToLoad*(tipo de objeto), *layerClassesToLoad*(clase de capa) y *layerTypesToLoad*(tipos de capas), para cargar una nueva capa.
 - **void closeLayer**(): Cierra una capa y a partir de ese momento no existirá en el plano.
 - **void closeUI**(): Cierra la interfaz de usuario.
 - **void activeAddObjectPanel**(*GameObject g, Vector3 positionToAdd*): Activa el panel de creación de un nuevo objeto. Almacenamos la posición y al objeto que estará topologicamente conectado.
 - **void activeAddObjectPanel**(*Vector3 positionToAdd*): Activa el panel de creación de un nuevo objeto. Almacenamos la posición ya que no estará conectado con ningún objeto de la escena.
 - **void addObjectFill**(*string type*): Proporciona a la interfaz información sobre el objeto seleccionado en ese momento.
 - **void addObject**(*List<string> properties, int layerNumber*): Recibe de la interfaz las propiedades que tendrá el objeto a crear.
- **Controller**: Clase abstracta que implementa algunos de los métodos indicados en *ControllerInterface* y que serán comunes para todos los controladores
- Atributos:
 - **GameObject informationPanel**: Referencia al panel de información de objetos.
 - **GameObject layersPanel**: Referencia al panel de capas.
 - **GameObject addLayersPanel**: Referencia al panel de carga de capas.
 - **GameObject optionsPanel**: Referencia al panel de opciones.
 - **GameObject selectedObject**: Objeto que actualmente está seleccionado y sobre el cual se realizará una acción.

- **GameObject positionToAdd**: Posición sobre la que se añadirá un objeto.
- Métodos:
 - *uiElements* **createInformationPanel**(*string text, string value, boolean isEnabled*): Crea un control personalizado que será utilizado en el panel de información.
 - *uiElements* **createInformationOperations**(*string text*): Crea un control personalizado que será utilizado en el panel de información y que al pulsarlo desencadenará una acción.
 - *uiElements* **createLayers**(*string text, int number*): Crea un control personalizado que será utilizado en el panel de capas cargadas.
 - *uiElements* **createLayersToAdd**(*string text*): Crea un control personalizado que será utilizado en el panel de carga de capas.
 - *void* **addEventToButtonAddLayer**(*Button b*): Crea el evento de un botón encargado de cambiar la información del panel de carga de capas. Al pulsar uno, los tipos que se podrán cargar en ese momento serán distintos.
 - *void* **editSelectedObject**(*List<string> newValues*): Recibe la información sobre un objeto que ha sido editado.
 - *void* **setSelectedObject**(*GameObject g*): Establece como seleccionado el objeto recibido.
- **ControllerGeneric**: Controlador genérico para un SIG 3D.
- **ControllerAR**: Controlador encargado de realizar tareas de un SIG 3D con realidad aumentada.
 - Atributos:
 - **MapController mapController**: Objeto encargado de almacenar los objetos virtuales que forman el mapa tipo "Google Maps".
 - **Camera camera**: Cámara utilizada en realidad aumentada.
 - **GameObject locationPrefab**: Objeto que indica la posición seleccionada por el usuario.

- **GameObject gpsPrefab**: Objeto que indica la posición GPS del usuario.
- **GameObject positionButton**: Botón que pulsa el usuario para establecer su posición.
- **GameObject gpsButton**: Botón que pulsa el usuario para establecer su posición como la obtenida desde el GPS.
- **GameObject lookatButton**: Botón que pulsa el usuario para indicar que ha terminado la orientación.
- **GameObject mapbox**: Referencia al objeto que nos proporciona los mapas.
- **GameObject mapboxLicence**: Licencia de uso del objeto mapbox.
- **GameObject worldRoot**: Objeto en el que almacenaremos nuestros objetos virtuales.
- Métodos:
 - *void retrieveGPSData()*: Función invocada cada tres segundos que recibe la posición obtenida por el sensor GPS.
 - *void establishPosition()*: Establece la posición seleccionada por el usuario.
 - *void establishLookAt()*: Establece la orientación procesada por el usuario.
 - *void useGPSPosition()*: Establece la posición obtenida por el GPS como situación del usuario.
 - *void enableAR()*: Habilita el uso de la realidad aumentada.
- **ModelInterface**: Interfaz que define los métodos que estrictamente debe de tener un modelo de esta aplicación.
 - Métodos:
 - *void loadLayerFromDB(string table, string where)*: Carga una capa desde una base de datos, proporcionando la tabla y la sentencia condicional.
 - *void deleteObject(ObjectType o)*: Borra un objeto de la base de datos.

- **void updateObject(ObjectType o):** Actualiza las propiedades de un objeto.
 - **void updateObject(SegmentObject s, Vector3 startPosition, Vector3 endPosition):** Actualiza la posición de un objeto de tipo segmento.
 - **void updateObject(PointObject p, Vector3 position):** Actualiza la posición de un objeto de tipo puntual.
- **Model:** Clase que funciona como modelo y almacena las capas cargadas
- **Atributos:**
 - **List<VectorLayer> layers:** Conjunto de capas vectoriales cargadas.
 - **List<TopologicalLayer> topologicalLayers:** Conjunto de capas topológicas.
 - **List<string> tables:** Conjunto de tablas usadas para cargar las capas del vector layers.
 - **List<ViewInterface> observers:** Vistas suscritas a las notificaciones que se enviarán al crear, modificar, cargar o borrar objetos.
 - **int layerLoaded:** Identificador de la última capa cargada.
 - **int activeLayer:** Capa actualmente activa.
 - **Métodos:**
 - **SegmentObject getSegment(int layer, long idPolyline, short positionIntoPolyline):** Función que devuelve un segmento a través del identificador de la polilínea y de su posición dentro de ella perteneciente a la capa indicada.
 - **PointObject getPoint(int layer, long id):** Función que devuelve un punto a través del identificador perteneciente a la capa indicada.
 - **void alterTopology():** Computa la topología de los objetos cargados.
 - **void insertNewPolyline(List<SegmentObject> segments, int layerIndex, ObjectType s):** Función que crea una nueva polilínea.
- **ViewInterface:** Interfaz que define los métodos que estrictamente debe de tener una vista de esta aplicación.

- Métodos:
 - **void uiActive**(*bool active*): Recibe el estado actual de la interfaz de usuario, si está activa no se procesarán los eventos de movimiento de cámara.
- **View**: Clase abstracta que implementa algunos de los métodos indicados en *ViewInterface* y que serán comunes para todas las vistas.
 - Atributos:
 - **Camera camera**: Cámara activa en este momento.
 - **GameObject initialCamera**: Cámara que se utilizará al inicio de la aplicación.
 - **List<GameObject> selecteds**: Objetos que han quedado seleccionados por el usuario.
 - **GameObject vertexParent**: Objeto auxiliar que almacenará los indicadores de que un objeto puede ser trasladado.
 - Métodos:
 - **void createSegment**(*SegmentObject segment*): Crea la representación virtual de un segmento.
 - **void createSegments**(*List<SegmentObject> segments*): Crea los segmentos que están dentro de la lista de segmentos.
 - **void createPoint**(*PointObject point*): Crea la representación virtual de un objeto puntual.
 - **void calculateSegmentTransform**(*SegmentObject segment, Vector3 position, Quaternion rotation, Vector3 scale*): Cálculo de las transformaciones que sufrirá un segmento para que su representación sea conforme a sus propiedades.
 - **GameObject typeToPrefab**(*string type*): A través del tipo de objeto.
 - **void OnTouch**(*Vector2 location*): Evento que sucederá cuando el usuario toque la pantalla.

- *void* **OnCreate**(*Vector2 touch*): Evento que sucederá cuando el usuario quiera crear un objeto en la posición pulsada.
- *void* **OnPan**(*Vector2 touch*): Evento que sucederá cuando el usuario arrastre el apuntador por la pantalla .
- **ViewGeneric**: Vista genérica para un SIG 3D.
 - Atributos:
 - **Camera mapCamera**: Cámara que mostrará los datos en forma de mini-mapa.
 - **QuadTree quadtree**: Estructura de datos donde se almacenarán los datos y que ayudará a la representación de los objetos en el minimapa.
 - Métodos:
 - *void* **emplaceCamera**(): Situa la cámara en el centro de la escena teniendo en cuenta las capas cargadas.
 - *void* **changeCamera**(): Cambia el tipo de cámara activa.
- **ViewAR**: Vista para un SIG 3D con realidad aumentada.
 - Atributos:
 - **Vector3 position**: Posición inicial del usuario respecto al mapa.
 - **Vector3 cameraTransform**: Rotación en forma de ángulos de Euler sufrida por la cámara.
 - **Vector3 cameraPosition**: Posición actual de la cámara respecto a la posición inicial del usuario.

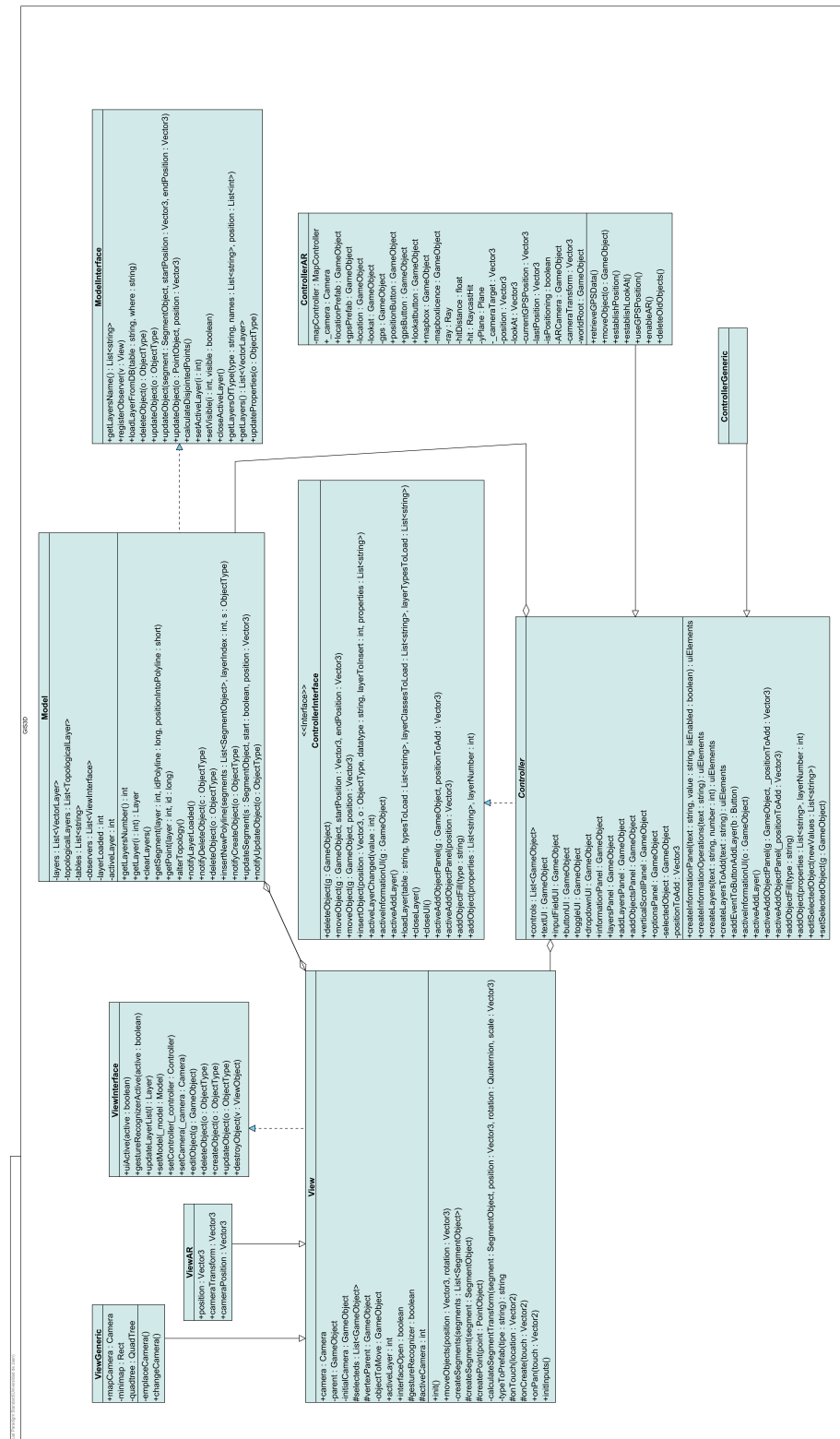


Figura 3.7: Diagrama de clases del paquete GIS3D.

IO

En este paquete se encuentran todas las clases relacionadas con el sistema de almacenamiento. En este caso, están las clases que realizan consultas SQL a nuestra base de datos. Para ello utilizaremos el patrón DAO (Data acces object). Así conseguimos separar la carga de datos de su representación en memoria.

- **LayerDAO**: Interfaz que deberá implementar todo sistema de carga de datos sin importar su naturaleza.

- Métodos:

- **Layer loadLayer()**: Carga una capa desde el sistema de almacenamiento elegido.

- **PostgreSQLDAO**: Realiza consultas a una base de datos de tipo PostgreSQL.

- Atributos:

- **string table**: Tabla sobre la que se tratarán los datos.
- **string where**: Condición sobre la consulta SQL.

- Métodos

- **List<string> getDistinct(string field)**: Ejecuta una consulta donde se devuelve los valores sin repetir del campo indicado por parámetro.
- **bool deleteObject(string typeObject, int id)**: Borra un objeto y devuelve si el resultado ha sido satisfactorio.
- **void insertObject(PointObject o, string table)**: Inserta un objeto de tipo puntual en la tabla especificada.
- **void insertObject(List<SegmentObject> segments, string table)**: Inserta una nueva polilínea en la tabla especificada.
- **void updateObject(PointObject o)**: Actualiza un objeto de tipo puntual.
- **void updateObject(List<SegmentObject> segments)**: Actualiza una polilínea.

- **PostgreSQLTopologyQuery**: Consulta a las tablas de la base de datos que contienen topología.

- Métodos

- *List<List<int>>* **query**(*string table, string where*): Extrae la topología de la tabla indicada y devuelve los identificadores de los objetos conectados, siendo el primero de cada lista un objeto de tipo puntual. Por ejemplo, [0] = 1394 <- Pozo [1] = 3421 <- Tubería.
- *void* **insert**(*string table, long id1, long id2, string initial*): Inserta en la tabla pasada como parámetro los identificadores de dos objetos que están conectados, por el inicio o por el final.

- **PostgreSQLPointQuery**: Consulta las tablas de la base de datos que contienen objetos de tipo puntual para realizar operaciones con ellos.

- Métodos

- *Dictionary<long, PointObject>* **query**(*string table, string where*): Devuelve los objetos puntuales de una tabla que cumplan con la cláusula condicional.
- *long* **insert**(*string table, PointObject p, int idNetwork, string coordinates*): Inserta un objeto puntual en la tabla indicada.

- **PostgreSQLSegmentQuery**: Consulta las tablas de la base de datos que contienen objetos de tipo segmento para realizar operaciones con ellos.

- Métodos

- *Dictionary<long, PointObject>* **query**(*string table, string where*): Devuelve los segmentos de una tabla que cumplan con la cláusula condicional.

- **PostgreSQLConnection**: Clase que implementa el patrón Singleton. Realiza las conexiones con la base de datos.

- Atributos:

- **string ipServer**: Dirección IP del servidor donde está alojado el servidor de base de datos.
- **string port**: Puerto en el que se sitúa el servidor de base de datos.
- **string user**: Usuario que realizará la conexión.
- **string password**: Contraseña usada para la autenticación del usuario.
- **string database**: Base de datos a la que se conectará el usuario.
- **NpgsqlConnection conn**: Conexión a la base de datos.
- Métodos
 - *NpgsqlDataReader* **execute**(*string sentence*): Ejecuta la sentencia SQL y devuelve los resultados en forma de lista.

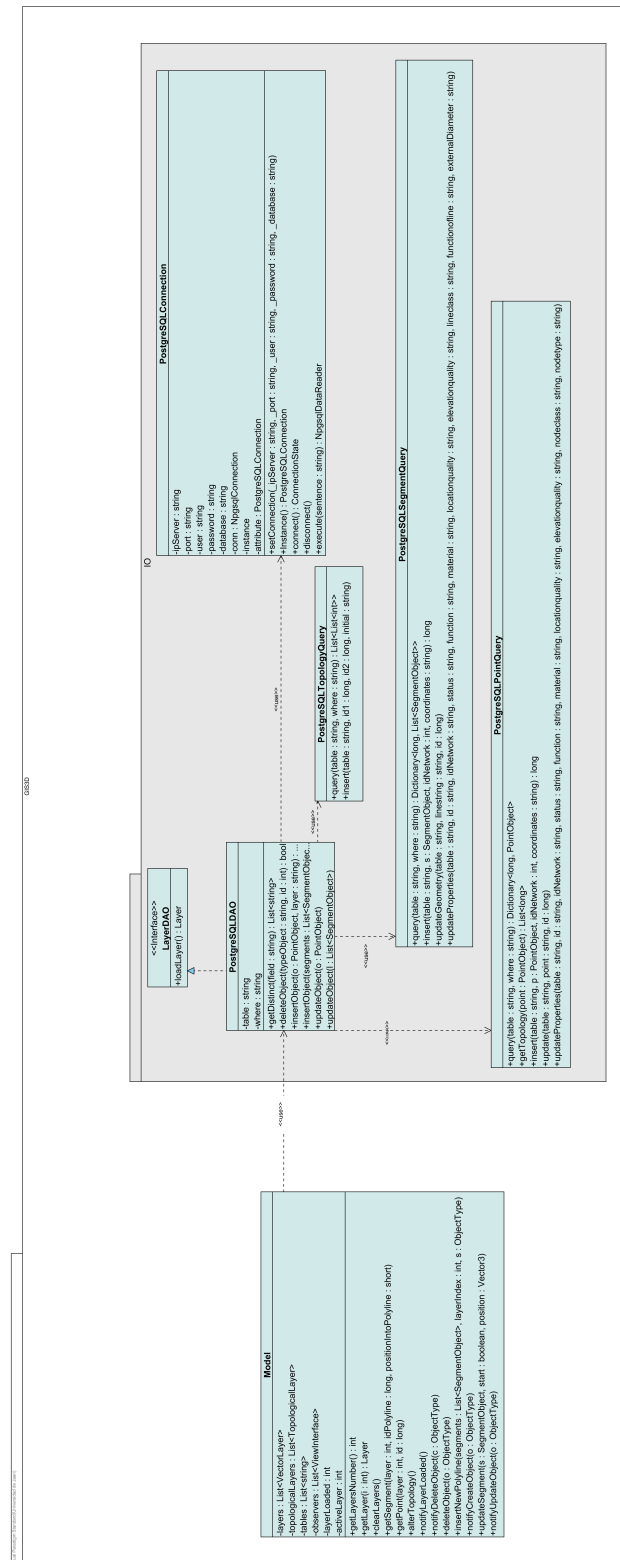


Figura 3.8: Diagrama de clases del paquete IO.

Graphics

Clases encargadas de la representación de los objetos cargados por el usuario.

- **ViewObject**: Clase abstracta que sirve de base para la representación de los modelos.
 - Atributos:
 - **GameObject c**: Objeto virtual que representa un elemento del subsuelo como pozos o conductos.
- **ViewSegment**: Clase que representa un objeto de tipo segmento.
- **ViewSegment**: Clase que representa un objeto de tipo puntual.
- **CameraType**: Clase abstracta que sirve de base para la definición de una cámara.
 - Atributos:
 - **Camera camera**: Referencia a la cámara que usa un objeto instanciado de esta clase.
- **CenitalCamera**: Tipo de cámara que proporciona una visualización tipo 2D.
- **FlyCamera**: Cámara libre, donde el usuario se mueve por la escena a través de los ejes XYZ.

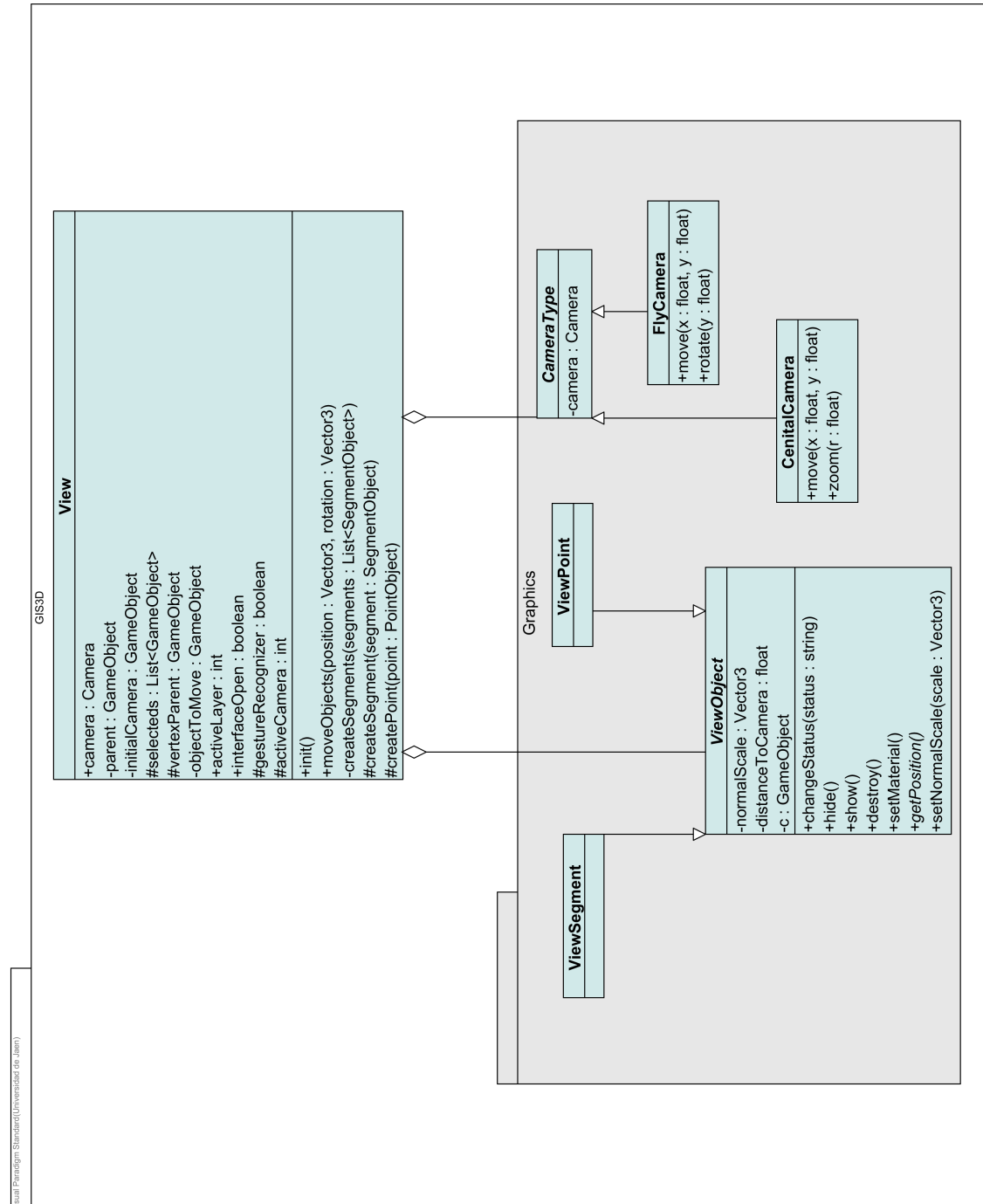


Figura 3.9: Diagrama de clases del paquete Graphics.

Core

En este paquete se encuentran los objetos claves de todo SIG, como son las capas o los objetos que contienen a éstas.

- **Layer**: Interfaz que define los métodos que estrictamente debe tener una capa de esta aplicación.
- **TopologicalLayer**: Capa que almacena la conexión existente entre una capa puntual y otra de segmentos.
 - Atributos:
 - **List<List<int> > connections**: Lista de identificadores cuya primera posición es el identificador del objeto puntual y el resto de contenido son los identificadores de las tuberías conectadas a este objeto.
 - **PointLayer pl**: Capa que contiene objetos puntuales, cómo arquetas.
 - **SegmentLayer sl**: Capa que contiene segmentos, cómo tuberías.
- **TopologyManager**: Clase encargada de asignar la topología entre los objetos de tipo puntual y los de tipo segmento.
 - Métodos:
 - **void computeTopology(PointLayer pl, SegmentLayer sl)**: Tras una consulta a la tabla donde se almacena la topología, asigna la vecindad de cada uno de los objetos.
- **VectorLayer**: Clase abstracta que representa una capa cuya función es almacenar objetos cuya naturaleza es vectorial.
 - Atributos:
 - **Vector3 max**: Punto máximo de la capa.
 - **Vector3 min**: Punto mínimo de la capa.
 - **string datatype**: Tipo de datos que contiene la capa.

- Métodos:

- *List<Vector3>* **getPoints()**: Lista de todos los puntos de la capa.
- *boolean* **contains(long id)**: Función que comprueba si el objeto con identificador similar al pasado por parámetro pertenece a la capa.
- *void* **visible(bool vis)**: Oculta y muestra una capa.
- *void* **close()**: Cierra una capa, eliminando todos los objetos relacionados con ella.
- *void* **reload()**: Vuelve a cargarse la capa.

- **PointLayer**: Clase que almacena objetos de tipo puntual.

- Atributos:

- **Dictionary<long, PointObject> points**: Almacena los objetos puntuales en un mapa usando como índice el identificador del punto.

- **SegmentLayer**: Clase que almacena objetos de tipo segmento.

- Atributos:

- **Dictionary<long, List<SegmentObject> segments**: Almacena los segmentos en forma de polilínea en un mapa usando como índice el identificador de la polilínea y los segmentos en una lista ordenada por su posición.

- **ObjectType**: Clase abstracta que sirve de base para el resto de objetos del sistema.

- Atributos:

- **long id**: Identificador único del objeto.
- **string material**: Material del que está compuesto el objeto.
- **string networkClass**: Clase de capa a la que pertenece el objeto.
- **string networkType**: Tipo de capa a la que pertenece el objeto.
- **string locationQuality**: Indica la calidad de los datos referentes a la localización del objeto en el plano XY.

- **string elevationQuality**: Indica la calidad de los datos referentes a la localización del objeto en el plano Z.
- **string function**: Función del objeto.
- **string status**: Estado del objeto.
- **Dictionary<string, object> properties**: Mapa donde se guardan las propiedades que no son comunes para todos los tipos de objetos que hereden de esta clase.
- Métodos:
 - *void hide()*: Oculta un objeto.
 - *void show()*: Muestra un objeto.
 - *void delete()*: Borra un objeto de la memoria y desconecta su topología.
- **PointObject**: Clase que almacena las propiedades de un objeto de tipo puntual.
 - Métodos:
 - *void addSegments(SegmentObject s)*: Añade segmentos que están conectados a este objeto.
- **SegmentObject**: Clase que almacena las propiedades de un objeto de tipo segmento.

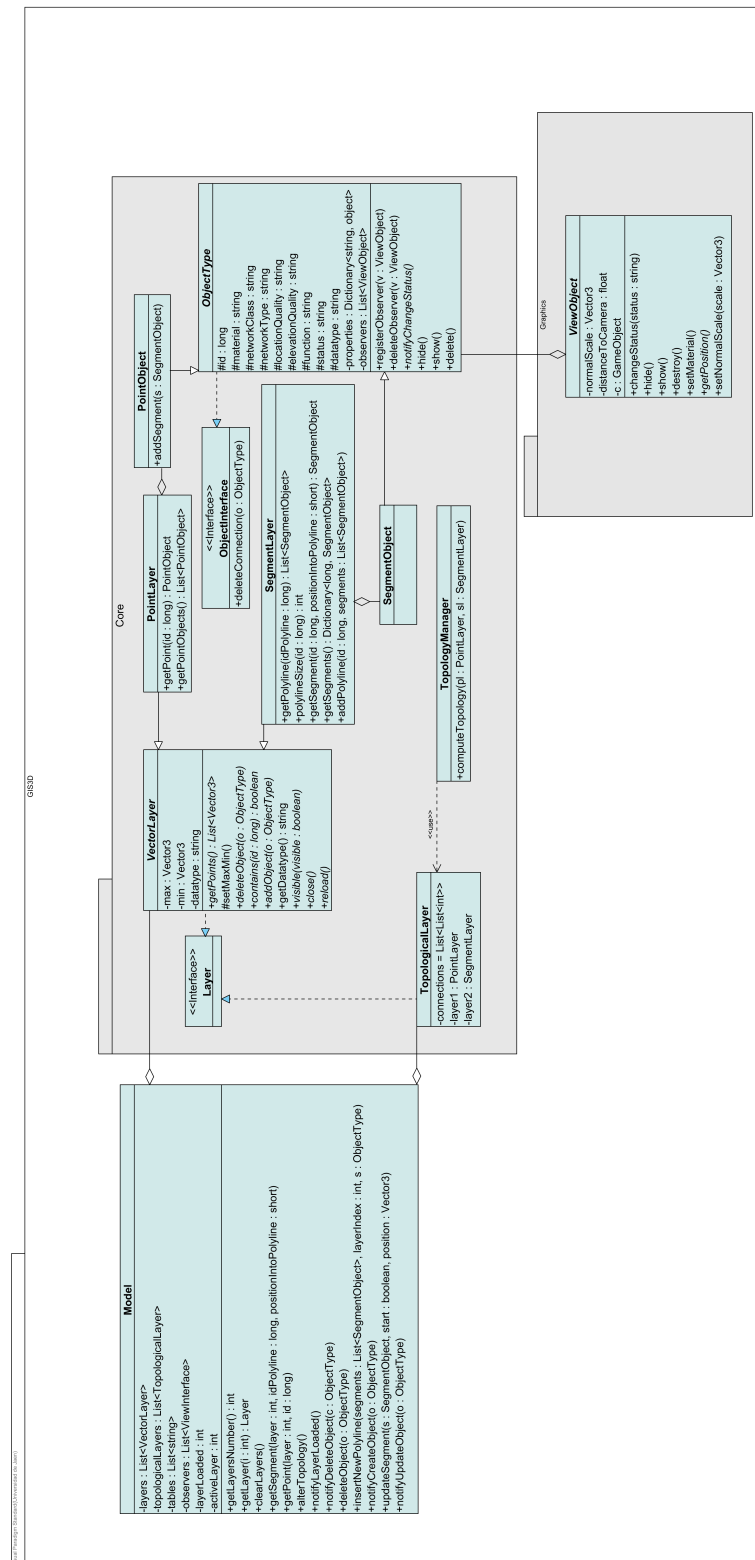


Figura 3.10: Diagrama de clases del paquete Core.

UI

En este paquete se encuentran las clases que proporcionan la funcionalidad a los elementos de la interfaz de usuario.

- **UIInterface**: Interfaz que define los métodos que estrictamente deben implementar las clases encargadas de la interfaz de usuario.
- **UI**: Clase abstracta que implementa alguno de los métodos que son comunes a todas las clases pertenecientes a la UI.
- **UILayersPanel**: Controla las tareas de la interfaz de usuario relacionadas con las capas cargadas.
- **UIInformationPanel**: Muestra la información del objeto seleccionado por el usuario y da la posibilidad de realizar operaciones con el mismo.
- **UIAddObject**: Formulario para la creación de un nuevo objeto.
- **UIAddLayer**: Muestra las capas disponibles para cargar y envía al controlador la selección del usuario.

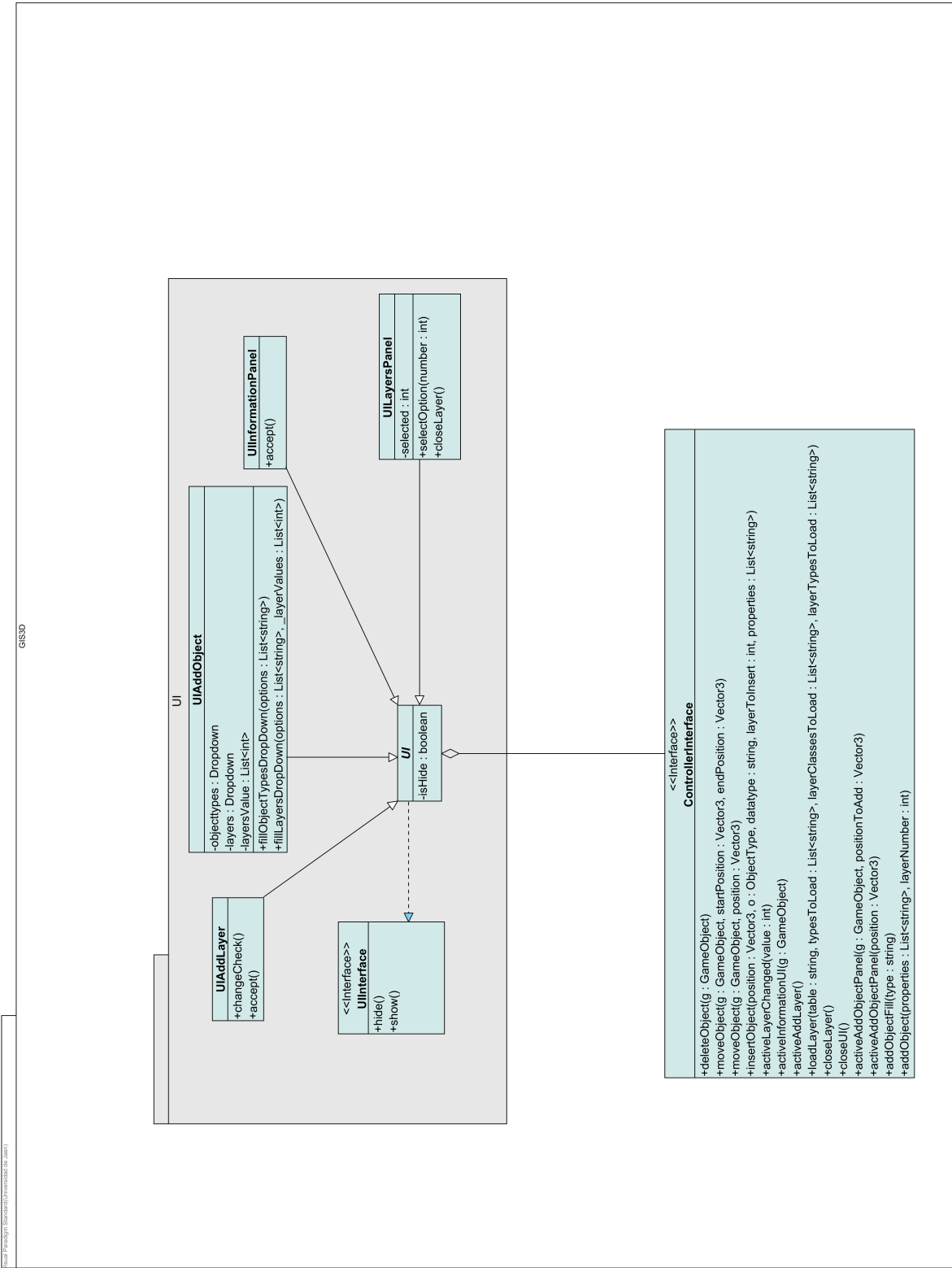


Figura 3.11: Diagrama de clases del paquete UI.

3.1.5. Base de datos

Una vez vista la Sección 1.4 donde se da una justificación del sistema de base de datos escogido, es decir PostgreSQL con la extensión PostGIS, procederemos a estudiar qué tipo de datos nos proporciona el sistema y cómo podemos adaptarlo a los datos vistos en el Capítulo 2. Para ello nos ayudaremos de la documentación de la web de PostGIS[Posa]. Claramente podemos identificar cómo los conductos de todo tipo de redes se asemejan al comportamiento del tipo de dato “Linestring”, ya que como los segmentos de estos conductos se interconectan unos con otros para formar una polilínea que tiene un inicio y un final. Ahora nos queda determinar de qué forma podemos representar los pozos, arquetas, caudalímetros y demás objetos que solo tienen una posición en el espacio. En este caso tenemos dos posibilidades, escoger el tipo “PolyhedralSurface”, representar y almacenar su forma en la base de datos o representar el dato como tipo “Point” y guardar sus propiedades de tamaño y forma en una tabla aparte para después utilizar una representación a la que aplicarle transformaciones visuales. La primera opción es demasiado compleja ya que para un GIS lo importante no es cómo se representa un dato, sino su posición en el espacio, por eso nos hemos decantado por la segunda opción.

No hay que olvidar que las polilíneas están delimitadas por ningún o por un objeto de tipo vértice, por lo tanto esto también es necesario contemplarlo en la base de datos. De igual manera, en la red de abastecimiento también sucede que las tuberías se conectan entre ellas, por lo tanto esto también será necesario almacenarlo. Esto es lo que define la topología de la red y es importante a la hora de hacer operaciones con los datos.

Pero no solo debemos almacenar la geometría de los objetos, también será necesario saber sus propiedades temáticas. Con esto podremos aumentar el realismo del sistema.

En la Figura 3.12 observamos la estructura de nuestro diseño de la base de datos. No se trata de un diseño tradicional de una base de datos ya que PostgreSQL es capaz de realizar herencia sobre tablas. Una tabla heredará todos los atributos del padre. Al realizar una operación sobre una tabla hija, los resultados también afectarán a la tabla padre.

A continuación explicaremos las tablas que hemos creado junto a sus atributos

- **network**: Representa un conjunto de objetos interconectados en forma de red.
 - *int id Primary Key*: Identificador único de una red.
 - *text name*: Nombre de la red.
 - *text owner*: Empresa propietaria de la red.
 - *networkClass name SET(Liquid, Gas, Solid, Electrical, Optical)*: Tipo de material que transporta la red.
 - *text type*: Tipo de la red (teléfono, saneamiento, abastecimiento...).
- **networkfeature**: Representa cualquier objeto perteneciente a una red, sin importar su naturaleza.
 - *int id Primary Key*: Identificador único de un objeto.
 - *statusType status SET(Unknown, OutOfService, tempOutOfService, InUse)*: Estado actual del elemento de la red (funciona, ya no se utiliza, no se utiliza pero es usable...).
 - *text function*: Función del objeto (transporte, diversificación, acometida...).
 - *text material*: Material del que está compuesto el elemento.
 - *spatialQualityType locationQuality SET(unknown, surveyed, calculated, interpolated, estimated, assumed)*: Indica como ha sido tomada la medida de la posición del objeto en el eje XY.
 - *spatialQualityType elevationQuality SET(unknown, surveyed, calculated, interpolated, estimated, assumed)*: Indica como ha sido tomada la posición del objeto entorno al eje Z.
 - *int idnetwork Foreign Key References network.id*: Identificador de la red a la que pertenece este elemento.
- **node**: Representación de los objetos que solo necesitan una posición en el espacio para ser referenciados correctamente, por ejemplo, pozos, válvulas...

- *nodeClass* **class** *SET(ComplexFunctional, SimpleFunctional, Terminal, StorageDevice, MeasurementDevice, ControllerDevice, TechDevice, AnyDevice)*: Tipo de objeto representado (de funcionalidad compleja, de almacenaje, terminal, de medida, ...).
 - *text* **type** : Nombre del objeto (pozo, válvula, arqueta,...).
 - *point* **geom** : Posición en el espacio del objeto.
- **distributionelement**: Representa los elementos de distribución como son las tuberías, túneles...
- *lineClass* **class** *SET(unknown, MainLine, TransportLine, SupplyLine, HouseServiceLine)*: Tipo del elemento de distribución (transporte, acometida, arterial).
 - *text* **functionOfLine**: Función del elemento de distribución (flujo constante, de retorno, de condensación).
 - *linestring* **geom**: Geometría de la tubería en formato polilínea, es decir, en el siguiente formato:(x1 y1 z1, x2 y2 z2, x3 y3 z3, ...).
- **pipe**
- *float* **externalDiameter**: Diámetro exterior de la polilínea.
- **connected**: Almacen de la topología de los objetos
- *int* **idpipe**: Identificador único de una tubería
 - *int* **idnode**: Identificador único de un nodo
 - *char* **isInitial**: Indica si la conexión entre la tubería y el nodo es por el principio o por el final

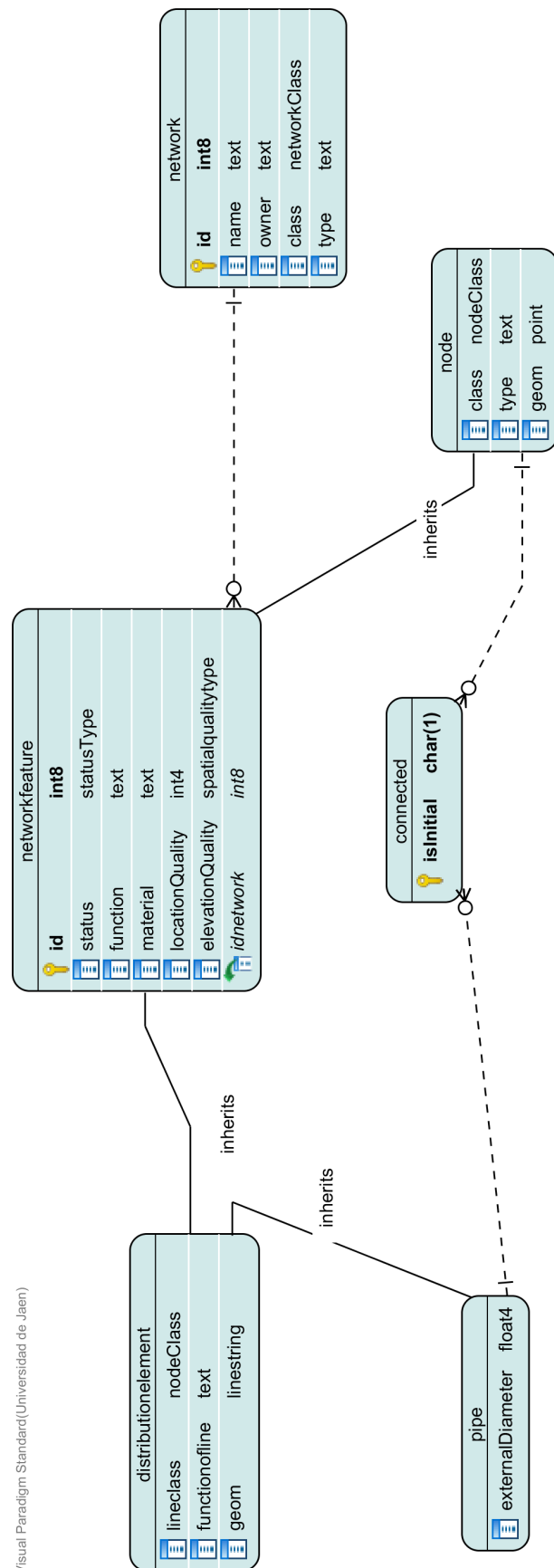


Figura 3.12: Diagrama Entidad Relación.

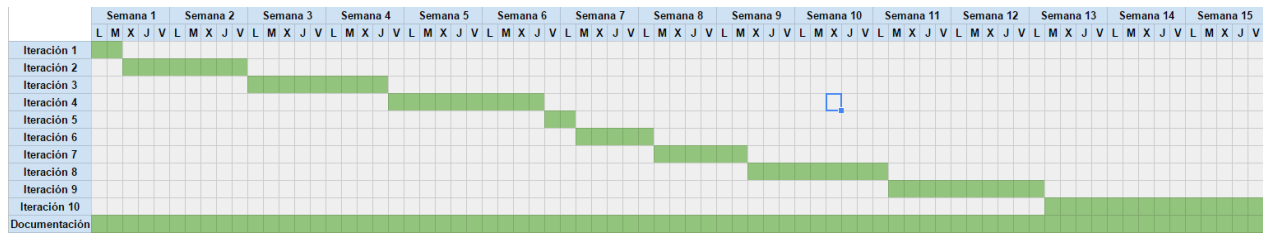


Figura 3.13: Planificación temporal.

3.2. Planificación

Antes de empezar con el desarrollo del proyecto, dividiremos las tareas en iteraciones y les asignaremos tiempos. Con esto conseguiremos que el proyecto esté terminado en el tiempo estipulado.

Según la normativa del trabajo fin de máster, el proyecto debe durar alrededor de las 300 horas. Si establecemos una jornada laboral de 4 horas diarias, éste debe terminar a los 75 días.

Tras desarrollar los diagramas de caso de uso, tenemos una primera idea de como serán nuestras iteraciones. A continuación las definimos y le asignamos la duración en días:

- **Iteración 1 - Estudio de los datos:** Buscaremos un software que consiga leer el fichero proporcionado por las empresas colaboradoras, observaremos el formato, que atributos posee cada dato, y que tipo de datos necesitaríamos almacenar. *2 días*
- **Iteración 2 - Diseño de la BBDD:** Entorno a los datos estudiados en la primera iteración, buscaremos información de otros modelos de almacenamiento y definiremos nuestro esquema. En caso de no encontrar ningún esquema que sirva para almacenar nuestros datos, definiremos uno propio. *8 días*
- **Iteración 3 - Tratamiento de los datos:** Trataremos de exportar los datos desde los ficheros y almacenarlos en nuestra base de datos espacial. Para ello debemos buscar algún software que sea capaz de realizar esta conversión. *9 días*
- **Iteración 4 - Visualización:** Empezamos a desarrollar la aplicación, lógicamente,

el primer paso es traer los datos desde la base de datos y poder visualizarlos por pantalla. En esta iteración también desarrollaremos dos tipos de cámara, una de tipo vuelo y otra de tipo cenital. Esta tarea abarca "Visualización de datos" y "Lectura de datos" dentro de los diagramas de caso de uso. *10 días*

- **Iteración 5 - Mostrar información:** Los objetos virtuales representan objetos de la realidad, pero a simple vista hay propiedades que no conocemos. Por lo que debemos mostrar toda la información asociada a un elemento subterráneo en forma de log al pulsar encima de él. Podemos englobar este trabajo dentro de la "Visualización de datos". *2 días*
- **Iteración 6 - Actualización de datos:** Cada objeto que representamos tiene propiedades geométricas y propiedades , debemos ser capaces de modificar ambas. La geometría se modificará por medio del arrastre de alguno de sus puntos, mientras que el resto se hará por formulario. *5 días*
- **Iteración 7 - Borrado de datos:** Eliminar un objeto no implica no mostrarlo por pantalla, sino que existen restricciones que deberemos seguir cumpliendo. Para ello se realizarán operaciones sobre la base de datos. *6 días*
- **Iteración 8 - Inserción de datos:** Como dijimos en anteriores capítulos, nuestro SIG 3D debe dar la posibilidad de crear objetos. La insercción de nuevos objetos implica crear una topología con otros elementos que estén conectados al nuevo insertado. *9 días*
- **Iteración 9 - UI:** Una vez que el sistema esté desplegado, no veremos los logs. Crearemos una interfaz de usuario para que el usuario pueda visualizar y realizar operaciones de forma más natural. *10 días*
- **Iteración 10 - Visualización RA:** Además de ofrecer un SIG 3D, también daremos al usuario la posibilidad de ver los elementos subterráneos a pie de calle. Para ello usaremos la realidad aumentada. Hay que tener en cuenta que las operaciones que anteriormente diseñamos para nuestro SIG 3D deben estar disponibles en este

visor. En esta iteración nos enfrentamos a la problemática del posicionamiento y orientación del usuario, por lo que será una iteración más costosa que el resto. 14 días

Cada iteración se alternará con la documentación asociada. Un resumen gráfico de la planificación lo encontramos en la Figura 3.13.

3.3. Presupuesto

Una vez que se ha definido el tiempo de duración del proyecto, queda saber el coste que acarrea embarcarse en este desarrollo. Para ello asignaremos recursos a las tareas que hemos previsto anteriormente.

Empezamos con los recursos humanos. Aunque el proyecto ha sido realizado por una sola persona, se han asumido dos roles. Asignamos un sueldo a cada tipo de personal implicado. Establecemos los días que hemos necesitado trabajar en un rol y lo multiplicamos por el número de horas de nuestra jornada laboral. En el Cuadro 3.1 vemos el desglose de cada persona.

Recurso	Cantidad	Sueldo €/h	Horas	Coste para el proyecto
Analista	1	9.80€/h	19 días · 4 horas	744.80€
Programador	1	6.89€/h	56 días · 4 horas	1543.36€

Cuadro 3.1: Presupuesto de recursos humanos

En el Cuadro 3.3 mostramos el presupuesto para hardware. Necesitaremos un ordenador donde desarrollar el programa, alquilar un servidor para almacenar nuestros datos en la nube, una Google Tango para realizar la visualización con realidad aumentada y hemos incluido un GPS submétrico para proporcionar mayor precisión al posicionamiento del usuario. El servidor que hemos alquilado cuenta con las siguientes características:

Procesador	Intel Xeon E5-1650v2 -
Cores/Threads	6c/12t
Frecuencia	3,5 GHz/3,9 GHz
RAM	64GB DDR3 ECC 1333 MHz
Discos	SoftRaid 2x3TB SATA3
Ancho de banda	250 Mbps
Tráfico	Ilimitado

Cuadro 3.2: Características del servidor contratado

Recurso	Cantidad	Importe	Coste por mes	Coste para el proyecto
Ordenador	1	1200€	25€	75€
Google Tango	1	500€	10.41€	31.26€
GPS submétrico	1	2.230,20€	46.46€	139.38€
Alquiler de servidor	1	64,99€/mes	64,99€/mes	194.97€

Cuadro 3.3: Presupuesto de hardware

También debemos tener en cuenta el coste del software que usaremos. Lo primero será adquirir una licencia de Windows 10. Unity es totalmente gratuito y la documentación se desarrolla en Latex. Para modelar los esquemas UML hemos utilizado Visual Paradigm en su versión más básica, abaratando los costes del proyecto. En el Cuadro 3.4 desglosamos el coste que supone la adquisición de licencias para el proyecto.

Recurso	Cantidad	Importe	Coste por mes	Coste total
Microsoft Windows 10	1	251€	5.22€	15.68€
Visual Paradigm Modeler version	1	6€/mes	6€	18€
FME Desktop	1	2013.79€	41.95€	125.85€

Cuadro 3.4: Presupuesto de licencias

Como últimos gastos, presentamos aquellos que no tienen cabida en otras secciones. En este caso necesitaremos una tarifa de datos para utilizarla con Google Tango.

Recurso	Cantidad	Importe	Coste total para el proyecto
Tarifa de datos 25 GB	1	25.60€	76.80€

Cuadro 3.5: Presupuesto de otros recursos

Así podremos realizar pruebas en cualquier parte sin tener que depender de una red inalámbrica Wi-Fi. El coste se muestra en el Cuadro 3.5.

Para finalizar, en el Cuadro 3.6 encontramos el coste total del proyecto.

Recurso	Cantidad	Coste total
Analista	1	744.80€
Programador	1	1543.36€
Ordenador	1	75€
Google Tango	1	31.26€
Trimble R1	1	139.38€
Alquiler de servidor	1	194.97€
Microsoft Windows 10	1	15.68€
Visual Paradigm Modeler version	1	18€
FME Desktop	1	125.85€
Tarifa de datos	1	76.80€
Coste total del proyecto		2965,10€

Cuadro 3.6: Coste total del proyecto

Este proyecto se venderá a las empresa interesadas con un coste de venta y despliegue de 8000€. Se les dará la posibilidad de almacenaje de datos y de actualizaciones software pagando una cuota mensual de 800€. Estando el desembolso inicial de proyecto cubierto tras el primer despliegue. En el caso de que algún comprador se atenera a la suscripción, la longitud del proyecto se alargaría, creando otros costes.

Capítulo 4

Iteraciones

4.1. Iteración 1

Estudio de los datos

Para esta iteración hemos escogido el software "Safe Software FME" para visualizar el fichero que nos ha proporcionado la empresa encargada del tratamiento del agua en la ciudad de Jaén. Hemos elegido este software por su polivalencia. Además de la visualización de todo tipo de ficheros, es capaz de realizar operaciones sobre los datos y exportarlos a otros formatos, hablaremos de esto en la tercera iteración. De momento solo usaremos la herramienta de visualización, llamada "FME Data Inspector".

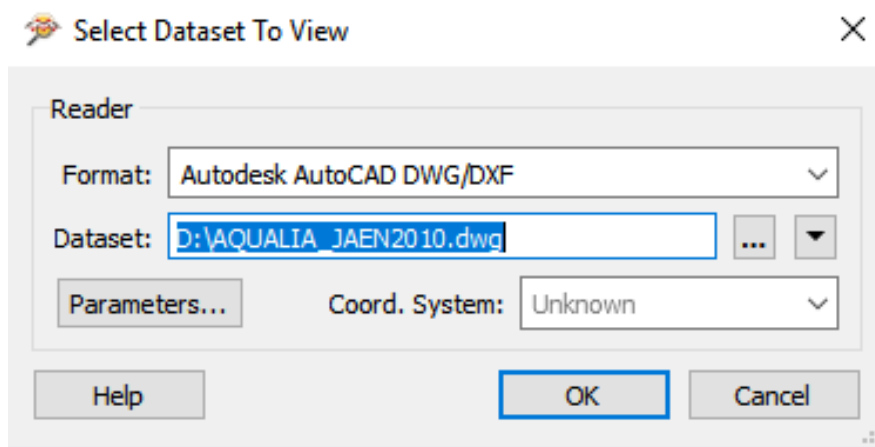


Figura 4.1: Carga de los datos con FME Data Inspector.

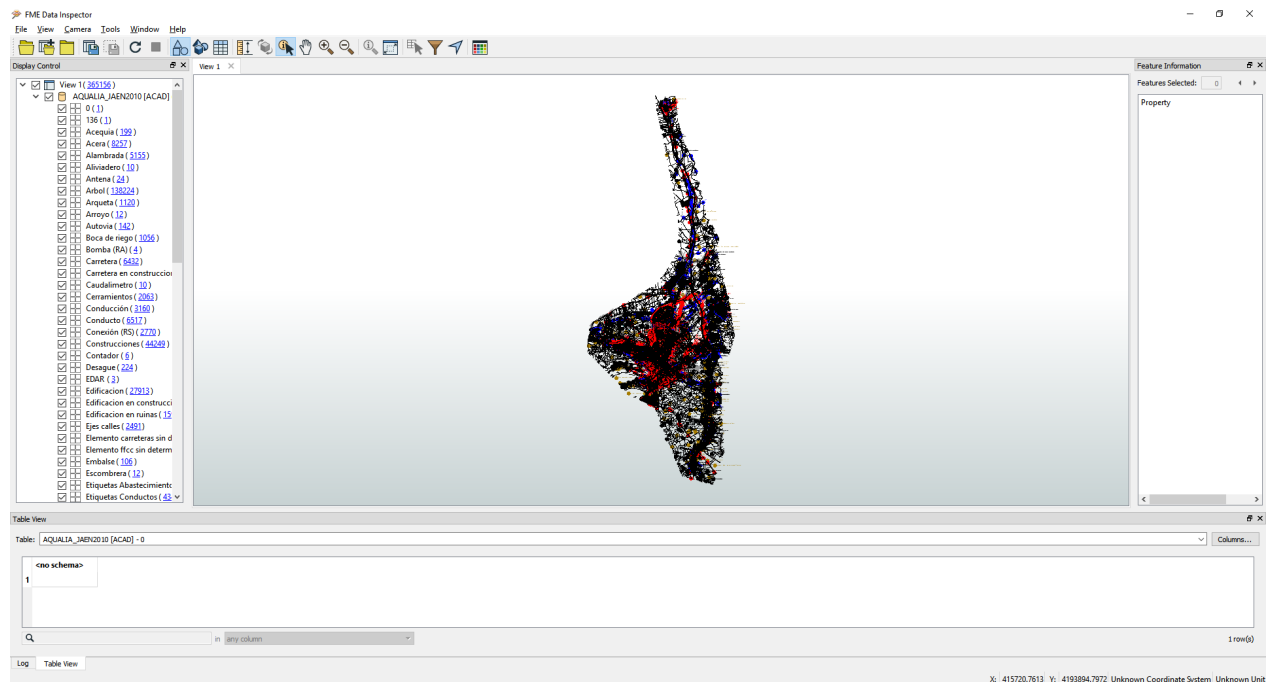


Figura 4.2: Datos cargados en FME Data Inspector.

Al iniciar nos aparece el programa sin información, esperando que abramos un fichero con nuestros datos. Figura 4.1

En la Figura 4.2 se muestra el contenido del fichero de datos. Analizaremos cada parte de la interfaz identificando la información que nos pueda ser útil. Empezamos observando la barra de información situada abajo a la derecha. Como vemos y anteriormente dijimos, los datos no están georeferenciados, por lo que posteriormente deberemos asignarle un sistema de coordenadas. La parte central está dedicada al lienzo de dibujo de los datos, en él observamos un conjunto de líneas y símbolos. Pasando a la parte izquierda, llamada "Display Control", nos encontramos con los datos guardados por la empresa. En esta parte vemos los objetos etiquetados en grupos, por ejemplo conducto, Acera, conducción, etc... Por último, en la parte derecha encontramos el panel llamado "Feature Information". Si pulsamos sobre algún objeto, aquí aparecerá la información relacionada con el elemento escogido.

Cada vez que deslizamos o hacemos un zoom sobre la vista de los datos deben repintarse todos los objetos sobre nuestro lienzo. El dataset proporcionado incluye 365156 elementos, por lo que cada vez que realicemos una acción sobre el mapa, se vuelven a

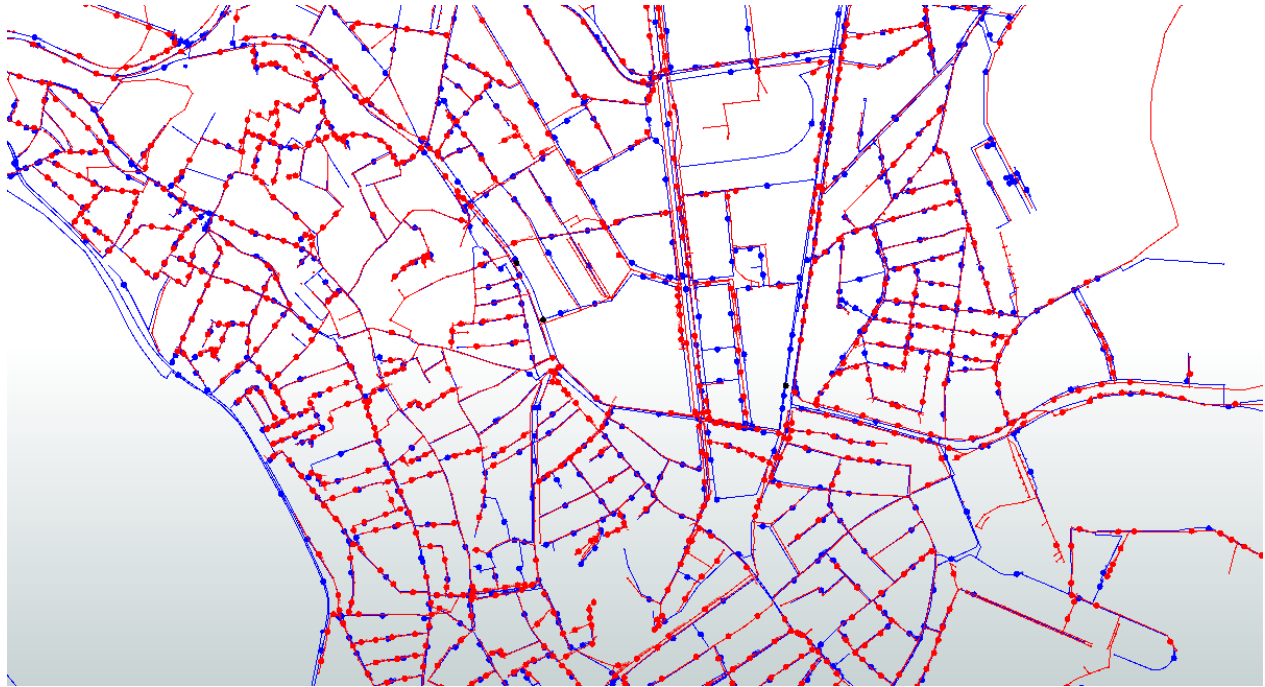


Figura 4.3: Red de saneamiento y de abastecimiento.

dibujar todos. Como vemos en la parte izquierda, hay muchos tipos de objetos que no son necesarios como son las carreteras, los árboles, las construcciones... Por lo tanto debemos ocultar las capas que no nos interesen.

En la Figura 4.3 observamos los datos correspondientes a la red de saneamiento (color rojo) y de abastecimiento (color azul) que incluye el fichero. En este momento solo se diferencia dos geometrías, puntos y líneas. Pero si hacemos zoom sobre alguno de esos puntos, encontramos una simbología asociada a cada tipo de objeto. En la Figura 4.4 apreciamos líneas, que son las tuberías, círculos y dos triángulos unidos por un vértice. Para descubrir que significa cada simbolo, pulsaremos sobre él y leeremos su tipo en el panel de información. Por ejemplo, los dos círculos concéntricos es la representación de un "pozo". En la misma imagen observamos unos puntos con letras y números, esta es la codificación usada para saber el diámetro de una tubería y su material. Por ejemplo, si vemos la etiqueta 100FUND, significa que la tubería tiene un diámetro exterior de 100 centímetros y que está hecha de fundición dúctil.

En la Figura 4.5 vemos la simbología usada para cada tipo de objeto. Las tuberías de cada red son representadas en forma de polilínea.

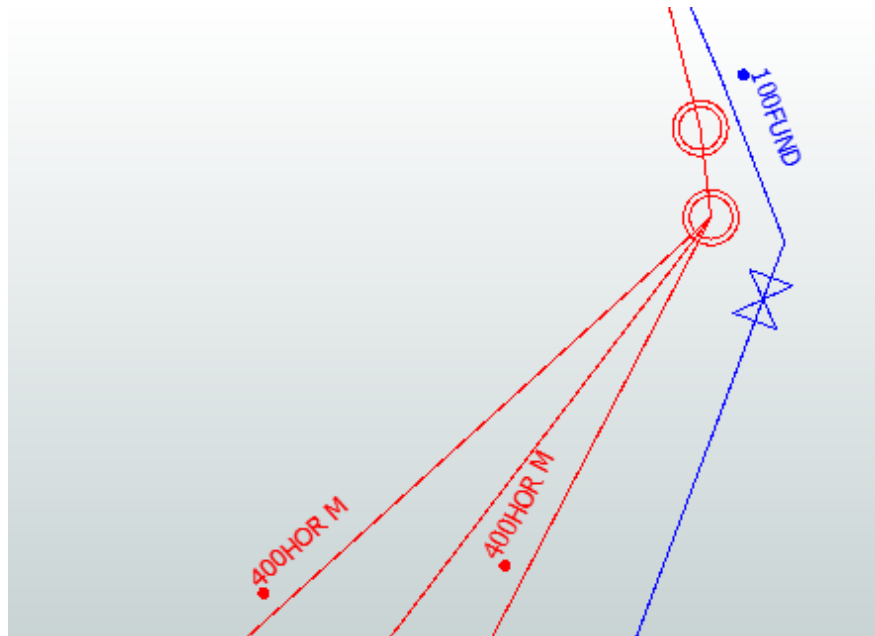


Figura 4.4: Simbología de los objetos de las redes.

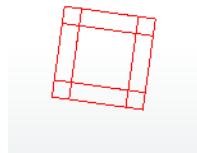
Una vez que ya hemos descubierto cómo se simbolizan los elementos subterráneos, buscaremos información de cómo tratar los datos que contienen. Empezaremos observando la información de los pozos. Los pozos recordamos que están representados como dos círculos concéntricos. Según si pulsamos sobre el círculo interior o exterior, se nos ofrece distinta información. El primero es tratado como un polígono en dos dimensiones, mientras que el segundo es tratado como una línea continua en tres dimensiones. Esta información es heterogénea y no se sabe si la altura es correcta o no.

De igual manera sucede con el resto de objetos, según donde pulsemos dentro de un símbolo se nos proporciona una información distinta a otra parte del mismo. Estos problemas se intentarán solucionar en la tercera iteración.

4.2. Iteración 2

Diseño de la base de datos

En la subsección 3.1.5 ya definimos el diseño que vamos a usar y se justificó correctamente. Por lo tanto, en esta Sección explicamos como hemos llegado a la conclusión



(a) Arqueta



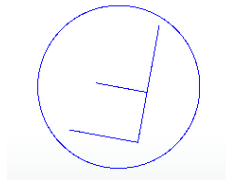
(b) Pozo



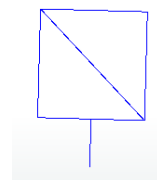
(c) Imbornal



(d) Boca de riego



(e) Filtro



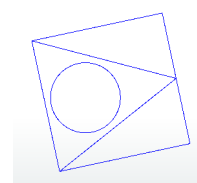
(f) Hidrante



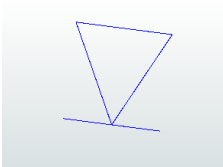
(g) Tapón



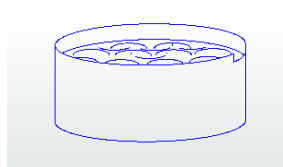
(h) Válvula de aislamiento



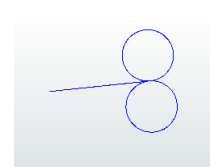
(i) Válvula de regulación



(j) Válvula de retención



(k) Vaso



(l) Ventosa

Figura 4.5: Simbología en formato CAD usada por la empresa encargada de la gestión del agua en Jaén

Property	Value	Property	Value
Feature Type	Pozo	Feature Type	Pozo
Coordinate System	Unknown	Coordinate System	Unknown
Dimension	3D	Dimension	2D
Number of Vertices	64	Number of Vertices	64
Min Extents	430893.832, 4182677.34, 461.981	Min Extents	430894.01, 4182677.518
Max Extents	430895.405, 4182678.914, 461.981	Max Extents	430895.227, 4182678.736
> Attributes (39)		> Attributes (48)	
▼ IFMELINE (64 Coordinates)	(430895.405, 4182678.127, 461.981)	▼ IFMEPolygon	
Closed	Closed In 3D	Linear Boundary	Yes
▼ Coordinates (64)	Coordinate Dimension: 3	Convex	Yes
0	430895.405, 4182678.127, 461.981	Orientation	Left Hand Rule
1	430895.402, 4182678.192, 461.981	▼ Boundary: IFMEPath (1 Segment)	
2	430895.392, 4182678.27, 461.981	Closed	Closed In 2D
3	430895.374, 4182678.347, 461.981	▼ Segment 0: IFMELINE (64 Coordinates)	(430895.227, 4182678.127), ..., (430895.227, 4182678.127)
4	430895.348, 4182678.421, 461.981	Closed	Closed In 2D
5	430895.315, 4182678.493, 461.981	▼ Coordinates (64)	Coordinate Dimension: 2
6	430895.275, 4182678.56, 461.981	0	430895.227, 4182678.127
7	430895.228, 4182678.624, 461.981	1	430895.225, 4182678.178
8	430895.176, 4182678.682, 461.981	2	430895.217, 4182678.238
9	430895.118, 4182678.735, 461.981	3	430895.203, 4182678.297
10 - 53	Show All or 20 more	4	430895.183, 4182678.355
54	430895.107, 4182677.511, 461.981	5	430895.157, 4182678.41
55	430895.166, 4182677.562, 461.981	6	430895.126, 4182678.462
56	430895.22, 4182677.62, 461.981	7	430895.09, 4182678.511
57	430895.268, 4182677.683, 461.981	8	430895.05, 4182678.557
58	430895.309, 4182677.75, 461.981	9	430895.005, 4182678.598
59	430895.343, 4182677.821, 461.981	10 - 53	Show All or 20 more
60	430895.37, 4182677.894, 461.981	54	430894.997, 4182677.65
61	430895.389, 4182677.971, 461.981	55	430895.042, 4182677.69
62	430895.401, 4182678.048, 461.981	56	430895.084, 4182677.735
63	430895.405, 4182678.127, 461.981	57	430895.121, 4182677.783
		58	430895.152, 4182677.835
		59	430895.179, 4182677.89
		60	430895.2, 4182677.947
		61	430895.215, 4182678.006
		62	430895.224, 4182678.066
		63	430895.227, 4182678.127

(a) Propiedades de un pozo según su círculo exterior

(b) Propiedades de un pozo según su círculo interior

Figura 4.6: Información sobre los pozos, a la izquierda el círculo exterior y a la derecha el círculo interior

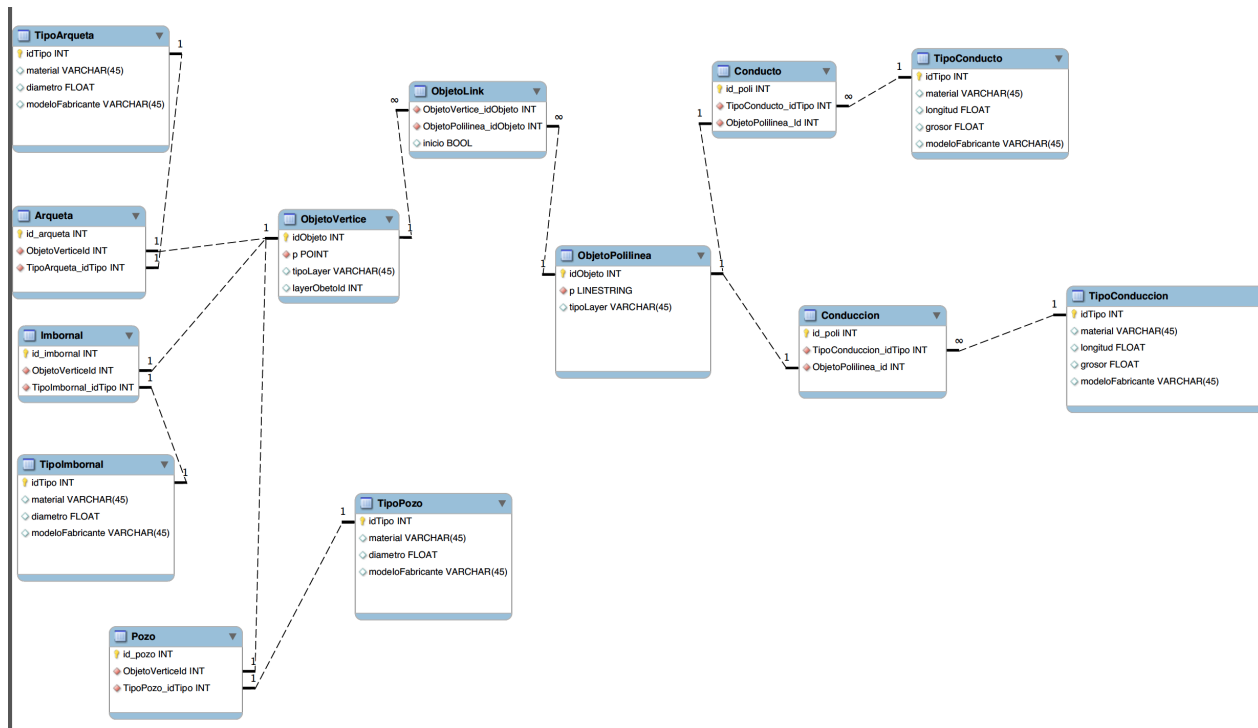


Figura 4.7: Primer diseño de nuestra base de datos.

de que ese diseño es el correcto para nuestro proyecto.

Empezamos creando un diseño propio tal y como se muestra en la Figura 4.7. En él observamos dos tablas donde almacenamos la geometría, "ObjetoVertice" y "ObjetoPolilinea" correspondientes a pozos y tuberías por ejemplo. Cada tabla contiene el campo "tipoLayer" que indica qué tipo de objeto es. Si queremos saber más información sobre un determinado objeto deberemos acudir a la tabla que nos indica el valor del campo anteriormente mencionado. Tras esto, para parametrizar el objeto entorno a sus dimensiones y material, existe una tercera tabla para cada tipo de objeto, como "TipoImbormal" o "TipoConducto". Contamos también con una tabla que nos conecta topológicamente los objetos según su geometría. Este diseño es de difícil mantenimiento, acceso y poco escalable pero nos da una idea de lo que debemos guardar en nuestro verdadero diseño de la base de datos.

Como dijimos en la Sección 1.1 la OGC propuso el uso del estándar GML y posteriormente CityGML, para compartir datos que posean geometría. CityGML además de geometría se encarga de la georeferenciación de esos datos dentro de ciudades. A éste

estandar se le pueden incorporar extensiones llamadas ADE (Application Domain Extensions). Cada ADE trata sobre el modelado de ciertas partes de la ciudad, por ejemplo túneles o puentes. Si investigamos los ADE's disponibles, encontramos "UtilityNetwork". Éste propone el uso de ciertas clases para compartir información sobre las infraestructuras del subsuelo.

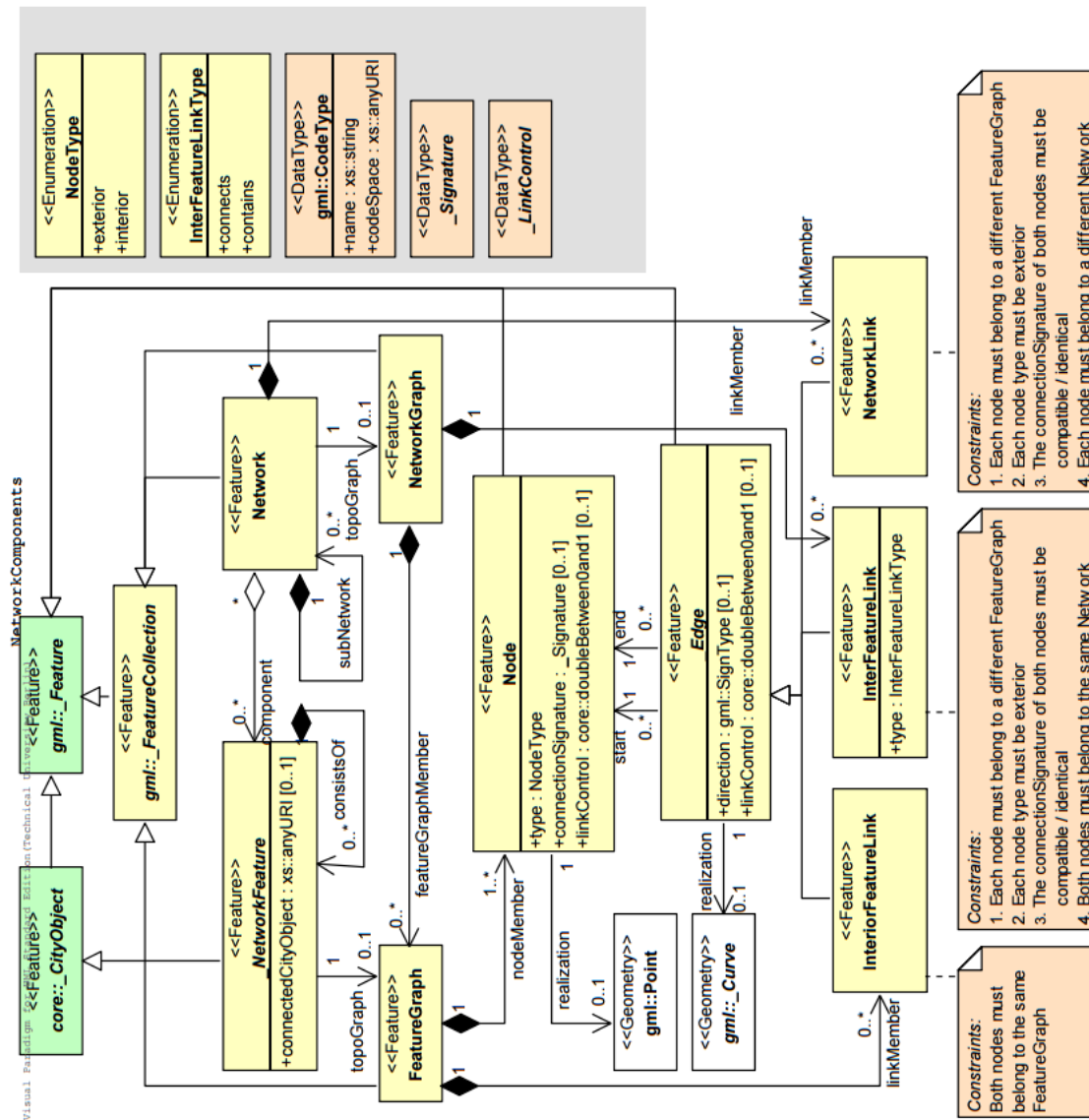


Figura 4.8: Diseño del núcleo propuesto en "Utility Network ADE" siguiendo el estándar CityGML.

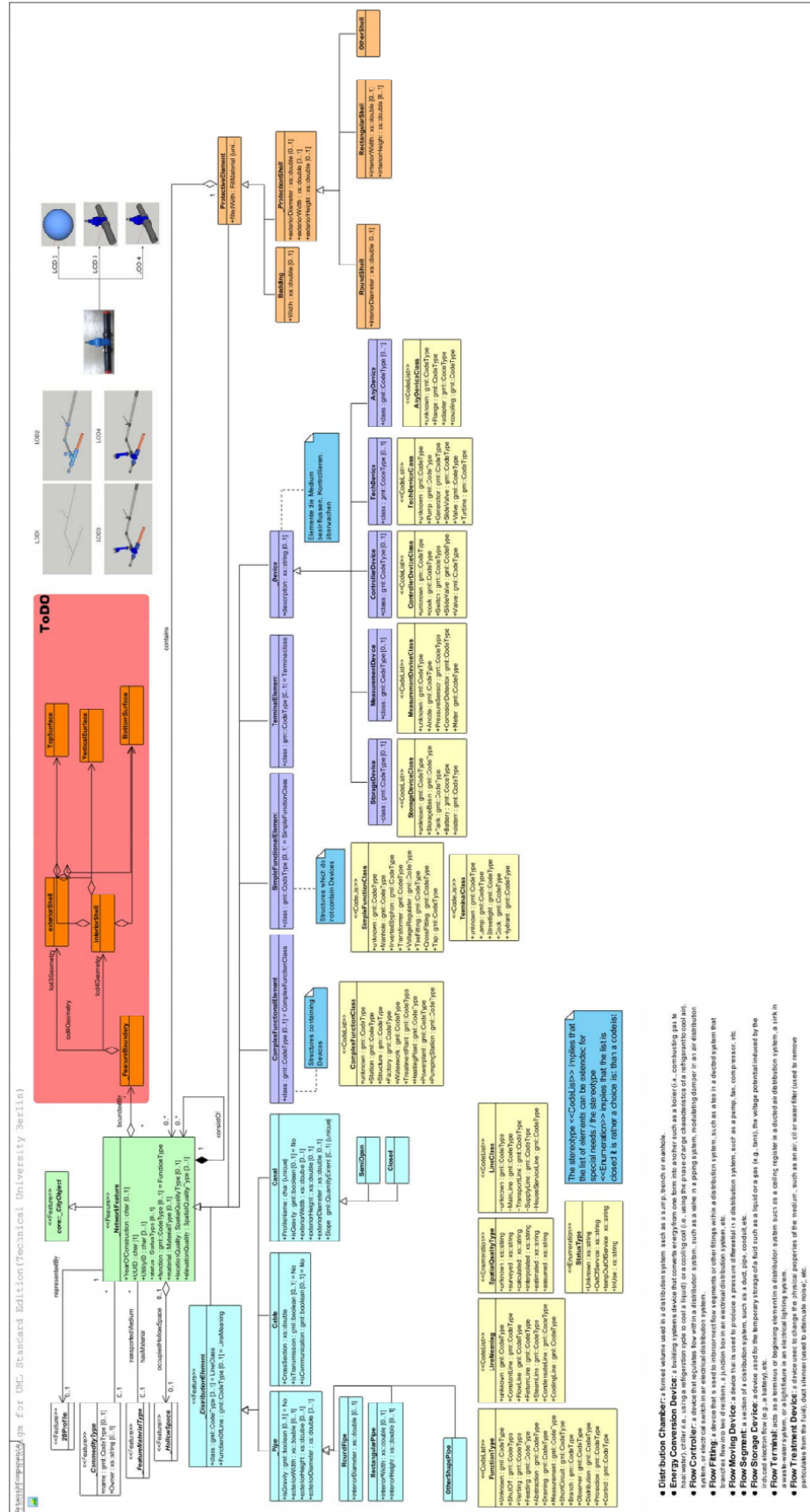


Figura 4.9: Diseño de los objetos propuesto en "Utility Network ADE" siguiendo el estándar CityGML.

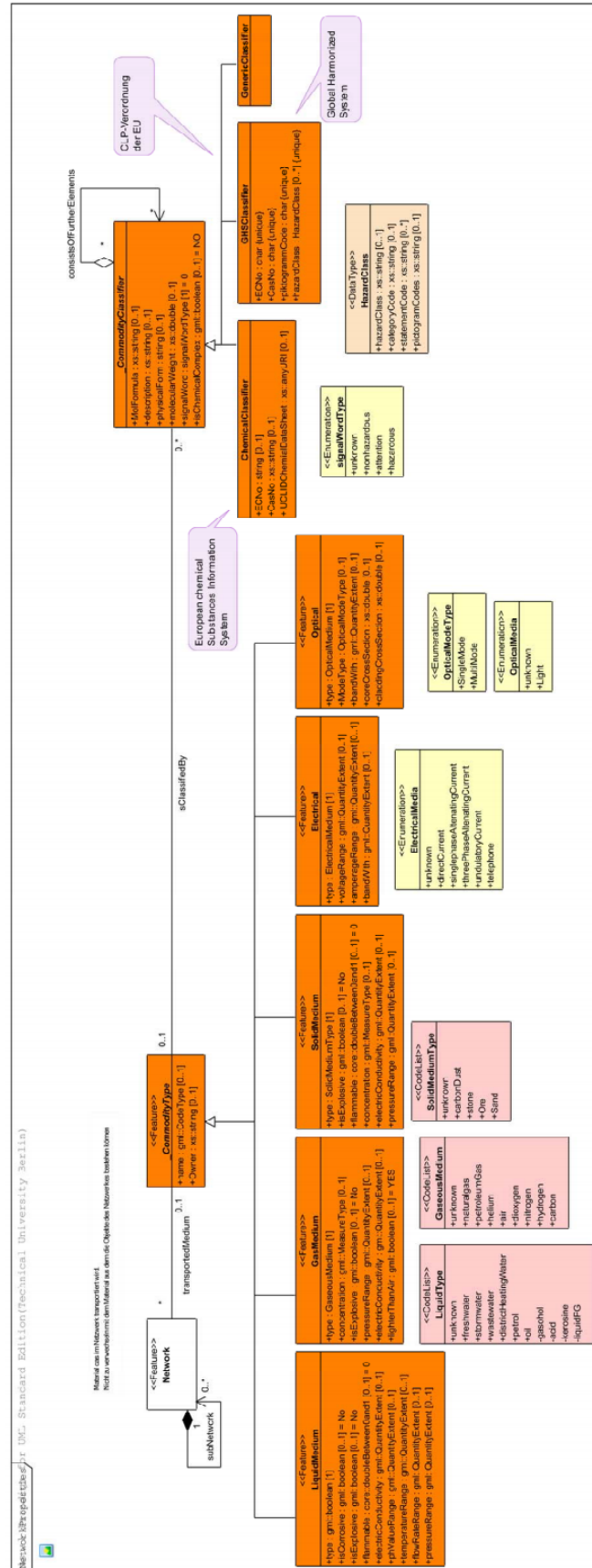


Figura 4.10: Diseño de las redes propuesto en "Utility Network ADE" siguiendo el estándar CityGML.

Actualmente este proyecto está abandonado, pero estudiaremos su estructura en busca de información que podamos extrapolar a nuestro diseño de la base de datos. La Figura 4.8 define el núcleo de la extensión. En él vemos como se guarda la geometría de cada objeto y su conexión con otros elementos. También podemos ver como se almacenan objetos en las redes. Su propuesta para la topología no nos ha parecido la más adecuada en este caso. En la Figura 4.9 se establece la clase "NetworkFeature". Esta clase contiene información general que poseen todos los objetos, sin importar su naturaleza. Por ejemplo encontramos el estado del objeto, que función desempeña o la calidad de los datos. De esta clase heredan objetos más concretos. Vemos la clase "DistributionElement", encargada de almacenar los objetos que cumplan una función de transporte. Un objeto más concreto incluso sería una "Pipe", donde se introducen tuberías parametrizadas por sus dimensiones. Pero si el objeto no está destinado para el transporte de materiales, y si para cumplir ciertas funcionalidad como almacenaje, medición o de control, también existen clases que se basan en la clase "NetworkFeature". Por último cabe destacar que se nos proporcionan enumeraciones que podemos usar en nuestro sistema como por ejemplo los distintos estados en los que se puede encontrar un objeto subterráneo. En la Figura 4.10 se muestran cómo pueden ser diseñadas las redes de objetos.

Este proyecto propone un modelo para la exportación de elementos de las redes subterráneas y que si conseguimos extrapolar esto a nuestro diseño de la base de datos puede sernos útil. Si observamos detenidamente tanto este proyecto como el diseño de base de datos que hemos propuesto en la Figura 3.12, encontramos ciertas similitudes:

- Las propiedades generales de cualquier objeto se almacenan de la misma manera en ambos casos.
- Los objetos pueden ser de dos tipos, de distribución o funcionales.
- Ambos contienen una referencia a la red a la que pertenecen.

Pero también encontramos diferencias, bien porque existe información que a nuestro proyecto no aporta nada o bien porque no necesitamos tanto detalle:

- Los objetos funcionales se han unido en una misma tabla usando un atributo para diferenciarlos.

- No hemos detallado la información sobre una red, limitandonos a guardar sus propiedades básicas.
- Usamos solo un material para representar el objeto.
- Nuestras tuberías serán parametrizadas solo por su diámetro exterior.
- De momento no contemplamos la posibilidad de conexión entre dos redes distintas.
- La topología es almacenada de forma diferente.

Aunque como dijimos anteriormente, el proyecto está abandonado, nos parece un buen diseño para almacenar nuestros datos por lo que nos hemos basado en él para crear nuestro diseño de base de datos. De nuevo remitimos al lector a la Subsección 3.1.5 para mostrar el resultado de nuestra base de datos.

Esta iteración queda finalizada y en la siguiente se abordará la problemática de la extracción de los datos de nuestro fichero CAD y hemos rellenado nuestra base de datos con ellos.

4.3. Iteración 3

Tratamiento de los datos

Como se comentó en la Sección 4.1, usaremos el software FME para llevar a cabo el proceso de exportación de los datos en formato CAD a otro formato que nos permita introducirlos en la base de datos. Utilizaremos la herramienta FME Workbench. La forma de trabajar es a través del arrastre de nodos representando cada uno de ellos una acción sobre los datos. Cada nodo está compuesto por entradas y por salidas, deberemos conectar la salida de un nodo con la entrada del siguiente.

El procedimiento llevado a cabo con FME se puede resumir en el siguiente proceso:

- Se elige un fichero desde el que cargaremos los datos y seleccionaremos que tipo de objeto deseamos leer. Cabe la posibilidad de establecer un sistema de coordena-

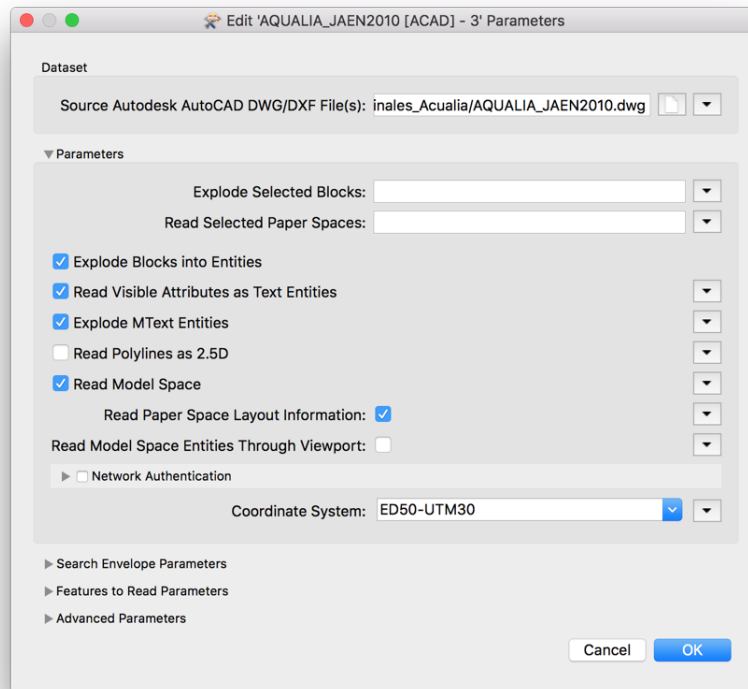


Figura 4.11: Reader de FME para lectura de entidades.

das de origen si es conocido. Por ejemplo, del fichero AQUALIAJAEN.dwg trabajaremos con los datos que estén etiquetados como Pozo en el sistema de coordenadas ED50. Figura 4.11.

- Crearemos nodos de operación tras la entrada. Con esto conseguiremos modificar los datos, ajustándolos a nuestras necesidades. Un ejemplo es realizar la caja envolvente de la representación de un pozo para después extraer el centroide.
- Exportaremos los datos a otro formato aplicándole un sistema de georeferenciación. Por ejemplo, a los puntos que hemos procesado los transformaremos a coordenadas ETRS89 y los exportaremos a un fichero tipo GML.

Tenemos que tener en cuenta que los objetos como pozos, arquetas o válvulas tienen una representación y una naturaleza distintas a las tuberías, por lo que será necesario identificar cuales son esas diferencias y tratar de forma diferente los datos. Es por eso que a continuación haremos una diferenciación entre el proceso de extracción de los datos.

Entidades puntuales

Trataremos como entidades puntuales aquellas que están posicionadas por el centro del objeto en cuestión. Es decir, no se tiene en cuenta su volumen para posicionarlos en el espacio, si no que será su centro el que determinará su situación en el mapa. Su representación es la de un polígono, pero nosotros solo necesitaremos la posición del objeto, por lo que buscaremos la forma de extraer solo una coordenada.

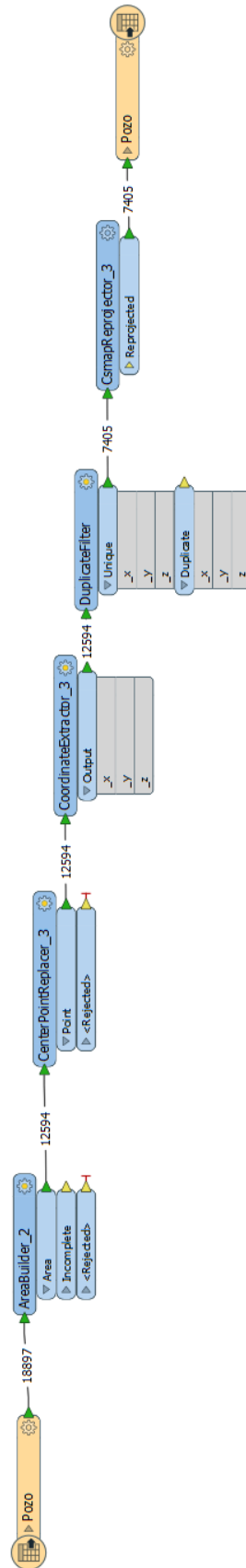


Figura 4.12: Flujo de FME para entidades puntuales.

Al estar estos objetos representados de diferentes maneras dentro del fichero CAD, será necesario crear un flujo de transformación de entidades el cual sea capaz de extraer las coordenadas de cada objeto independientemente de su representación. Este flujo se puede observar en la Figura 4.12. En la siguiente lista explicamos la funcionalidad de cada uno de esos nodos:

- **DWG Reader:** Este nodo lee la información desde un fichero con formato DWG y extrae un tipo de objeto. En este caso se han tomado los pozos que existen el fichero.
- **AreaBuilder:** Se construye la caja envolvente del cada representación.
- **CenterPointReplacer:** Transforma un conjunto de vértices en un solo punto del espacio a elegir entre el centroide, el centro de gravedad y un punto interior cualquiera. En este caso, y como parámetro por defecto, se ha escogido el centroide del polígono creado anteriormente. Por lo tanto, el nodo extrae el punto central de la caja envolvente que le llega como entrada.
- **CoordinateExtractor:** Este nodo formatea las coordenadas. En este caso, el parámetro por defecto es una colección de vértices por lo que se ha configurado de forma que solo reciba y expida una sola coordenada.
- **DuplicateFilter:** Eliminamos posibles puntos duplicados que puedan existir.
- **CsmapReprojector:** A partir de un sistema de coordenadas de referencia transforma los datos a otro sistema de coordenadas. En este caso se ha pasado de ED50, antiguo sistema de coordenadas en el que se situaba España, a ETRS89.
- **GML Writer:** Escribe los vértices que le llegan a este nodo en un fichero con formato GML.

Las pantallas de configuración de los elementos que no se han utilizado los parámetros por defecto podemos verlas en la Figura 4.13.

Entidades lineales

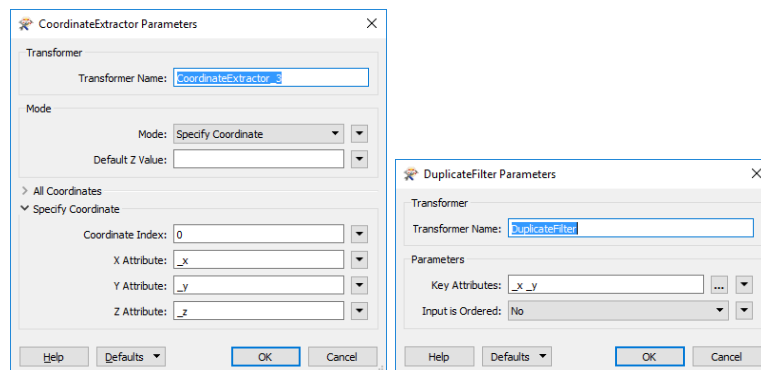


Figura 4.13: Procesamiento de entidades puntuales.

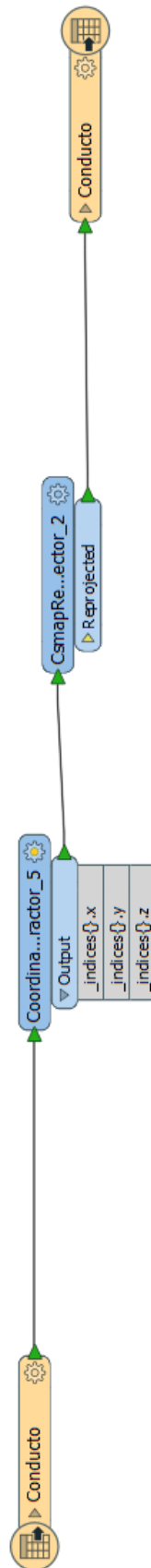


Figura 4.14: Flujo de FME para entidades de tipo polilínea.

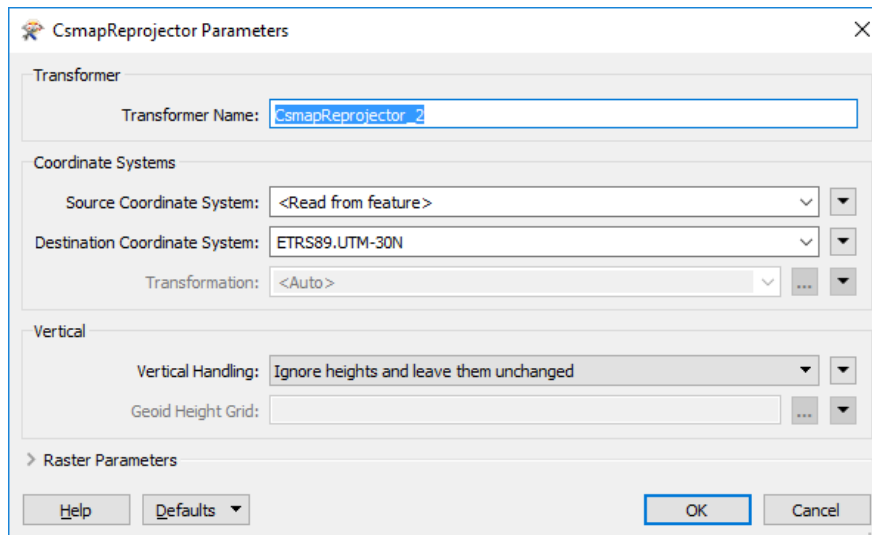


Figura 4.15: Procesamiento de entidades de tipo polilínea.

Estas entidades tienen la peculiaridad de que empiezan en un punto, terminan en otro diferente del espacio y contiene puntos intermedios, por lo tanto serán tratados como polilíneas ya que en los datos de entrada están codificados de esta forma. En la Figura 4.14 podemos ver el flujo que se ha seguido para la extracción de las mismas y los parámetros utilizados en los nodos que no se han utilizado los parámetros por defecto, podemos verlos en la Figura 4.15.

Explicaremos cada uno de los nodos:

- **DWG Reader:** Este nodo lee la información desde un fichero con formato DWG y extrae un tipo de objeto. En este caso se han tomado los conductos que existen el fichero.
- **CoordinateExtractor:** Este nodo extrae las coordenadas de otro nodo y las pasa al siguiente nodo. Al contrario que en el caso de los objetos puntuales, este tipo de objeto sí es una colección de vértices, los cuales representan una polilínea, por lo tanto los trataremos de forma que estos objetos formen un conjunto de vértices.
- **CsmapReprojector:** A partir de un sistema de coordenadas de referencia transforma los datos a otro sistema de coordenadas. En este caso se ha pasado de ED50, antiguo sistema de coordenadas en el que se situaba España, a ETRS89.

- **GML Writer:** Escribe los vértices que le llegan a este nodo en un fichero con formato GML.

Insercción de los datos

En este momento, tenemos un fichero por cada tipo de objeto que hemos tratado. Su formato actual es el de un fichero GML, es decir, la información está situada entre etiquetas tipo XML. A continuación vemos un ejemplo del fichero "Arquetas.gml" y "Conductos.gml"

```

<core:cityObjectMember>                <gen:GenericCityObject>
<gen:lod4Geometry>                      <gml:Point
srsName=.EPSG:3042"srsDimension="3»
<gml:pos>430813.736371703                4182900.67091206
429.225</gml:pos>                        </gml:Point>
</gen:lod4Geometry>    </gen:GenericCityObject>    </co-
re:cityObjectMember>                <core:cityObjectMember>
<gen:GenericCityObject>                <gen:lod4Geometry>
<gml:Point                srsName=.EPSG:3042"srsDimension="3»
<gml:pos>430701.500572734                4182889.97664921
429.636</gml:pos>                        </gml:Point>
</gen:lod4Geometry>    </gen:GenericCityObject>    </co-
re:cityObjectMember>                <core:cityObjectMember>
<gen:GenericCityObject>                <gen:lod4Geometry>
<gml:Point                srsName=.EPSG:3042"srsDimension="3»
<gml:pos>430818.624361287                4182901.00587855
429.168</gml:pos>    </gml:Point>    </gen:lod4Geometry>
</gen:GenericCityObject> </core:cityObjectMember>

```

```

<core:cityObjectMember>                <gen:GenericCityObject>
<gen:lod4Geometry>                      <gml:LineString
srsName=.EPSG:3042"srsDimension="3»
<gml:posList>427863.563291071          4181962.39761732
555.731          427863.646303498      4181963.39961786
555.728          427863.841312963      4181964.19361742
555.721</gml:posList>                  </gml:LineString>
</gen:lod4Geometry>    </gen:GenericCityObject>    </co-
re:cityObjectMember>                <core:cityObjectMember>
<gen:GenericCityObject>                <gen:lod4Geometry>
<gml:LineString      srsName=.EPSG:3042"srsDimension="3»
<gml:posList>427884.343339492          4181971.0954877
554.825          427883.294335113      4181970.50349408
554.88          427881.355325772      4181969.31050576
554.983</gml:posList>                  </gml:LineString>
</gen:lod4Geometry>    </gen:GenericCityObject>    </co-
re:cityObjectMember>                <core:cityObjectMember>
<gen:GenericCityObject>                <gen:lod4Geometry>
<gml:LineString      srsName=.EPSG:3042"srsDimension="3»
<gml:posList>427919.685087737          4181959.47423843
553.541          427918.599101519      4181960.30924661
553.468          427917.637117601      4181961.35525419
553.361</gml:posList>                  </gml:LineString>
</gen:lod4Geometry>    </gen:GenericCityObject>    </co-
re:cityObjectMember>

```

Para incorporar estos datos a nuestro sistema, hemos creado un programa con el lenguaje C++. El proceso que sigue este programa es el siguiente:

- Lectura del fichero completo.
- Almacenaje en una lista de todos los valores situados entre las etiquetas <gml:Pos></gml:Pos> en el caso de objetos puntuales y <gml:PosList></gml:PosList> para las polilíneas.

- Inserción de cada elemento de la lista en la tabla correspondiente de la base de datos dándole una altura 0 a todos los objetos ya que revisando los ficheros exportados, hay elementos que no contienen esta coordenada o si la tienen, está mal situada. Al resto de campos de las tablas, como el material o el estado, se les han introducido valores por defecto momentaneamente.

Aprovechando que ya tenemos toda la información en la base de datos, uniremos los objetos topológicamente mediante una operación de PostGIS. Para ello debemos crear un buffer de influencia para cada objeto puntual e insertar su identificador junto con los identificadores de las polilíneas en la tabla "connected" cuyo punto inicial esté dentro del buffer generado por el punto. Haremos lo mismo para los puntos finales de las polilíneas.

En este momento queda establecer las propiedades temáticas de los objetos que ya están en la base de datos. Tras una reunión con el propietario de los datos, sacamos la información de los materiales que utilizan para cada tipo de objeto que consideramos puntual. Siempre utilizan el mismo para todos los objetos que son del mismo tipo, siendo por ejemplo todos los pozos hechos de ladrillo. En la misma reunión nos explicaron que los puntos que veíamos cerca de las tuberías con información del diámetro y del material podríamos tratarlo como las propiedades de las tuberías más cercanas a este punto.

Como consecuencia, establecimos el material de cada tipo de objeto puntual al que nos dijeron. Mientras, quedaba asignar las propiedades temáticas de las polilíneas. Para ello extrajimos los puntos junto a su información y los introducimos en una tabla de nuestra base de datos, separando materiales y diámetros junto a su posición con la ayuda de FME Workbench y del programa en C++ que realizamos. Una vez que todos los datos estaban cargados, realizamos una operación de búsqueda del punto más cercano, propia de PostGIS, a los puntos que contenían las propiedades. De esta forma se introdujeron las propiedades temáticas de los objetos dentro de la base de datos.

El siguiente paso fue asignarle una altura correcta a los datos. En la reunión comentada anteriormente, nos dijeron que nuestros objetos puntuales están situados entre medio metro y dos metros por debajo del nivel del suelo. Sobre las tuberías, éstas se sitúan en las mismas cotas. El proceso a seguir sería el siguiente:

- Descargar de la web del instituto geográfico español el terreno LiDAR correspondiente a Jaén e incorporarlo a nuestra base de datos.
- Asignar a cada objeto puntual la altura del terreno más cercano a su posición
- Establecer la altura de los extremos de las polilíneas que estén conectados topológicamente a algún objeto puntual
- Interpolar aquellos puntos extremos que sigan con una altura igual a cero
- Interpolar los puntos intermedios de cada polilínea

Por último modificamos el valor del campo "status" de todos los objetos con el valor "InUse", el campo "locationQuality" como "estimated" y el campo "elevationQuality" como "assumed" para el caso de objetos puntuales y "interpolated" para las polilíneas.

4.4. Iteración 4

Visualización

En estos momentos ya tenemos toda la información almacenada en nuestro servidor de base de datos. Lo siguiente que haremos será visualizar los objetos por medio de su representación virtual. Empezaremos a desarrollar un software utilizando Unity 3D como motor gráfico que realice consultas a la base de datos, extraiga la información y genere objetos virtuales que el usuario pueda apreciar.

Lo primero que hicimos fué crear los objetos virtuales ayudándonos de Blender. Para ello modelamos de forma aproximada objetos como pozos, imbornales o arquetas. Las tuberías quedan representadas como un cilindro. En la Figura 4.16 vemos la representación tridimensional de un imbornal, una arqueta, un pozo y una tubería.

Continuamos buscando las texturas que representan a los materiales y crear objetos de tipo "Material" de Unity para posteriormente ser aplicados a los objetos según el valor extraído de la base de datos. Vemos estos materiales en la Figura 4.17.

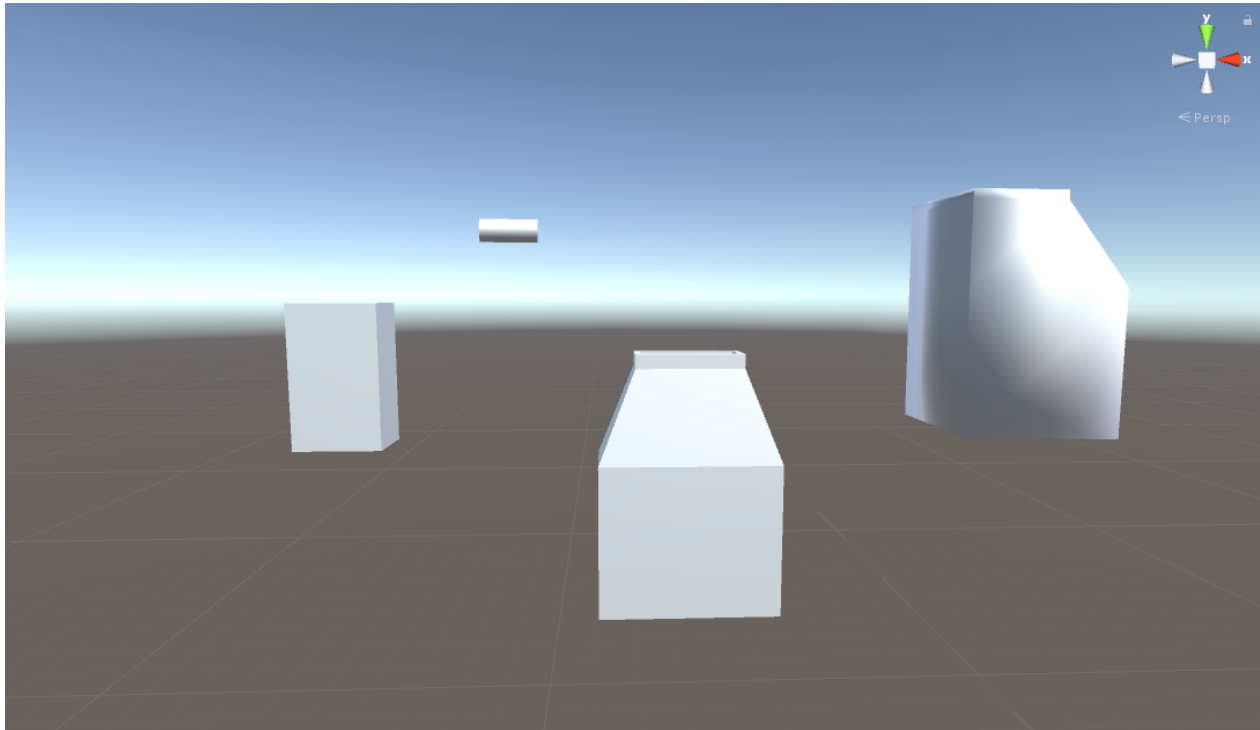


Figura 4.16: Representación tridimensional de los objetos.

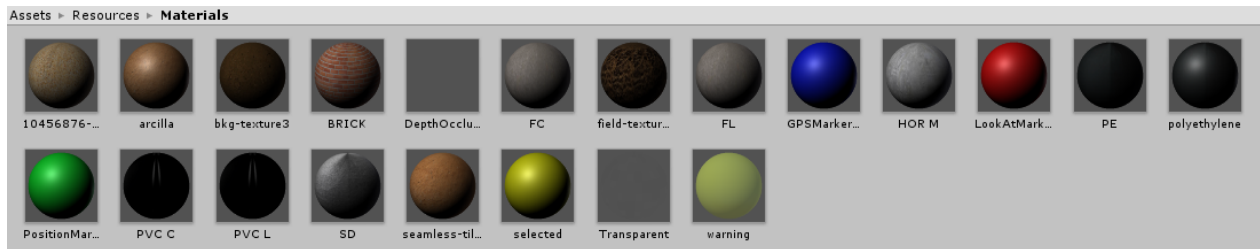


Figura 4.17: Materiales que serán aplicados a los objetos.

Una vez construida la representación de los objetos, empezamos a desarrollar según el diagrama de clases que hemos visto en la Sección (referencia a seccion).

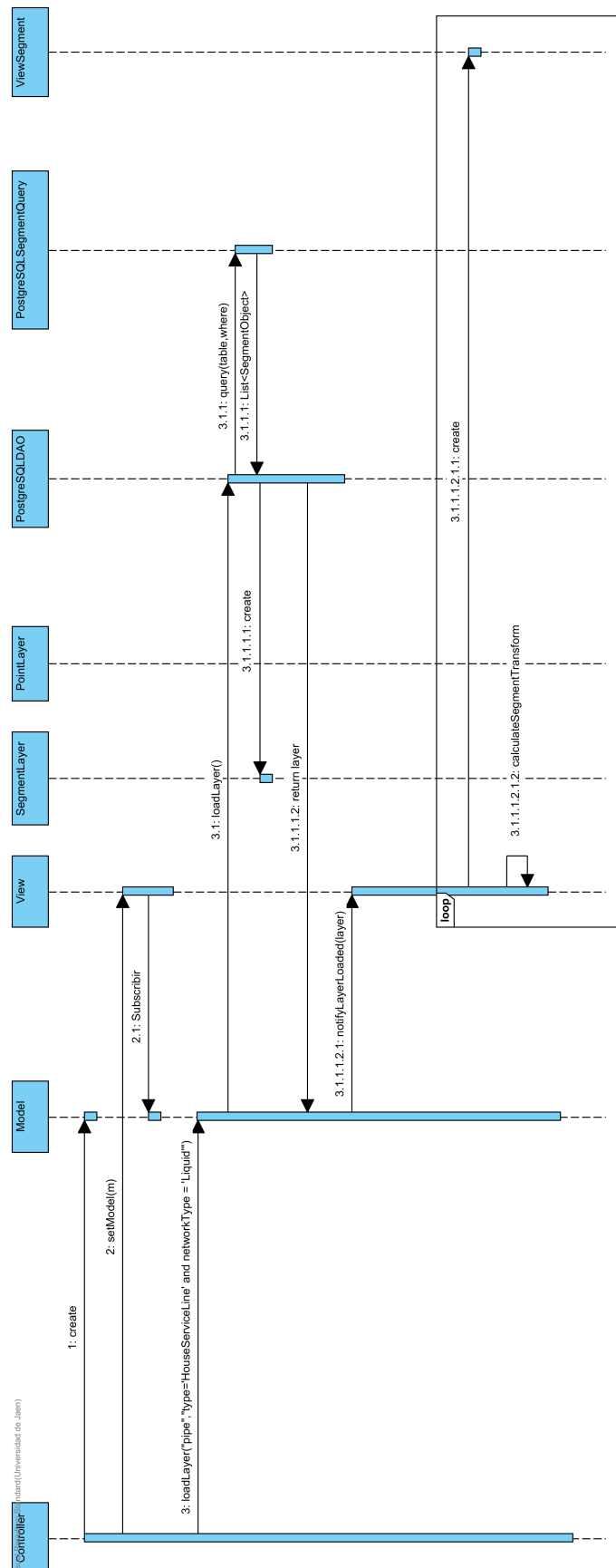


Figura 4.18: Diagrama de secuencia de la visualización de objetos.

En la Figura 4.18 observamos la comunicación entre las clases en un diagrama de secuencia para conseguir mostrar los objetos. En primer lugar, el controlador manda al modelo a cargar los datos de una tabla con una cierta condición. El modelo le pasa la consulta a la clase PostgreSQLDAO que a su vez se la pasa a PostgreSQLSegmentQuery. Una vez realiza la acción, éste último devuelve una lista de objetos de tipo segmento. PostgreSQLDAO crea una capa de segmentos y se la ofrece al modelo, que la guarda y avisa a las vistas subscritas de que hay nuevos objetos en el sistema. Es la vista quien crea la representación visual de los objetos y quien los sitúa en la escena.

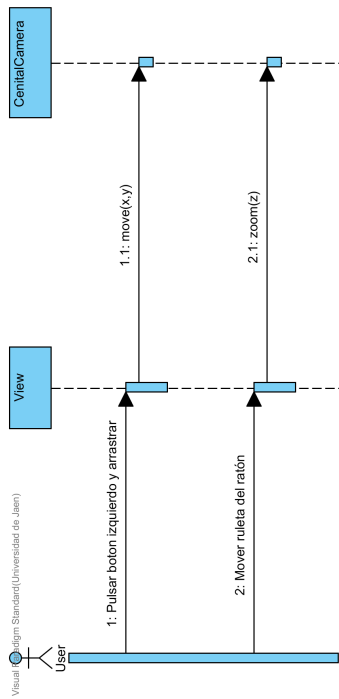


Figura 4.19: Diagrama de secuencia del movimiento de la cámara cenital.

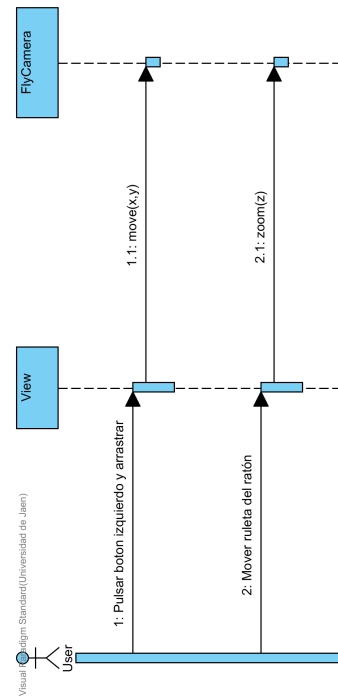


Figura 4.20: Diagrama de secuencia del movimiento de la cámara libre.

Para finalizar esta iteración debemos crear dos tipos de cámaras, una cenital y otra libre. La primera no tendrá opción de ser rotada y toda la vista será la del plano XY, mientras que la segunda se podrá mover sobre cualquier plano, siendo más fácil la visualización 3D en ésta última que en la de tipo cenital. Ambas funcionan de la misma manera como podemos comprobar en la Figura 4.19 y en la Figura 4.20. El usuario pulsa y arrastra el ratón a través de la pantalla, este evento lo recoge la vista y manda a la cámara a moverse en el plano XY. Si la acción del usuario es rotar la ruleta del ratón conseguirá moverse a través del eje Z.

Enlace a vídeo demostrativo (Cámara cenital): https://youtu.be/MF3mlwZKi2E?list=PLrwxKtY_YNVtVgt4dWXRTKskAAIm5UbRy

Enlace a vídeo demostrativo (Cámara libre): https://youtu.be/gmPMs3_B_xY?list=PLrwxKtY_YNVtVgt4dWXRTKskAAIm5UbRy

4.5. Iteración 5

Mostrar información

En esta iteración abordamos la casuística del usuario que al pulsar sobre un objeto espera recibir información sobre el mismo. En este momento no tenemos una interfaz de usuario para mostrar de forma más adecuada la información, por lo que sacaremos esta información por la consola.

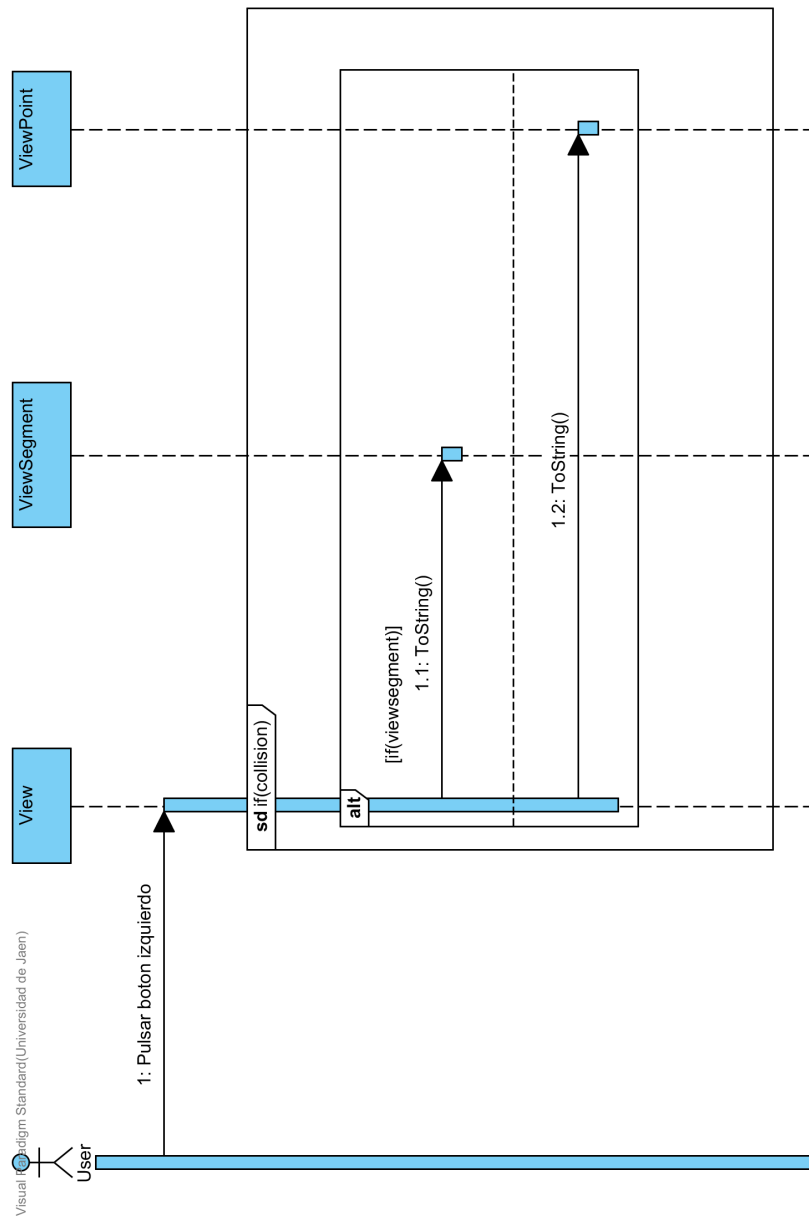


Figura 4.21: Diagrama de secuencia para mostrar información al usuario.

El método que hemos creído conveniente utilizar ha sido sobrecargar el método `ToS-tring()` que todos los objetos de Unity poseen para que aparezca información como su identificador y su posición. En la Figura 4.21 se muestra el diagrama de secuencia que hace posible este proceso. El usuario pulsa el botón izquierdo en algún lugar de la pantalla, en ese momento se crea un rayo virtual desde la cámara en la dirección pulsada y si este colisiona con algún objeto virtual, deberá comprobar de que tipo es. Tras cumplir esa condición, el objeto imprimirá alguna de sus propiedades.

Enlace a vídeo demostrativo: https://youtu.be/vFazuaLMP6g?list=PLrwxKtY_YNVtVgt4dWXRTKskA

4.6. Iteración 6

Actualización de datos

En esta iteración nos centraremos en la modificación de los valores de los objetos. Los valores temáticos, por falta de una interfaz de usuario en estos momentos, serán modificados al azar. Mientras que la geometría será alterada con la interacción del usuario.

El usuario pulsará sobre el objeto que desea mover. En el caso de que sea un objeto de tipo puntual, aparecerá una esfera de la que arrastrará hacia la nueva posición. Si es un objeto de tipo segmento, aparecerán una esfera en cada extremo. En este momento hay que tener en cuenta que los objetos están topológicamente conectados, por lo que si cambiamos un vértice que está compartido por dos o más objetos, todos se deberán mover a la nueva posición.

La forma de operar será la natural para un usuario. Seleccionará el objeto, seleccionará el vértice que desea mover y lo arrastrará por la pantalla hasta su nueva posición.

El diagrama de secuencia asociado a esta iteración puede observarse en la Figura 4.22.

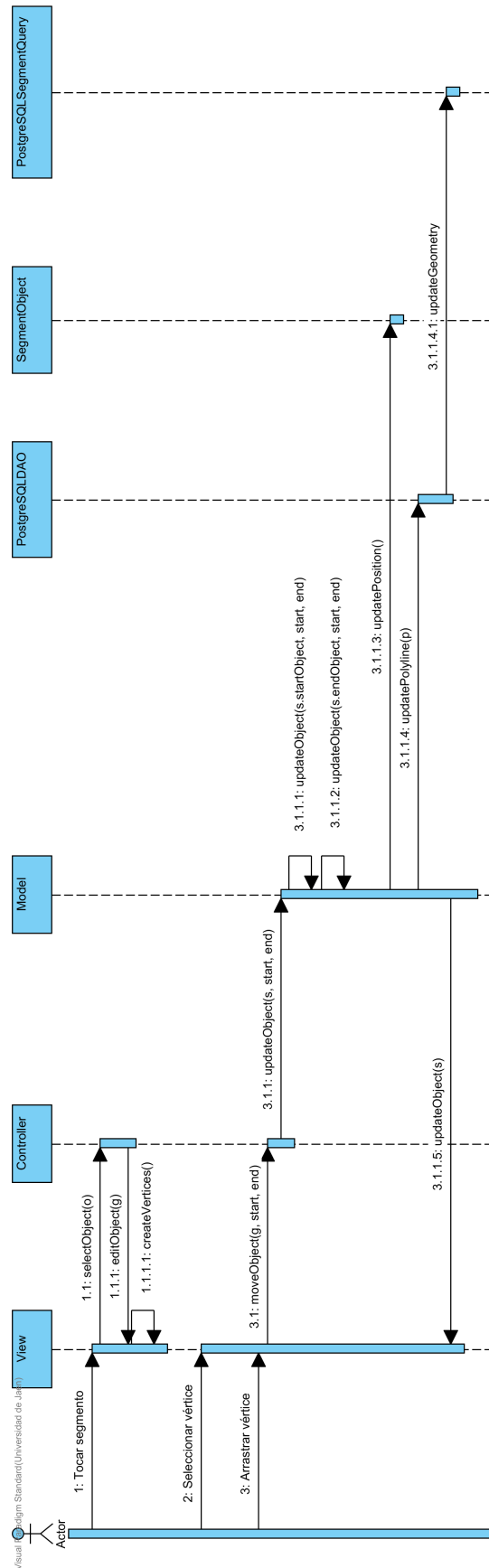


Figura 4.22: Diagrama de secuencia de la modificación de la geometría de un objeto.



Figura 4.23: Borrado de un segmento.

Enlace a vídeo demostrativo: https://youtu.be/DbEe6T7sSEk?list=PLrwxKtY_YNVtVgt4dWXRTKskA

4.7. Iteración 7

Borrado de datos

En esta iteración trataremos sobre la elección del usuario de borrar un objeto. El caso más simple es el de borrar un objeto puntual, ya que solo nos basta con eliminar su registro de la base de datos y borrar sus referencias en la tabla donde almacenamos la topología. Pero todo esto cambia cuando el usuario elige quitar un segmento. Ésto implica la creación de al menos una nueva polilínea si ese segmento es un extremo, o la división en dos polilíneas si el segmento es intermedio. En cuanto a la topología de esta casuística es muy diversa ya que implica borrados, insercciones o actualizaciones. Explicaremos este caso a través de la Figura 4.23. Al borrar el segmento intermedio de la Polilínea 1, debemos borrar la polilínea completa e insertar la Polilínea 2 y 3 en la base de datos. Esto también implica eliminar sus registros de la tabla topológica. Tras esto, con el id de la Polilínea 2 debemos insertar un registro en la tabla "connected" con el identificador del objeto puntual y el nuevo identificador. Haremos lo mismo con la Polilínea 3.

El diagrama de secuencia asociado a esta iteración puede observarse en la Figura 4.24.

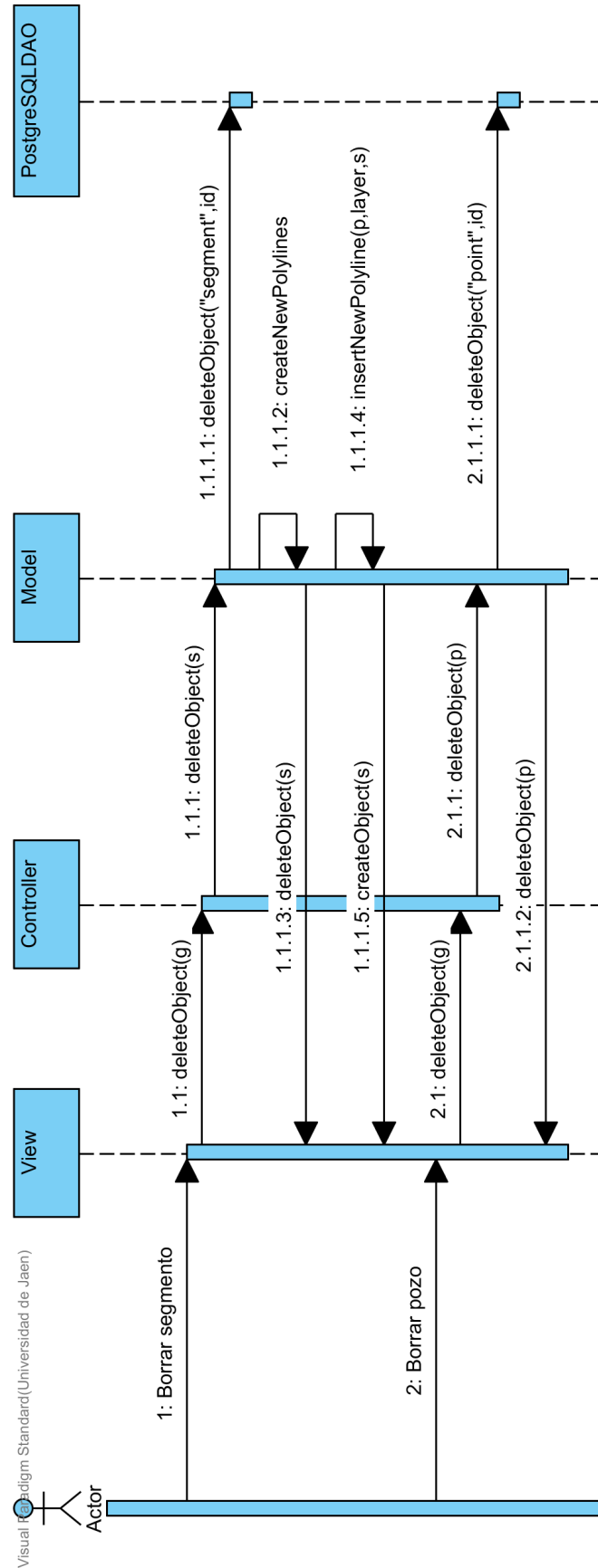


Figura 4.24: Diagrama de secuencia del borrado de objetos.

Enlace a vídeo demostrativo: https://youtu.be/4n_AXCNgT0g?list=PLrwxKtY_YNVtVgt4dWXRTKskA

4.8. Iteración 8

Insercción de datos

Con esta iteración concluiremos un SIG 3D destinado a la administración más bien para el trabajo en una oficina. En ésta abordaremos como un usuario es capaz de crear objetos.

Para este caso hemos de aplicar ciertas restricciones. Un segmento siempre debe empezar o terminar en un objeto puntual, por lo que si el usuario quiere crear una tubería deberá utilizar un elemento tipo pozo por ejemplo para crearla, o crear un nuevo pozo y después conectar tuberías a él. Esto implica la creación de objetos tanto en la tabla correspondiente como en la tabla topológica. Cabe también la posibilidad de que el usuario introduzca un segmento detrás de una polilínea ya existente, por lo que habría que modificar ésta en la base de datos y en nuestro software.

Mostramos el diagrama de secuencia en la Figura 4.25

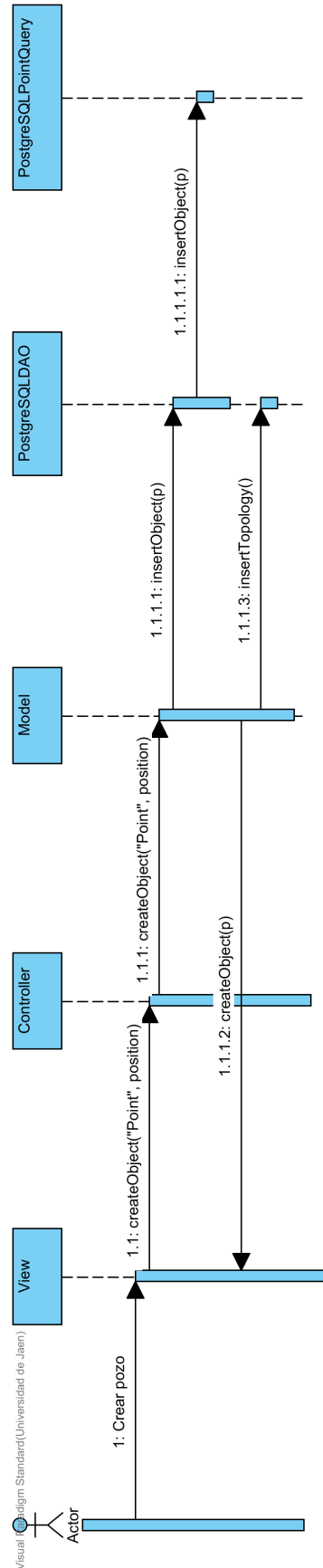


Figura 4.25: Diagrama de secuencia para la insercción de nuevos objetos.

Enlace a vídeo demostrativo: https://youtu.be/hZoklMA8tnU?list=PLrwxKtY_YNVtVgt4dWXRTKskA

4.9. Iteración 9

Interfaz de usuario

Una vez implementadas todas las funcionalidades básicas de nuestro software, es hora de incluir la interfaz de usuario. Ésto ayuda a cualquier persona a entender mejor lo que está haciendo en cada momento, trabajando así de una forma más natural.

Para crearla necesitaremos diseñar las ventanas y formularios que el usuario se encontrará en su interacción con el programa. En las siguientes subsecciones mostraremos su aspecto y haremos una explicación sobre ellas.

Pantalla principal



Figura 4.26: Pantalla inicial de la aplicación tras cargar una capa.

En la Figura 4.26 tenemos lo que será la pantalla inicial de nuestra aplicación. Abajo a la izquierda observamos un mini mapa de desplazamiento rápido. Pulsando sobre ese cuadrado podremos cambiar la posición de la cámara rápidamente a cualquier lugar del mapa. Las partes en amarillo representan un cúmulo de datos en esa posición. En la parte derecha tenemos un cuadro de botones. El primero de ellos sirve para alternar entre la cámara de libre y la cenital. El segundo muestra las capas cargadas en ese momento. El tercero muestra el panel destinado a cargar nuevas capas.

Añadir una capa

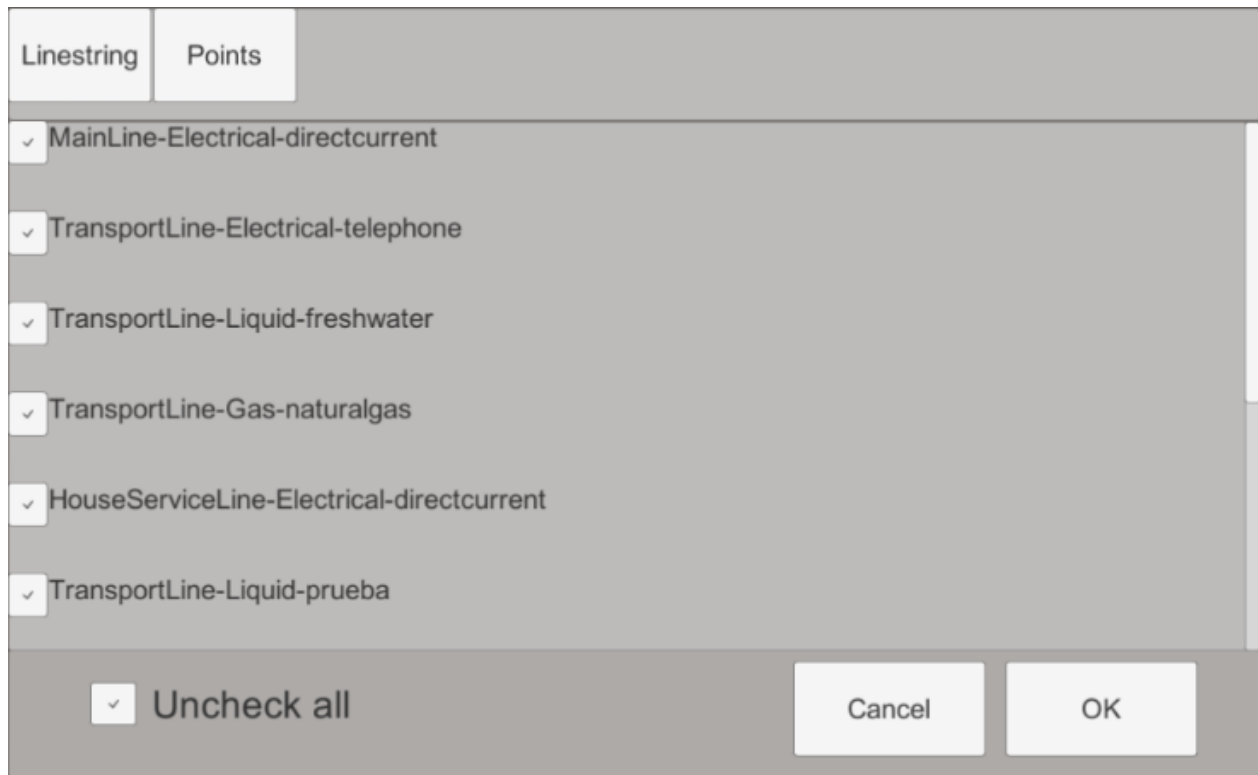


Figura 4.27: Pantalla destinada a la carga de nuevas capas.

Esta parte de la interfaz está destinada a la inclusión de capas tal y como refleja la Figura 4.27. En la parte superior observamos botones a modo de pestañas con los tipos de datos que la aplicación es capaz de cargar, en este momento puntos y polilíneas. En la parte central se encuentran los datos a cargar en formato "Tipo de objeto- Red - Naturaleza de la red". Todos aquellos elementos que estén activos serán cargados si pulsamos el botón "OK". En la parte inferior izquierda encontramos un botón para desactivar y activar todos los objetos de una sola vez.

Capas cargadas

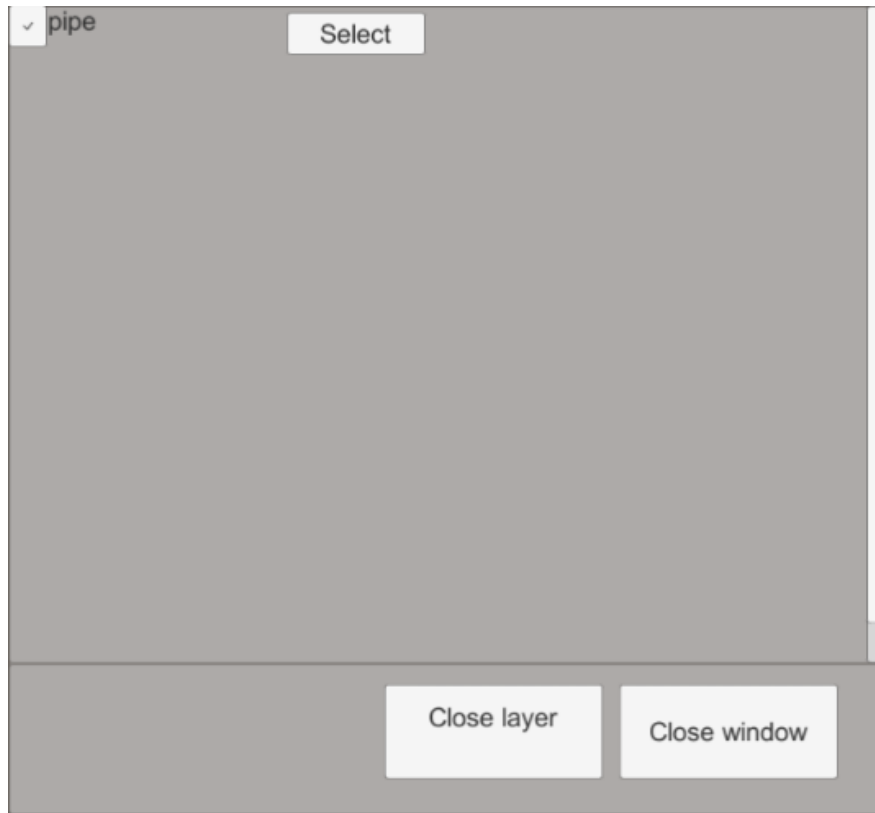


Figura 4.28: Pantalla donde se muestran las capas cargadas.

El panel que representa la Figura 4.28 sirve para ocultar o cerrar las capas que hemos cargado. En el se nos presenta el nombre de la capa, un botón de selección y un checkbox para ocultar y mostrar la capa sin borrarla de memoria. Para eliminar una capa debemos pulsar sobre el botón de selección de la capa y pulsar el botón que dice "Close layer".

Seleccionar objeto

Status	InUse
Function	Distribution
Material	PVC C
Location quality	surveyed
Elevation quality	interpolated
Line class	HouseServiceLine

Edit

Delete

Cancel

Save

Figura 4.29: Formulario de edición y de acciones.

En la Figura 4.29 encontramos el formulario que aparecerá cuando un usuario selecciona un objeto. En él mostramos la información relativa al objeto. Esta información puede ser modificada cambiando los valores de los campos de texto. Más abajo encontramos una barra donde se sitúan las acciones que se pueden realizar con el objeto, en este caso borrar y editar. Si hubiera acciones propias de un objeto como abrir o cerrar una válvula, aparecerían en este lugar.

Creación de objetos

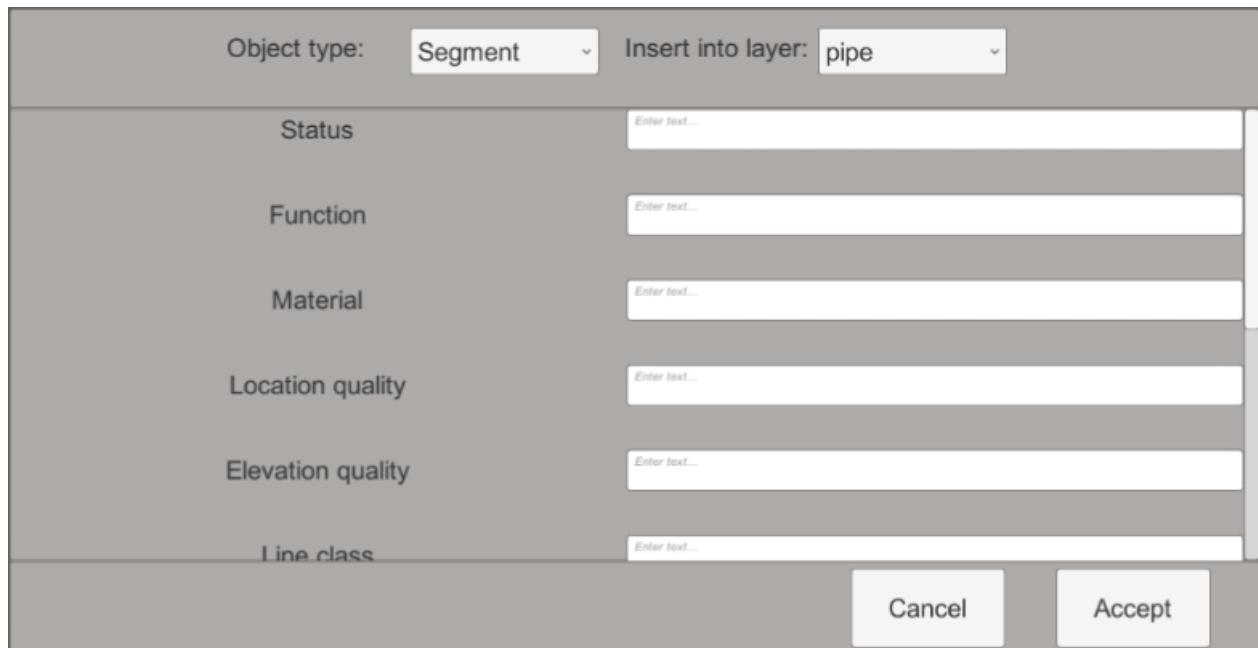
The image shows a software interface for creating objects. At the top, there are two dropdown menus: 'Object type:' with 'Segment' selected, and 'Insert into layer:' with 'pipe' selected. Below these are six text input fields, each with a label to its left: 'Status', 'Function', 'Material', 'Location quality', 'Elevation quality', and 'Line class'. Each input field contains the placeholder text 'Enter text...'. At the bottom right of the form are two buttons: 'Cancel' and 'Accept'.

Figura 4.30: Formulario de creación de objetos.

Por último, mostramos el formulario de creación de nuevos objetos en la Figura 4.30. En él se nos presentan tres partes. La primera está compuesta por dos "ComboBox" donde el usuario debe elegir el tipo de dato que quiere insertar y la capa. Si no tenemos cargada la capa adecuada para la creación de un tipo de objeto, éste no podrá ser creado y no aparecerá como opción. En la parte central nos encontramos los campos que obligatoriamente deben ser rellenados para crear un nuevo objeto. En la parte inferior nos encontramos con los botones para aceptar y cancelar la creación del objeto.

4.10. Iteración 10

Visualización con Realidad Aumentada

Una vez desarrollado todo el software y la interfaz de usuario, nos disponemos a la realización de la herramienta de realidad aumentada. Al basarnos en los principios de la

programación orientada a objetos, junto con sus buenas prácticas, podremos reaprovechar buena parte de lo ya implementado. En esta iteración no solo debemos resolver la visualización de los datos a través de una tablet, sino la extensión del problema para situar al usuario y orientarlo para que los datos aparezcan en su lugar correcto.

La realidad aumentada se ha utilizado en su mayor parte en zonas acotadas y cerradas. Por lo que sacar esta tecnología a un mundo abierto supone un estudio de las distintas posibilidades.

En primera instancia se puede pensar en la utilización de los sensores de la tablet. Como método de posicionamiento podemos utilizar el GPS integrado y como método de orientación, la brújula digital. Aunque parezca una buena idea, son sensores que no tienen precisión. Recordamos que el error del GPS puede llegar hasta los 50 metros y de media, la señal indica 5 metros de desviación. Por otra parte, el uso de la brújula no es recomendable ya que se descalibra a menudo y además es sensible al entorno que le rodea. Por lo tanto no utilizaremos éste método.

Lo siguiente que pensamos fue en el uso de algún framework de mapas que pudiera obtener un plano con las calles y edificios cercanos. El usuario elegiría donde está situado y arrastraría una flecha hacia donde estuviera mirando.



Figura 4.31: Captura del framework UnitySlippyMap.

La primera opción que nos encontramos fue "UnitySlippyMap". Esta herramienta, como vemos en la Figura 4.31, ofrece un plano con una textura del mapa deseado. Para ello hace uso de llamadas a la API de "Open Street Map". El problema de esta librería es que aún se encuentra en fase de pruebas y en ocasiones no funciona correctamente. Aún así conseguimos adaptarlo a nuestro proyecto. Para la orientación no es el método más adecuado ya que hay lugares donde el usuario no puede identificar fácilmente hacia donde está mirando y es poco probable que acierte totalmente con la flecha, haciendo así que el ángulo esté rotado más de la cuenta. Hicimos pruebas con este sistema, y además de los problemas ya especificados, observamos que conforme llevábamos a lugares que no tenían la misma altitud que el lugar inicial, los objetos subterráneos salían del suelo. Esto es debido a que Google Tango no es capaz de saber si estamos bajando o subiendo mientras andamos. Por lo tanto descartamos nuevamente esta opción y nos dispusimos a buscar otra.



Figura 4.32: Aplicación creada con Mapbox y Unity.

Encontramos la solución a todos los problemas expuestos anteriormente con el uso de la herramienta Mapbox y su SDK para Unity. Este paquete nos ofrece un mapa de la ciudad con el mismo formato que "UnitySlippyMap" pero también incorpora la orografía del terreno y la representación virtual de los edificios, consiguiendo resultados tan espectaculares como los mostrados en la Figura 4.32. El método de posicionamiento que ofrecemos al usuario es la unión de los dos puntos anteriores, dándole la posibilidad de elegir entre posicionarse con la señal GPS o seleccionando su ubicación. Para la orientación, hemos decidido aprovechar la representación virtual de los edificios. Encendemos la cámara del dispositivo y el usuario será el encargado de superponer los edificios virtuales y reales. Al tener la orografía del terreno, a la misma vez que el usuario camina con la tablet en la mano, el objeto virtual que representa al usuario también se moverá por el terreno de la aplicación.

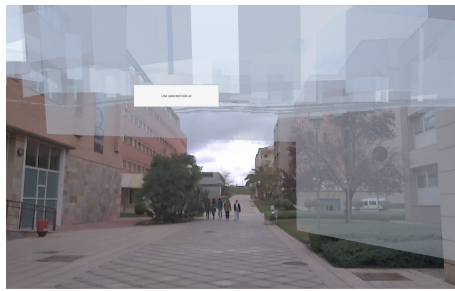
Explicaremos paso a paso lo comentado en el párrafo anterior ayudándonos de la Figura 4.33. En las Subfiguras a y b encontramos dos puntos distintos encima del mapa, el azul representa la posición del usuario según la señal GPS, mientras que el verde es el usuario quien proporciona la posición donde el cree que está. Tras esto, la representación



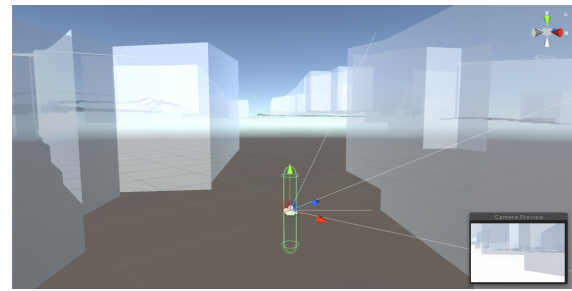
(a)



(b)



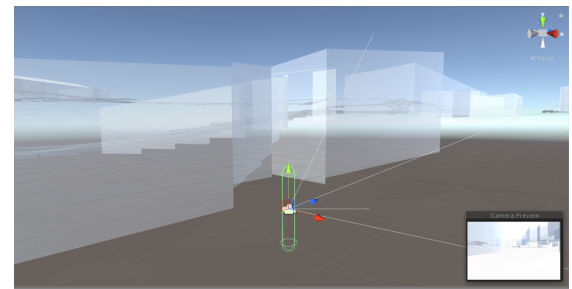
(c)



(d)



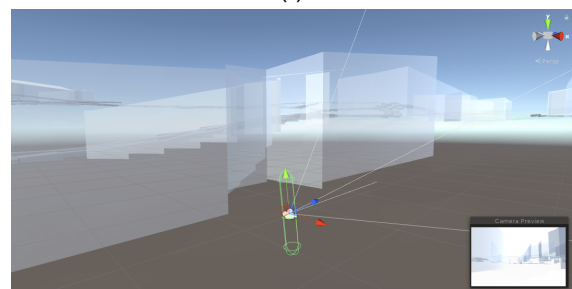
(e)



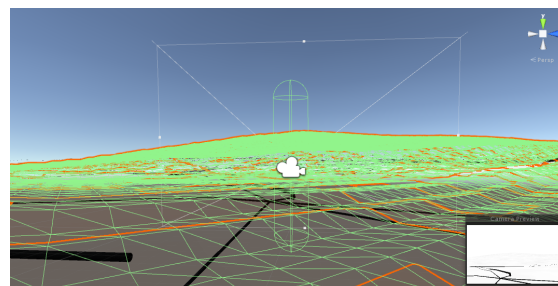
(f)



(g)



(h)



(i)

Figura 4.33: Pasos que sigue el usuario para posicionar y orientar la tablet.

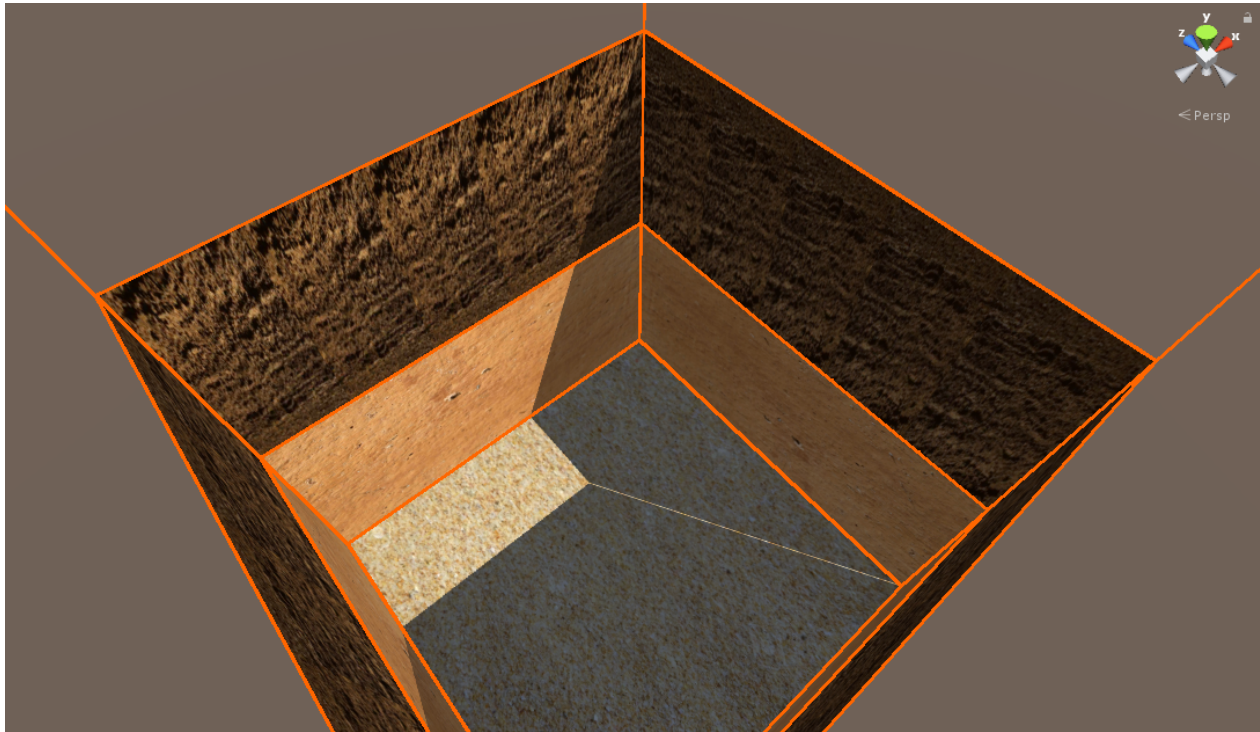


Figura 4.34: Objeto que simula una excavación debajo del usuario.

virtual del usuario es situada encima del terreno, en la posición que el eligió. La Subfigura c muestra lo que el usuario ve a través de la tablet, y en la Subfigura d lo que está pasando ese momento internamente en el software. El usuario arrastra su dedo por la pantalla, acoplando los edificios virtuales y los reales, esto internamente significa la rotación del mapa entorno a la posición del usuario. Si en este momento ambos coinciden, podremos poner en marcha el sistema de realidad aumentada. En caso contrario debemos realizar un ajuste de altura. El usuario arrastrará su dedo hacia arriba y hacia abajo para que los edificios virtuales coincidan con los reales, como vemos en la Subfigura g. En la Subfigura h se aprecia cómo esta vez hemos cambiado el punto de mira de la cámara que representa al usuario. A partir de este momento podremos navegar visualizando e interaccionando las redes subterráneas con la ayuda de la realidad aumentada. En la Subfigura i vemos la representación del terreno en el software.

Para dotar de realismo a la aplicación, hemos desarrollado un objeto virtual representando una excavación en el terreno como vemos en la Figura 4.34. Esta caja no permite ver lo que hay fuera de sus límites, por lo que el usuario deberá apuntar con la tablet

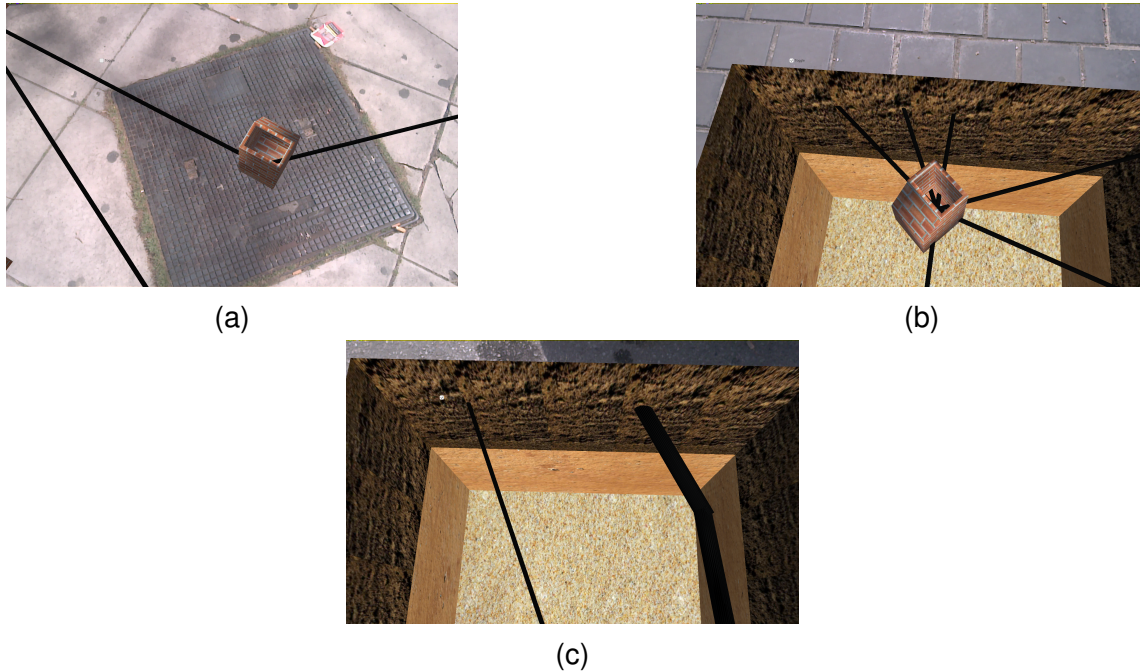


Figura 4.35: Resultado de la aplicación de realidad aumentada.

hacia el suelo para observar las infraestructuras subterráneas.

En la Figura 4.35 observamos los resultados obtenidos de esta iteración. En la Subfigura a se ve un pozo que está situado muy cerca del verdadero, esto es diferenciable ya que podemos observar la tapa de acceso a dicho pozo. En la Subfigura b vemos un pozo conectado con un conjunto de tuberías. Por último, en la Subfigura c vemos dos tuberías de distinto tamaño.

Capítulo 5

Resultados

Este trabajo puede suponer un gran avance para cambiar la forma de trabajar de las empresas encargadas del mantenimiento de las infraestructuras subterráneas. Cuando un operario necesite realizar algún trabajo sobre una tubería podrá usar el software, ver donde está situada y dar las órdenes de excavación. Las dos empresas colaboradoras han podido observar el funcionamiento de nuestros resultados y entendiendo que el proyecto se encuentra en una fase de prototipo, tras analizar la propuesta han dado el visto bueno a esta forma de trabajar, dando la posibilidad en un futuro de usar nuestra herramienta.

Respecto al dispositivo de realidad aumentada que hemos utilizado, somos conscientes de que no es una tecnología consolidada en estos momentos y se puede hablar más de un prototipo. Cada día son más las personas que están interesadas en usar esta tecnología por lo que sus características irán mejorando a lo largo del tiempo y haciendo cada más útiles estos sistemas. Muchas de las funcionalidades del SDK de Google Tango están de forma experimental por lo que no nos queda más que esperar algún tiempo y podremos trabajar de esta nueva forma.

Hemos completado todos los puntos que hemos establecido en las condiciones para la finalización del proyecto. Aunque solo sea un prototipo, han quedado establecidas las bases para futuras funcionalidades. Finalmente se han implementado dos sistemas, una herramienta con capacidades de SIG 3D para equipos de escritorio y otro para los operarios en la calle. El primero incluye funcionalidades de edición que otros SIG no poseen. El



Figura 5.1: Trimble R1.

segundo cuenta con la novedad de la utilización de la realidad aumentada para observar las redes del subsuelo.

Otra de las novedades que han sido incluidas en este proyecto es el uso de objetos que están topológicamente conectados. Esto facilita el entendimiento de la red. Otra ventaja es editar un objeto, todos los conectados a él se moverán juntos.

Enlace a vídeo demostrativo (GIS 3D con realidad aumentada): <https://youtu.be/pYT9u-WBf6o>

Enlace a vídeo demostrativo (GIS 3D de escritorio): https://youtu.be/Tp6AgFR_ptU

5.1. Pruebas

Hemos realizado pruebas donde el usuario utiliza varios métodos de posicionamiento. Con ello descubriremos cual es la mejor forma de posicionarse en nuestra aplicación. Hemos probado tres formas de posicionamiento:

- El usuario selecciona su posición en el mapa.
- Utilizamos la señal GPS que nos proporciona el dispositivo.

- Utilizamos un GPS submétrico externo, que conectado mediante Bluetooth al dispositivo, nos posiciona con una precisión centimétrica. Utilizaremos el modelo R1 de Trimble (Figura 5.1). Este tipo de dispositivo es capaz de conseguir precisión centimétrica dado que puede conectarse con múltiples satélites como GPS, GLONASS, Galileo o BeiDou además de soportar correcciones como SBAS, VRS o redes RTX. En nuestro caso hemos utilizado una conexión a la Red Andaluza de Posicionamiento mediante el protocolo NTRIP, consiguiendo una precisión de hasta 78 centímetros. Este aparato se conecta a cualquier otro dispositivo a través de una conexión Bluetooth. En el caso de Android, será necesario habilitar la opción de "Ubicaciones simuladas" y mantener la aplicación proporcionada por Trimble en segundo plano. A partir de ahora, la posición GPS que trata la tablet o el smartphone es la que es recibida por dicho dispositivo.

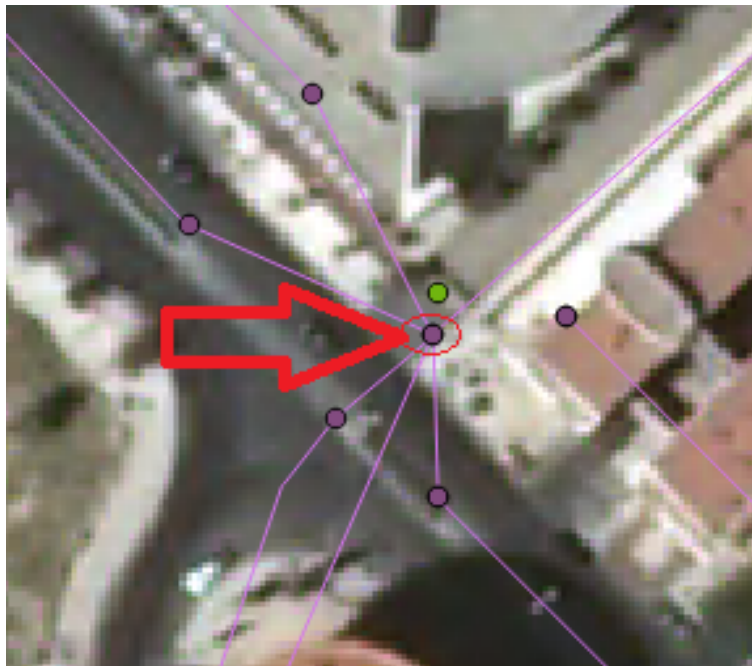


Figura 5.2: Figura de referencia para medir la fiabilidad de nuestra aplicación

Ahora es el momento de ver cual es la mejor forma entre las tres posibles de posicionarnos. Para determinarlo buscaremos un lugar donde los datos sean inconfundibles con otros lugares cercanos y mediremos la distancia existente desde nuestra posición hasta el punto donde se muestre dichos datos en la pantalla de Tango. Utilizaremos la Figura

5.2 como referencia. En ella observamos puntos y líneas de color morado, esos serán nuestros datos de tuberías y pozos. El punto verde identifica al lugar desde donde nos situaremos para posicionarnos y orientar la vista. El punto señalado con la flecha será el que buscaremos con el dispositivo de realidad aumentada.

Empezaremos analizando el error cometido con el GPS integrado del dispositivo. Este método podemos afirmar que no es nada adecuado para nuestro sistema, aún así daremos la posibilidad al usuario de seleccionarlo. Tras diversos intentos hemos medido que el error se sitúa entre los tres y cuatro metros. Utilizando este posicionamiento a la hora de hacer un mantenimiento produciría multitud de fallos tras excavar para reparar un elemento.

El segundo método que analizamos será mediante la selección que el usuario cree tener en un mapa. De esta forma hemos conseguido un error medio en torno a los 70 cm. Parece un buen método siempre y cuando se confíe en la habilidad del usuario para pulsar sobre un lugar. A la hora de seleccionar un sitio sería aconsejable que éste fuera característico como el final de una calle o cerca de la esquina de un edificio.

La manera de posicionamiento que nos ha dado mejor precisión ha sido con el uso del GPS submétrico conectado con la Red Andaluza de Posicionamiento. Hemos conseguido bajar el error medio hasta los 32 cm en el mejor de los casos y 40 cm en el peor. El único inconveniente es que hemos tenido que esperar cerca de tres minutos para recibir una corrección decente. Si el trabajador está realizando tareas de mantenimiento, este es el mejor método sin duda pero si es necesario realizar una reparación urgente tal vez no sea la forma más apropiada, dado su tiempo de espera. Para casos donde se necesita rapidez se debería optar por elegir la posición.

Bibliografía

- [BDK15] Amir H. Behzadan, Suyang Dong, and Vineet R. Kamat. Augmented reality visualization: A review of civil infrastructure system applications. *Advanced Engineering Informatics*, 29(2):252–267, apr 2015.
- [BKSS90] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. The r^* -tree: An efficient and robust access method for points and rectangles. *SIGMOD Rec.*, 19(2):322–331, May 1990.
- [BZ11] Martin Breunig and Sisi Zlatanova. 3d geo-database research: Retrospective and future directions. *Computers & Geosciences*, 37(7):791–803, jul 2011.
- [Ege89] M. J. Egenhofer. A formal definition of binary topological relationships. In *3rd International Conference, FODO 1989 on Foundations of Data Organization and Algorithms*, pages 457–472, New York, NY, USA, 1989. Springer-Verlag New York, Inc.
- [FB74] R. A. Finkel and J. L. Bentley. Quad trees a data structure for retrieval on composite keys. *Acta Inf.*, 4(1):1–9, March 1974.
- [Gut84] Antonin Guttman. R-trees: A dynamic index structure for spatial searching. *SIGMOD Rec.*, 14(2):47–57, June 1984.
- [HNP95] Joseph M. Hellerstein, Jeffrey F. Naughton, and Avi Pfeffer. Generalized search trees for database systems. In *Proceedings of the 21th International Conference on Very Large Data Bases, VLDB '95*, pages 562–573, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc.

- [LSP15] Sangho Lee, Jangwon Suh, and Hyeong-Dong Park. BoreholeAR: A mobile tablet application for effective borehole database visualization using an augmented reality technology. *Computers & Geosciences*, 76:41–49, mar 2015.
- [Mic] Microsoft. Microsoft sql server. Accessed: 05-06-2017.
- [MyS] MySQL. Mysql. Accessed: 05-06-2017.
- [NHS81] J. Nievergelt, H. Hinterberger, and K. C. Sevcik. *The grid file: An adaptable, symmetric multi-key file structure*, pages 236–251. Springer Berlin Heidelberg, Berlin, Heidelberg, 1981.
- [OGC10] OGC. Ogc standards: Gml in jpeg 2000 for geographic imagery encoding specifications, 2010.
- [Ora] Oracle. Oraclespatial. Accessed: 05-06-2017.
- [Posa] PostGIS. Postgis. Accessed: 05-06-2017.
- [Posb] PostgreSQL. Postgresql. Accessed: 05-06-2017.
- [Sam05] Hanan Samet. *Foundations of Multidimensional and Metric Data Structures (The Morgan Kaufmann Series in Computer Graphics and Geometric Modeling)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005.
- [Spa] SpatialLite. Spatiallite. Accessed: 05-06-2017.
- [SZR12] Gerhard Schall, Stefanie Zollmann, and Gerhard Reitmayr. Smart vidente: advances in mobile augmented reality for interactive visualization of underground infrastructure. *Personal and Ubiquitous Computing*, 17(7):1533–1549, sep 2012.
- [Tam82] Markku Tamminen. Efficient spatial access to a data base. In *Proceedings of the 1982 ACM SIGMOD International Conference on Management of Data*, SIGMOD '82, pages 200–206, New York, NY, USA, 1982. ACM.

- [ZHC] Sisi Zlatanovaa, Daniel Holwegb, and Volker Coorsc. Geometrical and topological models for real-time gis.
- [ZHHL15] Xiaolei Zhang, Yong Han, DongSheng Hao, and Zhihan Lv. ARPPS: Augmented reality pipeline prospect system. In *Neural Information Processing*, pages 647–656. Springer Nature, 2015.
- [ZHHL16] Xiaolei Zhang, Yong Han, DongSheng Hao, and Zhihan Lv. ARGIS-based outdoor underground pipeline information system. *Journal of Visual Communication and Image Representation*, 40:779–790, oct 2016.
- [ZPV16] Stefanie Zollmann, Christian Poglitsch, and Jonathan Ventura. VISGIS: Dynamic situated visualization for geographic information systems. In *2016 International Conference on Image and Vision Computing New Zealand (IVCNZ)*. Institute of Electrical and Electronics Engineers (IEEE), nov 2016.

Índice de figuras

1.1. Plano de la infraestructura subterránea de una ciudad.	8
1.2. Plano en formato CAD de los elementos del subsuelo.	9
1.3. Modelado de los datos de una base de datos espacial.	13
1.4. SmartVidente.	14
1.5. Estructura de un sistema ARMOR.	16
1.6. Resultados usando ARMOR.	16
1.7. Resultados de BoreholeAR.	17
1.8. Posición inicial del dispositivo en ARGIS.	18
1.9. iPad.	23
1.10. Tablet con el sistema operativo Android.	24
1.11. Smartphone con el sistema operativo Android.	25
1.12. Gafas con RA Microsoft Hololens.	25
1.13. Tablet con RA Google Tango.	26
1.14. Smartphone con RA Lenovo Phab 2.	28
1.15. Smartphone con RA Asus ZenPhone 3.	29
2.1. Arqueta	31
2.2. Esquema de como puede ser construido un pozo	32
2.3. Pozo	32
2.4. Imbornal	34
2.5. Conducto	35
2.6. Conducción	35
2.7. Esquema de una red de saneamiento	36

2.8. Boca de riego	37
2.9. Caudalímetro	38
2.10. Contador	39
2.11. Desagüe	39
2.12. Regulador de presión	40
2.13. Tubería	40
2.14. Canalización	41
2.15. Cámara	42
3.1. Fases del proceso de desarrollo del proyecto.	46
3.2. Raíz de los diagramas de caso de uso.	49
3.3. Diagrama de caso de uso "Operaciones CRUD".	50
3.4. Diagrama de caso de uso "Visualización de los datos".	50
3.5. Diagrama de paquetes.	51
3.6. Diagrama de clases.	53
3.7. Diagrama de clases del paquete GIS3D.	61
3.8. Diagrama de clases del paquete IO.	65
3.9. Diagrama de clases del paquete Graphics.	67
3.10. Diagrama de clases del paquete Core.	71
3.11. Diagrama de clases del paquete UI.	73
3.12. Diagrama Entidad Relación.	77
3.13. Planificación temporal.	78
4.1. Carga de los datos con FME Data Inspector.	83
4.2. Datos cargados en FME Data Inspector.	84
4.3. Red de saneamiento y de abastecimiento.	85
4.4. Simbología de los objetos de las redes.	86
4.5. Simbología en formato CAD usada por la empresa encargada de la gestión del agua en Jaén	87
4.6. Información sobre los pozos, a la izquierda el círculo exterior y a la derecha el círculo interior	88

4.7. Primer diseño de nuestra base de datos.	89
4.8. Diseño del núcleo propuesto en "Utility Network ADE" siguiendo el estándar CityGML.	91
4.9. Diseño de los objetos propuesto en "Utility Network ADE" siguiendo el estándar CityGML.	92
4.10. Diseño de las redes propuesto en "Utility Network ADE" siguiendo el estándar CityGML.	93
4.11. Reader de FME para lectura de entidades.	96
4.12. Flujo de FME para entidades puntuales.	98
4.13. Procesamiento de entidades puntuales.	100
4.14. Flujo de FME para entidades de tipo polilínea.	101
4.15. Procesamiento de entidades de tipo polilínea.	102
4.16. Representación tridimensional de los objetos.	107
4.17. Materiales que serán aplicados a los objetos.	107
4.18. Diagrama de secuencia de la visualización de objetos.	108
4.19. Diagrama de secuencia del movimiento de la cámara cenital.	110
4.20. Diagrama de secuencia del movimiento de la cámara libre.	110
4.21. Diagrama de secuencia para mostrar información al usuario.	112
4.22. Diagrama de secuencia de la modificación de la geometría de un objeto.	114
4.23. Borrado de un segmento.	115
4.24. Diagrama de secuencia del borrado de objetos.	116
4.25. Diagrama de secuencia para la insercción de nuevos objetos.	118
4.26. Pantalla inicial de la aplicación tras cargar una capa.	120
4.27. Pantalla destinada a la carga de nuevas capas.	121
4.28. Pantalla donde se muestran las capas cargadas.	122
4.29. Formulario de edición y de acciones.	123
4.30. Formulario de creación de objetos.	124
4.31. Captura del framework UnitySlippyMap.	126
4.32. Aplicación creada con Mapbox y Unity.	127
4.33. Pasos que sigue el usuario para posicionar y orientar la tablet.	128

4.34. Objeto que simula una excavación debajo del usuario.	129
4.35. Resultado de la aplicación de realidad aumentada.	130
5.1. Trimble R1.	132
5.2. Figura de referencia para medir la fiabilidad de nuestra aplicación	133
5.3. Pantalla inicial.	143
5.4. Formulario de carga de capas de tipo polilínea.	144
5.5. Formulario de carga de capas de tipo punto.	145
5.6. Pantalla inicial con datos cargados.	146
5.7. Formulario de capas abiertas.	147
5.8. Formulario de selección de un objeto.	148
5.9. Edición de un objeto.	149
5.10. Inserccion de un objeto.	150
5.11. Posicionamiento del usuario en la aplicación de realidad aumentada.	151

Índice de cuadros

3.1. Presupuesto de recursos humanos	80
3.2. Características del servidor contratado	81
3.3. Presupuesto de hardware	81
3.4. Presupuesto de licencias	81
3.5. Presupuesto de otros recursos	82
3.6. Coste total del proyecto	82

Anexo A: Manual de usuario

Aplicación de escritorio

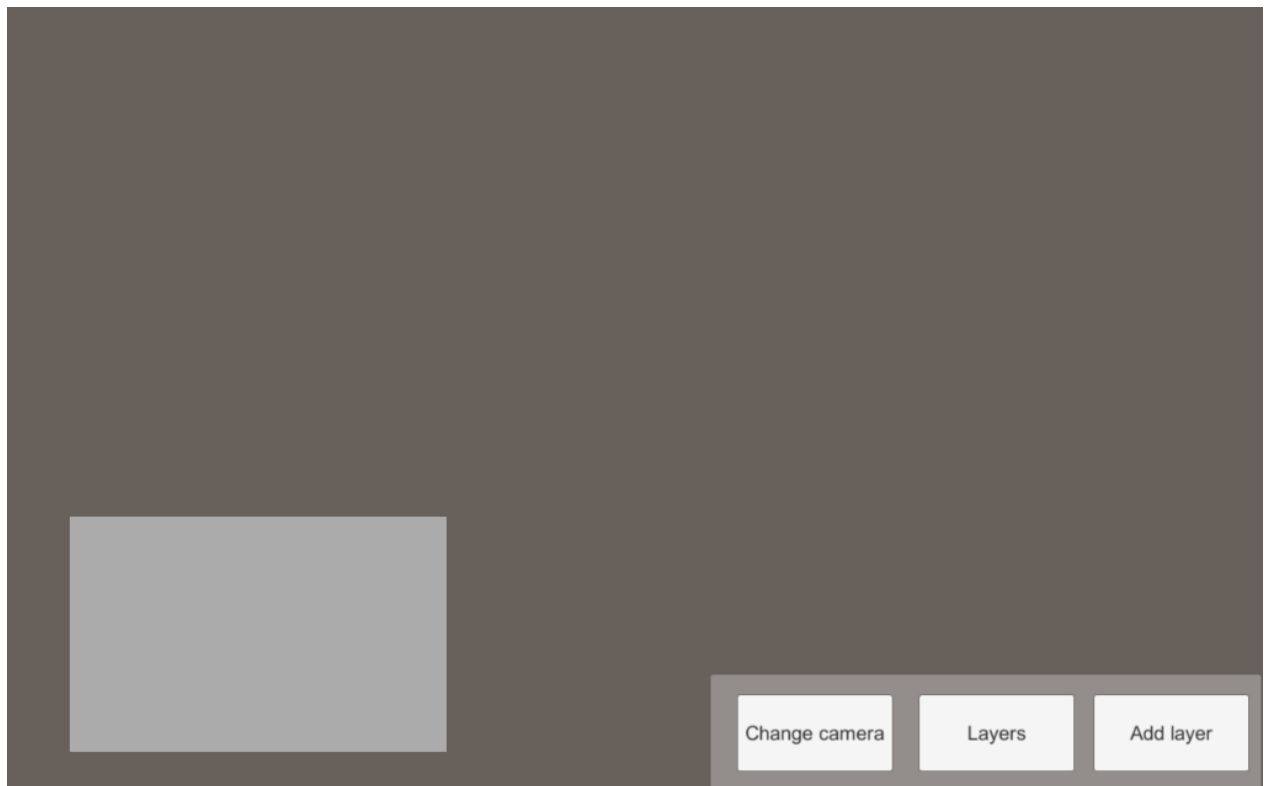


Figura 5.3: Pantalla inicial.

Nada más cargar la aplicación, nos aparecerá la pantalla reflejada en la Figura 5.3. En ella distinguimos las siguientes partes:

- Minimapa: En él se muestra donde hay objetos cargados en el mapa. Pulsando sobre él, se realizará un desplazamiento rápido.

- **Menú de botones:** En él se organizan las operaciones más básicas de la aplicación. Contiene los siguientes botones:
 - **Change camera:** Alterna entre cámara cenital y cámara libre.
 - **Layers:** Menú de capas, explicado en la Subsección 5.1.
 - **Add layer:** Menú de carga de nuevas capas, explicado en la Subsección 5.1.
- **Zona de visualización:** Comprende la mayor parte de la pantalla, en ella se dibujarán los datos cargados.

Cargar una nueva capa

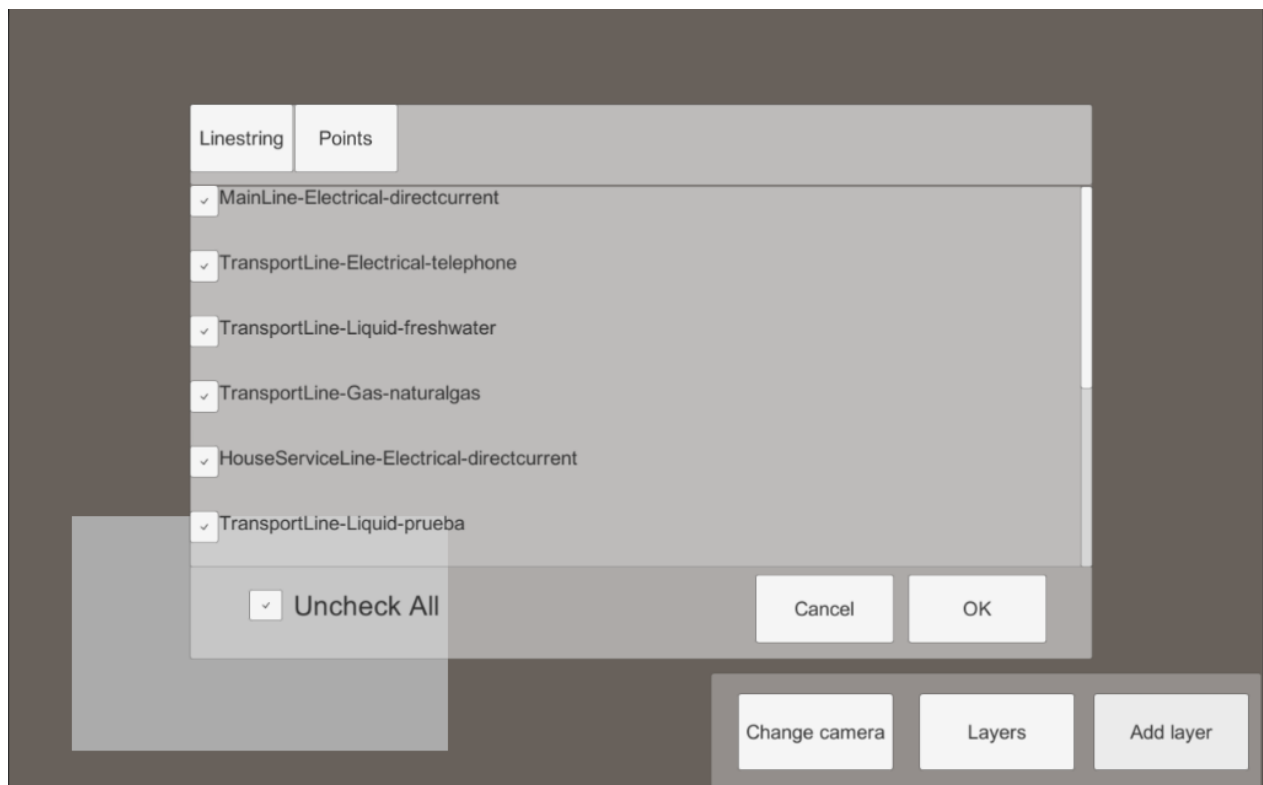


Figura 5.4: Formulario de carga de capas de tipo polilínea.

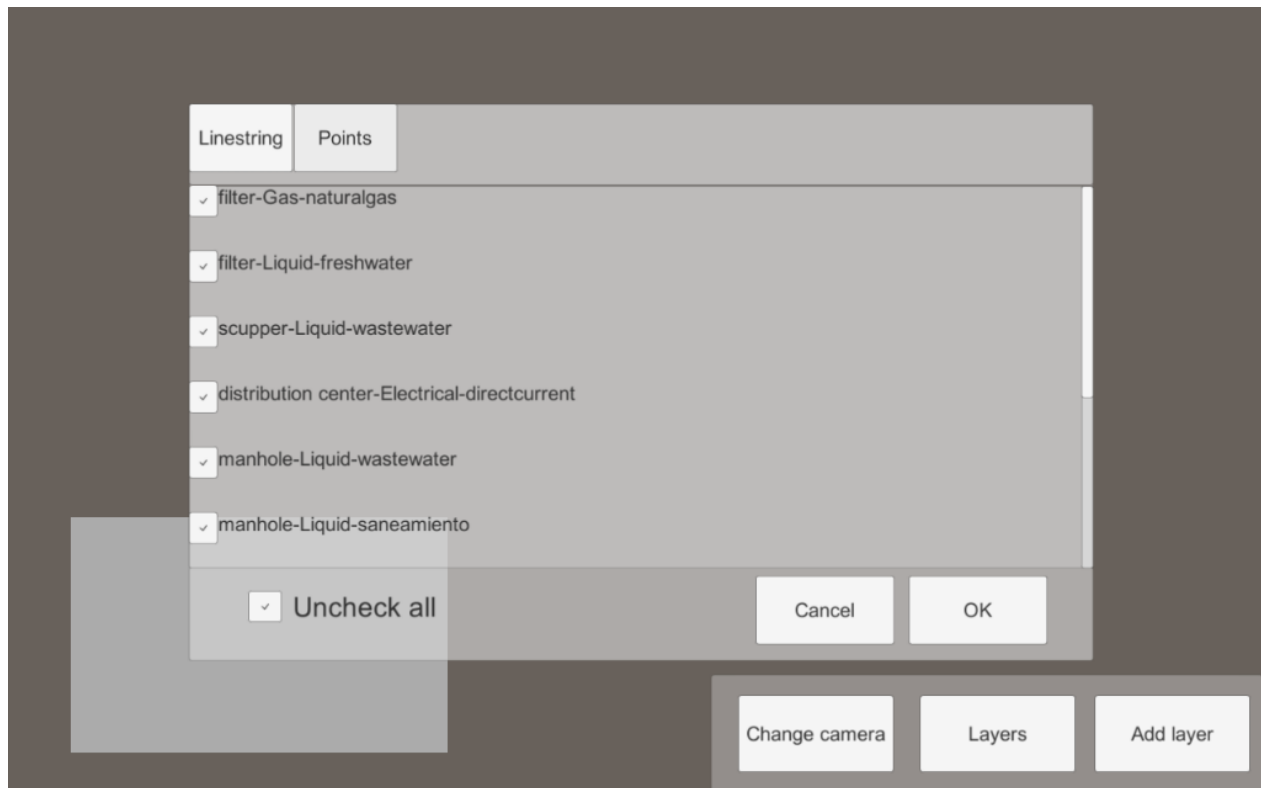


Figura 5.5: Formulario de carga de capas de tipo punto.

Este formulario tiene como objetivo cargar nuevas capas para visualizarlas e interactuar con ellas. Se divide en tres partes:

- Menú de pestañas: Al pulsar sobre una de ellas cambia el tipo de dato que vamos a cargar.
- Menú de datos: En esta parte, los datos que estén activos serán cargados tras pulsar el botón "OK". Los datos se encuentran en formato: "Tipo de objeto - Red - Tipo de red".
- Menú de acciones: Nos encontramos con un control de selección y deselección rápida y con los botones de cancelar y aceptar.



Figura 5.6: Pantalla inicial con datos cargados.

Tras seleccionar los objetos que queremos crear, la pantalla principal queda como en la Figura 5.6.

Menú de capas



Figura 5.7: Formulario de capas abiertas.

Esta parte es la encargada de interactuar con las capas previamente cargadas. Analizamos el formulario mostrado en la Figura 5.7:

- Capas: Por cada capa aparecerán tres controles:
 - Ocultar/Mostrar capa: Al seleccionar o deseleccionar el control se mostrará o se ocultará la capa.
 - Nombre: Nombre identificativo de la capa.
 - Botón de selección: Al pulsarlo, la capa queda activa.
- Menú de botones: Incluye los botones para operar con capas:
 - Close layer: Cierra la capa activa.
 - Close window: Cierra el formulario.

Selección de un objeto

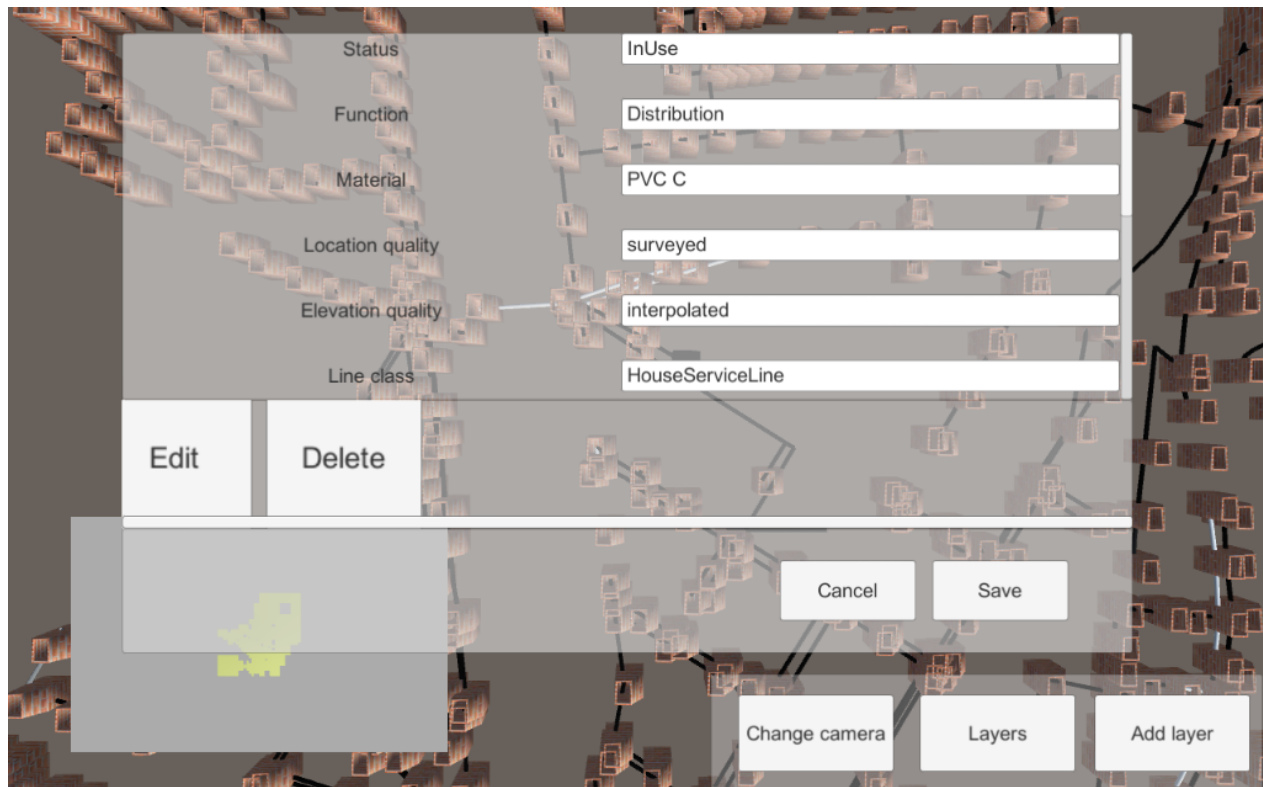


Figura 5.8: Formulario de selección de un objeto.

Al hacer click sobre un objeto, aparece un formulario con los datos del objeto que hemos seleccionado (Figura 5.8). Este informe se divide en:

- Menú de información: Se muestra la información del objeto, sus propiedades pueden ser cambiadas modificando los valores de los campos de texto.
- Menú de operaciones: Se realizará una acción sobre el objeto seleccionado. Las acciones que puede realizar el usuario en este momento son:
 - Editar la geometría del objeto. Explicado en la Subsección 5.1.
 - Borrar un objeto
- Menú de botones: Incluye los botones de cancelar y de guardar los datos que hemos cambiado.

Edición de la geometría



Figura 5.9: Edición de un objeto.

Tras pulsar sobre el botón de edición del formulario 5.1, nos aparecerá un vértice si la operación de edición es sobre un objeto puntual, y dos si se trata de una tubería, uno en cada extremo. Para mover el objeto deberemos pulsar primero sobre el vértice que queremos modificar y después arrastrarlo por la pantalla. Si este vértice está compartido por dos o más objetos, se moverán todos a la vez.

Insercción

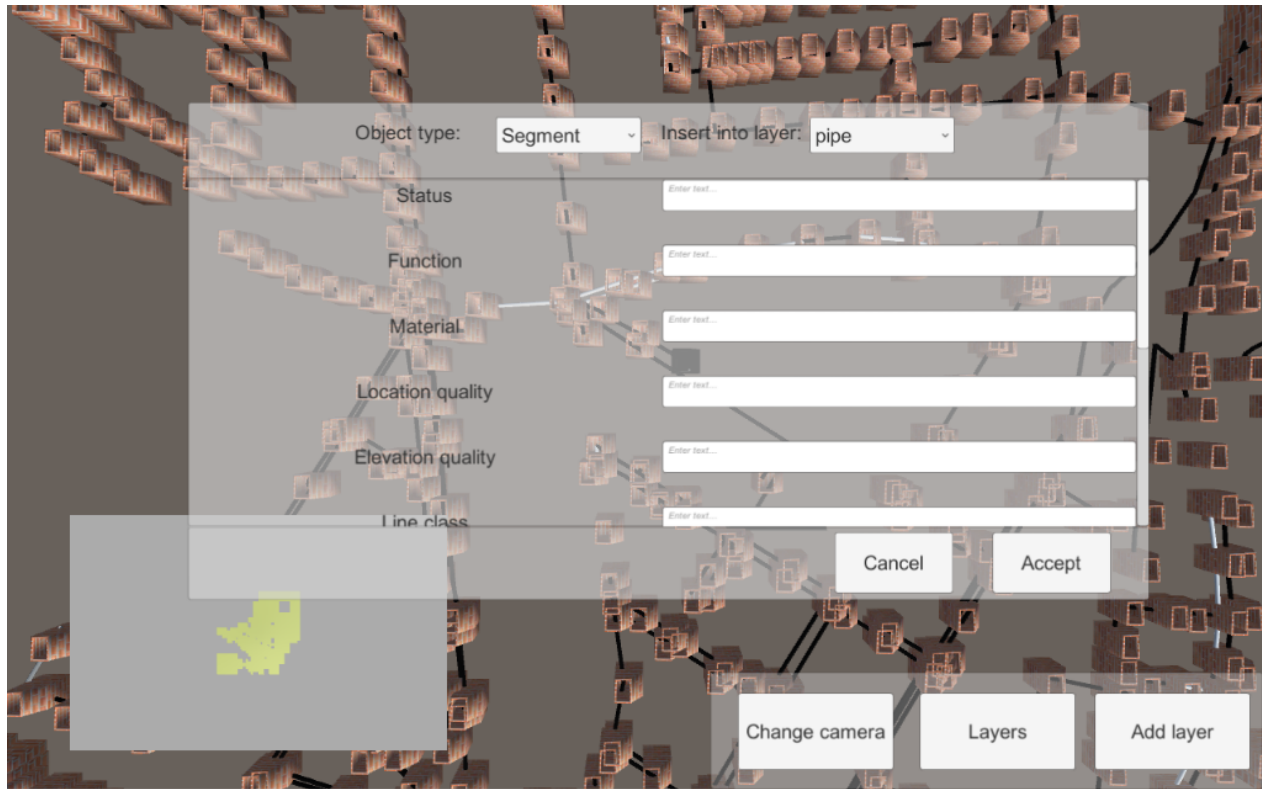


Figura 5.10: Insercción de un objeto.

Para que aparezca el formulario que vemos en la Figura 5.10 debemos hacer un click y mantener presionado sin soltar durante 3 segundos. Existen dos opciones, que no hemos seleccionado nada y por lo tanto el objeto se creará tras rellenar la información o que ese click haya sido encima de otro objeto. Si sucede lo segundo, si el objeto pulsado es una tubería, se podrá crear un nuevo segmento o un objeto puntual. Si es un objeto puntual, solo se podrá crear un nuevo segmento.

En el formulario observamos tres partes:

- **Menú de capas:** El primer control indica el tipo de objeto que queremos crear (segmento o punto) mientras que el segundo sirve para indicar en qué capa se incluirá el nuevo objeto.
- **Menú de información:** Este menú sirve para proporcionar las propiedades temáticas del objeto a crear. Obligatoriamente se deben rellenar todos los campos .

- Menú de botones: Botones de aceptar y cancelar.

Aplicación de realidad aumentada

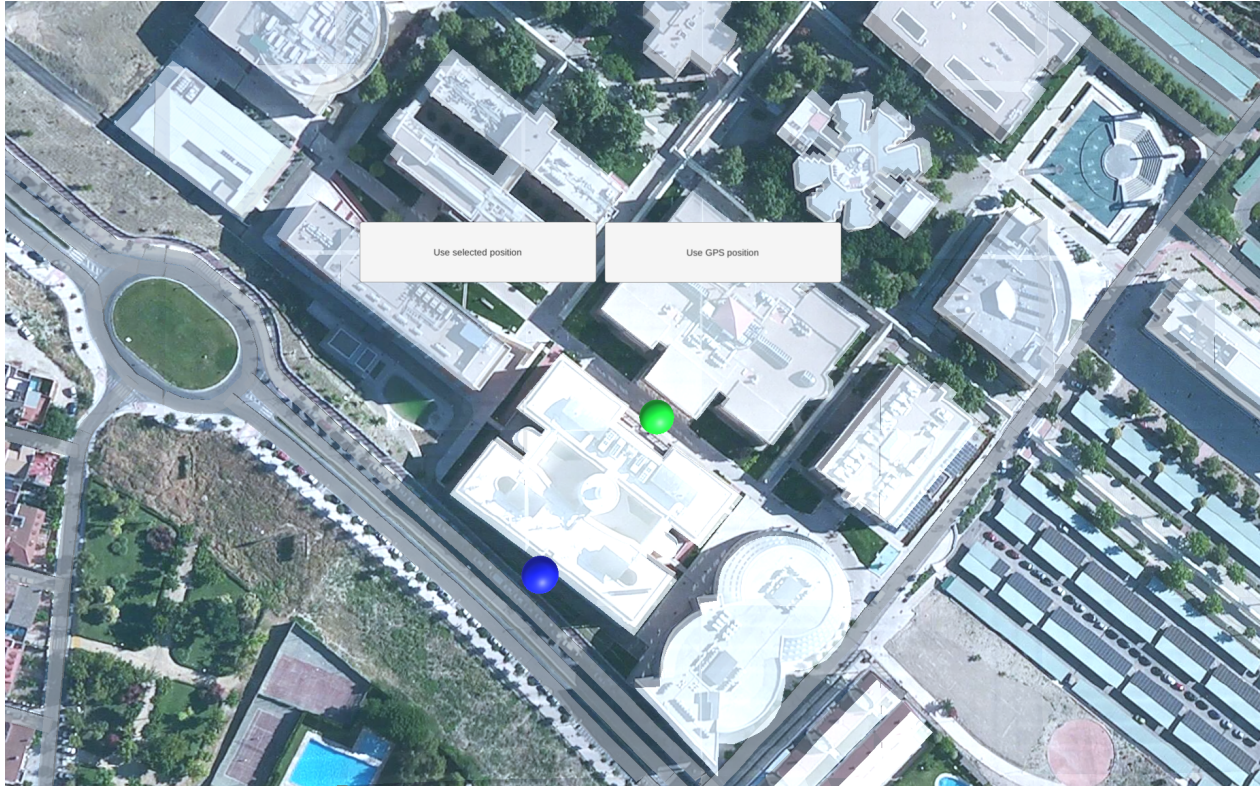


Figura 5.11: Posicionamiento del usuario en la aplicación de realidad aumentada.

Al iniciar el programa nos encontramos con un mapa, en él aparecerán dos bolas. La primera es de color azul e indica la situación del usuario en el mapa según la posición leída por el sensor GPS. La segunda aparecerá cuando el usuario pulse sobre el mapa, esto significa que prefiere seleccionar su posición. Vemos un ejemplo en la Figura 5.11.

Podremos pulsar cualquiera de los dos botones para indicar si queremos posicionarnos por selección o por GPS.

Para la orientación, el usuario deberá acoplar los edificios virtuales y los reales. Una vez que inicie la calibración, el usuario mantendrá la tablet mirando siempre hacia el mismo lugar. Con el desplazamiento de un dedo por la pantalla conseguirá rotar el mapa de los edificios alrededor de él. En este momento los edificios pueden quedar por encima

o por debajo de los reales. Para solucionarlo, el usuario moverá dos dedos hacia arriba o hacia abajo a la misma vez, haciendo que los edificios bajen o suban. Con esto la calibración está realizada y tras pulsar el botón podremos empezar a cargar capas.

Una vez realizado el proceso de calibración, si apuntamos con la tablet hacia el suelo veremos un agujero que representa una excavación. Las tuberías y demás elementos aparecerán dentro de él. La iteración con el software para cargar capas, selección de objetos y demás es la misma que la vista en la Sección 5.1.