UNIVERSIDAD NACIONAL DE LA PLATA

FACULTAD DE INFORMÁTICA

TESINA DE LICENCIATURA

Titulo: Interface visual para definir y manipular representaciones indoor
Autores: Gonzalo Tarántola y Sergio Mendoza Jaufret
Director: Silvia Gordillo
Codirector: Alejandra Beatriz Lliteras
Asesor profesional:
Carrera: Licenciatura en Informática

Resumen

El Sistema de Posicionamiento Global (GPS) se ha universalizado para brindar posiciones en espacios outdoor, lo que facilitó, sumado a la representación de dichos espacios, la búsqueda de caminos outdoor. Por otro lado, el posicionamiento en espacios indoor no es posible realizarlo con GPS lo que ha llevado a diversas propuestas para posicionar y luego representar dichos espacios para brindarle caminos a los usuarios dentro de los mismos.
Actualmente no se encuentran ampliamente explorados los mecanismos para brindar caminos que involucren ambos tipos de espacios (indoor-outdoor) lo que conlleva a la ausencia de una integración transparente con aplicaciones que utilizan sistemas de posicionamiento outdoor (por ejemplo GPS) con espacios indoor al momento de asistir al usuario en la búsqueda de caminos que involucren ambos tipos de espacios.
En este trabajo se presenta una herramienta visual para representar espacios indoor y brindar mecanismos en el mismo para asistir en la búsqueda de caminos dentro de los mismos (llamada "*Editor de espacios indoor*") y otra herramienta visual que permite usar las representaciones de espacios indoor generados por la primera herramienta mencionada e integrar el cálculo de caminos entre posiciones que involucren al espacios outdoor y al espacio indoor (llamada "*Buscador de caminos*").

Palabras Claves

Representación de espacios indoor, búsqueda de caminos indoor, búsqueda de caminos outdoor, integración de búsqueda de caminos indoor-outdoor, posicionamiento, herramienta visual

Conclusiones

En este trabajo se presentó una extensión a un modelo preexistente con el objetivo de brindar al usuario caminos que involucren representaciones de espacios outdoor e indoor de manera integrada. Se prototiparon dos herramientas visuales: una para definir visualmente espacios indoor y otra para la búsqueda y visualización de caminos entre posiciones que pueden estar ubicados indistintamente en espacios outdoor como indoor.

Trabajos Realizados

Se analizo bibliografía actualizada en el área de posicionamiento y representación de espacios indoor y outdoor, se propuso una extensión a un modelo preexistente y se prototiparon dos herramientas visuales: Editor de espacios indoor y Buscador de caminos indoor-outdoor.

Trabajos Futuros

Ambas herramientas propuestas, poseen puntos de extensión para ser abordados en trabajos futuros, por ejemplo:
Editor de espacios indoor: implementación de nuevas formas de movilidades en dichos espacios, abordaje de representación de espacios indoor multiniveles
Buscador de caminos: inclusión de otras estrategias de búsqueda de caminos que involucre a otros proveedores externos, ampliar la búsqueda de caminos para que involucre múltiples representaciones de espacios indoor simultáneamente

Fecha de la presentación: Octubre 2014

Índice general

Capítulo 1

Introducción

En este capítulo se presenta la motivación para realizar el presente trabajo, se exponen los objetivos del mismo y se describen brevemente los capítulos subsiguientes.

1.1. Motivación

Desde hace unos años se ha masificado el uso del Sistema de Posicionamiento Global (GPS) para diversos fines, por ejemplo para navegación (terrestre o marítima), localización y rastreo de personas, animales u objetos. Este Sistema, si bien está universalizado para posicionamiento en espacios outdoor [21] [15], no es apropiado para posicionamiento en espacios indoor [21] [15][5], tanto porque en espacios cerrados puede perderse la señal de satélite, como por la mayor precisión necesaria en ámbitos pequeños. Si bien hay algunos sistemas que proveen posicionamiento en espacios indoor [9] [19] [30], estos no ofrecen una integración transparente con aplicaciones que utilizan sistemas de posicionamiento outdoor (por ejemplo GPS) al momento de asistir en la búsqueda de caminos que involucren ambos tipos de espacios [8]. Tenemos entonces sistemas que funcionan en espacios abiertos, con lo cual es posible encontrar caminos entre dos puntos de interés dentro de una ciudad; y sistemas equivalentes para espacios cerrados, con los cuales podemos orientarnos de una oficina a otra dentro de un mismo edificio. Si los combinamos, podríamos obtener un sistema que nos permita, por ejemplo, orientarnos desde dentro de un edificio hasta la calle, de ahí hasta el acceso a otro edificio distante, y de ahí a alguna ubicación dentro del mismo. Un ejemplo sería poder encontrar un camino desde la

esquina de las calles 1 y 49 de la ciudad de La Plata hasta la Biblioteca de la Facultad de Informática de la UNLP.

Este trabajo se centra en la representación de espacios indoor y en la búsqueda de caminos entre posiciones del espacios indoor y outdoor, logrando una integración correcta y transparente al usuario entre ambas representaciones.

1.2. Objetivos

Realizar dos herramientas visuales prototípicas. Una de ellas proveerá, de manera unificada, la búsqueda de caminos en representaciones indoor y outdoor. La otra herramienta, en tanto, permitirá la creación de espacios indoor sobre los que se realizarán las búsquedas. Se usará el modelo propuesto en [7] con algunas modificaciones y ampliaciones sugeridas para el presente trabajo.

1.3. Organización del documento

En el *Capítulo 2* se presentará la temática de posicionamiento y se describirán distintos sistemas de posicionamiento indoor y outdoor. Se abordará la representación del espacio desde la necesidad de cálculo de caminos presentando diferencias y similitudes entre los espacios indoor y outdoor.

En el *Capítulo 3* se describirá el modelo expuesto en [7] y se presentarán las modificaciones propuestas en este trabajo sobre dicho modelo. Además se explicará el diseño de las dos herramientas visuales que se realizarán en este trabajo.

En el *Capítulo 4* se explicará la manera en que el modelo y las herramientas nombradas anteriormente fueron implementados.

En el *Capítulo 5* se presentan dos ejemplos de uso de las herramientas desarrolladas para mostrar su utilización.

En el *Capítulo 6*, finalmente, se expondrán las conclusiones de este trabajo y se propondrán posibles trabajos futuros sobre las herramientas presentadas.

En el *Apéndice A* se describe de la forma de compilación y ejecución del código fuente provisto como parte de este trabajo.

Capítulo 2

Posicionamiento y representación de espacios

En este capítulo se presentan algunas de las diferentes maneras de brindar posicionamiento y asistencia en la búsqueda de caminos apoyados en la representación del espacio ya sea indoor como outdoor. En [59] se definen los términos espacio indoor y espacio outdoor para referirse a los ambientes o espacios construidos por el hombre para realizar actividades humanas. Mientras que los espacios indoor son aquellos que están "encerrados" dentro de edificios, los outdoor son los que están al aire libre.

A continuación presentamos conceptos relacionados al posicionamiento y a la representación del espacio.

2.1. Posicionamiento

De acuerdo a [12] el posicionamiento consiste en definir la ubicación (o posición), de manera no ambigua, de una persona u objeto en relación a un punto de referencia.

La obtención de una posición es la función espacial mas básica tanto en espacios externos (outdoor) como internos (indoor) [15]

En nuestro trabajo describiremos algunos mecanismos de posicionamiento outdoor e indoor.

2.1.1. Posicionamiento outdoor

Si bien las técnicas más primitivas para el posicionamiento estaban basadas en la observación de las estrellas, en esta Sección nos enfocaremos en métodos más modernos, los cuales pueden dividirse en dos grandes grupos: los que usan señales de radio emitidas por estaciones en tierra, y los que utilizan un sistema de satélites.

2.1.1.1. Posicionamiento mediante señales de radio emitidas por bases terrestres

Se denomina posicionamiento por radio al uso de diversas características de las señales de radio para determinar una posición en la Tierra, ya sea de un objeto remoto o del observador. Las señales de radio son emitidas desde puntos conocidos, comúnmente denominados *beacons* (balizas, en inglés). Para obtener una posición se analiza alguna característica específica de la señal, por ejemplo su frecuencia o el tiempo que tardan en llegar dos emisiones, y a partir de eso se estima la distancia al emisor o la dirección en la que se encuentra. Las diferentes formas de realizar y combinar estas estimaciones dan lugar a diferentes técnicas de *radiolocalización*. Un ejemplo de uso de la localización por radio se da en los teléfonos celulares, que permiten estimar la posición de un aparato usando las señales emitidas por las torres. Esto se logra midiendo el ángulo de llegada de las señales de las distintas torres visibles o bien calculando la diferencia en el tiempo de recepción de la señal de las diversas torres [60].

Los primeros sistemas de navegación por radio usaban algún tipo de antena direccional para determinar la localización de dos base fijas en posiciones conocidas. Se usaba triangulación para trazar sobre un mapa las direcciones en las que se encontraban ambas bases, obteniendo la posición del observador en la intersección de ambas rectas. Más adelante, para simplificar la operación se diseñaron sistemas en los que la antena en tierra rotaba con una frecuencia conocida y enviaba una señal en morse que la identificaba. Para obtener la dirección en la que se encontraba una base, el navegante medía el tiempo entre los picos de intensidad de la señal y realizaba el cálculo de la posición como siempre. En los años 30 se introduce el radar, que permite determinar directamente la distancia a un objeto midiendo el tiempo que tarda en volver parte de la señal de radio reflejada por él.

Los dos primeros sistemas que hicieron uso de técnicas de posicionamiento por radio, fueron los llamados LORAN y Omega, que serán descriptos a continuación.

LORAN (LOng RAnge Navigation) fue un sistema de navegación por radio diseñado para que barcos y aviones pudieran determinar su posición y velocidad usando señales de radio de baja frecuencia emitidas por transmisores fijos sincronizados entre sí [4]. Fue desarrollado a partir de 1941 por los Estados Unidos. Al depender de estaciones en tierra solamente estaban cubiertas algunas regiones, en particular América del Norte, Europa y parte del Pacífico. Para determinar una posición se medía la diferencia en el tiempo de recepción de las señales de los transmisores. Con los datos de un solo par de transmisores se obtenía una curva hiperbólica sobre la que se encontraba la verdadera posición. Para determinar completamente la posición era necesario contar con otro par de transmisores, aunque en la práctica se usaban solamente tres transmisores, uno primario y dos secundarios. Para asegurar la sincronización de las transmisiones se usaba la misma señal: la base primaria la emitía y las secundarias recibían esa transmisión y enviaban su señal. A pesar de que su uso ha declinado mucho desde la masificación de los sistemas de posicionamiento por satélites, todavía quedan algunas estaciones operativas. La Figura 1 muestra cómo funciona el sistema *LORAN*.

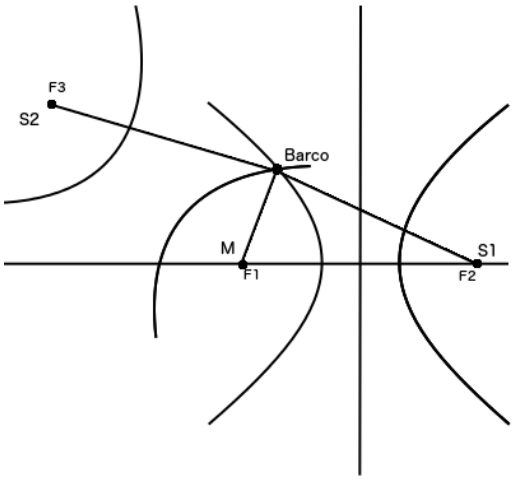


Figura 1: Funcionamiento del sistema LORAN.

En la Figura 1, *M* y *S1* son los focos de una hipérbola, y *M* y *S2* son los focos de una segunda hipérbola. La posición del *Barco* queda definida por el cruce de ambas hipérbolas.

OMEGA fue el primer sistema de navegación por radio con cobertura global [4]. Era operado por Estados Unidos. El sistema comenzó a operar

en 1971 y dejó de funcionar oficialmente el 30 de septiembre de 1997. En la Figura 2 puede apreciarse la distribución de las estaciones emisoras. El mecanismo usado para sincronizar las señales en los sistemas previos a OMEGA funcionaba para distancias moderadas, pero no para cubrir todo el planeta, porque las ondas de radio de baja frecuencia no viajan en línea recta y rebotan en la ionosfera, lo que complica establecer con certeza una señal "primaria". La creación del reloj atómico en los años 1955 permitió que la sincronización se realizara una sola vez: su error es de cerca de 1 segundo cada 30 millones de años, más que suficiente para mantener a los diferentes emisores sincronizados.

Figura 2: Ubicación de las estaciones del sistema Omega.

Como puede verse en la Figura 2, la estaciones emisoras de OMEGA estaban distribuidas por todo el planeta, con el fin de lograr un alcance global.

2.1.1.2. Posicionamiento usando sistemas de satélites

Un *Sistema Global de Navegación Satelital*, conocido por sus siglas en inglés GNSS de *Global Navigation Satellite System*, es un conjunto de satélites con cobertura global y de los que se conoce su posición exacta en cada momento, usados para determinar la posición de un objeto [18].

El primer sistema de navegación que no dependía de estaciones terrestres [58] fue **TRANSIT**. Este sistema contaba con un conjunto de satélites en órbita polar[1] y hacía uso del efecto Doppler para estimar la posición.

[1]Una órbita polar es aquella en la que el satélite pasa sobre los polos de la Tierra o muy cerca de ellos.

Cada satélite viajaba en una ruta conocida emitiendo una señal de una frecuencia determinada. Debido al efecto Doppler, la frecuencia de la señal recibida era diferente a la frecuencia emitida, dependiendo del movimiento del satélite con respecto al receptor. Midiendo esa diferencia a intervalos cortos, el observador podía determinar de qué lado del satélite se encontraba. Tomando mediciones de varios satélites y conociendo sus órbitas, se obtenía la posición del observador. Comenzó a operar en 1964 y era usado originalmente por la Marina de los Estados Unidos. Con el correr del tiempo su uso se amplió a otros sectores, incluyendo civiles. Los satélites dejaron de brindar el servicio en 1996.

En los sistemas satelitales actuales (GPS, GLONASS, IRNSS y Compass), cada uno de los satélites emite continuamente un conjunto de información codificada, entre la que se incluye su posición y la hora exacta de la emisión. De este modo un receptor puede calcular su distancia al satélite usando el tiempo que tardó en recibir el mensaje. La Figura 3 muestra cómo se localiza un punto en la tierra a partir del tiempo que tarda en llegar el mensaje desde cada satélite. Con la información de un satélite se obtiene una esfera, centrada en el satélite, de las posiciones posibles en las que puede estar el receptor (parte A de la Figura 3). Con la información de un segundo satélite se reducen las posiciones a una circunferencia, producto de la intersección de las dos esferas correspondientes (parte B). Intersectando dicha circunferencia con la esfera de un tercer satélite se obtienen dos posiciones posibles (parte C), una de las cuales quedará lejos de la superficie terrestre y puede ser descartada, obteniendo así un único punto.

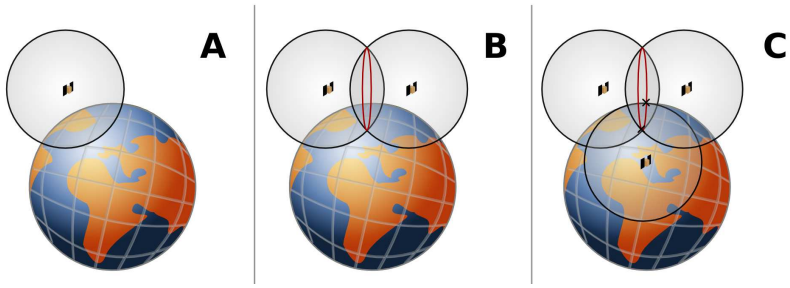


Figura 3: Funcionamiento del GPS

Si bien los satélites usan relojes atómicos para tener una precisión del orden de los nanosegundos, dichos relojes son demasiado costosos para instalarlos en los receptores. Por este motivo, hay un pequeño desfase entre el reloj del receptor y los de los satélites, lo que se soluciona agregando los datos de un satélite extra. De esta manera se tienen cuatro mediciones

para resolver las cuatro incógnitas (latitud, longitud, altura y el desfase horario).

2.1.1.3. Implementaciones

En la actualidad existen varios sistemas de navegación global satelital, con distinto grado de madurez. Los dos sistemas que están operativos en este momento son los llamados *GPS* y *GLONASS*, mientras que *IRNSS* y *Compass* todavía no han terminado de armar la constelación de satélites. A continuación se explican estos sistemas en más detalle.

El **Sistema de Posicionamiento Global**, conocido por sus siglas en inglés GPS de *Global Positioning System*, es operado por el Departamento de Defensa de los Estados Unidos. Los primeros satélites experimentales de lo que hoy es el GPS fueron lanzados a partir de 1978, y se volvieron completamente operacionales en 1995. La constelación actual está formada por 24 satélites. En un primer momento la señal era degradada intencionalmente para el uso civil, con el objetivo de limitar su precisión en hasta 100 metros, pero el 2 de mayo de 2000 esta restricción fue eliminada [61].

El **Sistema de Navegación Global por Satélite**, conocido por sus siglas en ruso GLONASS de *Globalnaya Navigatsionnaya Sputnikovaya Sistema*, es operado por el Ministerio de Defensa de la Federación Rusa. Concebido durante los años 60, el gobierno de la entonces Unión Soviética decidió implementarlo en 1976. Los primeros satélites fueron lanzados en 1982, y el sistema se volvió completamente operativo en 1996. Actualmente consta de 24 satélites activos, y no tiene restricciones de precisión para uso civil. Algunos receptores modernos son capaces de usar GPS y GLONASS al mismo tiempo, lo que mejora considerablemente la precisión.

El sistema **Galileo** está siendo construido por la Unión Europea y la Agencia Espacial Europea. Se empezó a desarrollar en 1999, los primeros satélites fueron lanzados en 2005 y se espera que esté completamente operativo para el año 2019, con una constelación de 27 satélites operacionales. Dispondrá de un servicio gratuito con precisión similar a los otros sistemas GNSS, un servicio pago de mayor precisión y un servicio de emergencia en caso de que los otros servicios se deshabiliten por cuestiones de seguridad.

El sistema de navegación por satélite **Compass** es desarrollado por la Administración Espacial Nacional China. Los primeros satélites fueron

lanzados durante el año 2000, y desde el año 2011 hay 20 satélites brindando cobertura dentro del territorio de la República Popular de China y regiones limítrofes. Se estima que para el año 2020 una constelación de 35 satélites brinde servicio en todo el planeta. Tendrá dos tipos de servicios, uno gratuito con una precisión de unos 10 metros, y uno pago de mayor precisión.

El **Sistema de Navegación Satelital Regional Indio**, conocido por sus siglas en ingles IRNSS de *Indian Regional Navigational Satellite System* está siendo construido por la Agencia India de Investigación Espacial. Los primeros satélites de la constelación fueron lanzados durante los años 2013 y 2014. En el año 2015 deberían estar operativos los 7 satélites que darán cobertura al territorio de la República de la India y sus alrededores.

Limitaciones de los GNSS

El problema con todos los GNSS nombrados anteriormente es que su rendimiento es muy pobre en zonas con muchos edificios altos o con follaje denso y, principalmente, en ambientes cerrados. En espacios urbanos con mucha densidad de edificios, es muy común el fenómeno de propagación multicamino. Se denomina así al fenómeno que hace que una misma señal de radio llegue al receptor desde más de un camino. La Figura 4 ilustra este problema: las líneas completas representan las señales directas y las líneas punteadas representan las señales reflejadas en los edificios.

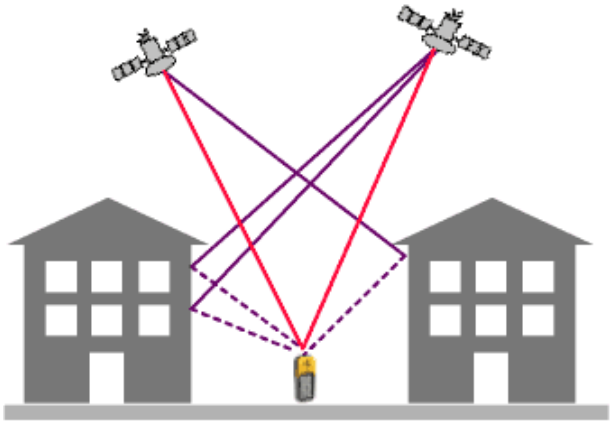


Figura 4: Propagación multicamino

Algunas causas posibles de este efecto son la reflexión en la ionosfera o la refracción y reflexión en cuerpos de agua, montañas o edificios. Entre

los efectos provocados por este fenómenos están la interferencia y el desplazamiento en fase de la señal. La interferencia puede atenuar la potencia de la señal. En el caso de la señal de satélite, la misma señal puede rebotar y llegar al receptor varias veces. Si el tiempo que tardan en llegar las diferentes copias es muy grande, es muy difícil determinar si se trata de la misma señal o no, imposibilitando obtener una posición confiable [5]. La propagación multicamino es un problema para GNSS incluso en ambientes abiertos, porque puede hacer que un receptor estacionario calcule posiciones levemente diferentes en distintos momentos, según cómo vayan llegando las señales.

2.1.2. Posicionamiento indoor

En espacios indoor también es necesario y útil determinar automáticamente la posición de los objetos o individuos, al igual que en espacios outdoor. Sin embargo, no es posible emplear los mismos mecanismos de posicionamiento debido a las limitaciones técnicas de los GNSS en ambientes cerrados y por las diferentes expectativas en el uso del posicionamiento indoor, ya que se requiere una mayor precisión que en los sistemas outdoor. En general los productos disponibles en el mercado no suelen cumplir con las normas internacionales que definen las características de los sistemas de posicionamiento indoor. Esta falta de estandarización en el diseño de los sistemas indoor es parte de la razón por la que su uso no es tan difundido [15].

Uno de los principales problemas de usar señales de satélite para el posicionamiento indoor es la gran atenuación del nivel de la señal, lo que contribuye a empeorar el problema de la *propagación multicamino*: la señal "verdadera" puede tener menor potencia que sus "copias", como explicamos en la Sección 2.1.1.2. En un ambiente indoor típico el receptor procesa una señal sumamente atenuada que llega atravesando paredes, mientras que otras señales no deseadas pueden haber llegado a través de, por ejemplo, una ventana, sufriendo mucha menos atenuación. Ambas señales pueden interferir entre si. En un ambiente abierto suele ser posible determinar cuál es la señal correcta, pero en un ámbito cerrado, la diferencia de potencia entre la señal (correcta) atenuada y el resto de las señales puede ser más grande que lo tolerado por los mecanismos de corrección de errores presentes en las señales. Esto implica que la adquisición y el seguimiento de la señal requiere de un receptor más complejo (que uno outdoor) y no es

posible garantizar resultados óptimos [3]. Es posible (en teoría) diseñar nuevas señales de satélite que puedan ser captadas confiablemente en espacios cerrados, pero todavía no han sido implementadas [3].

Otro punto a tener en cuenta es que, incluso cuando se logra adquirir y mantener una señal en un espacio indoor, es necesario que la precisión del posicionamiento indoor sea muy alta: un error de apenas un par de metros puede significar que la posición calculada está en la habitación de al lado de donde realmente está el receptor.

El mecanismo físico más comúnmente usado como base para calcular la posición es, al igual que en los sistemas outdoor, las ondas de radio [8], aunque también hay sistemas que emplean mecanismos ópticos o sonoros. Estos sistemas usan emisores fijos en posiciones conocidas.

A continuación se detallan algunas técnicas de posicionamiento indoor, clasificadas según la tecnología usada.

Implementaciones

Por los motivos antes mencionados surge la necesidad de utilizar técnicas diferentes para el posicionamiento indoor, entre las que se encuentra las que se basan en GPS, en la identificación de radiofrecuencia, en ultrasonido, en señales de telefonía celular o en Bluetooth.

Dentro de las técnicas basadas en señales de GPS dos ejemplos son los sistemas **Locata** y **Supersense**. El sistema **Locata** utiliza una red de transmisores de radio terrestres cuyas señales se usan para mejorar la precisión de un receptor de GPS o incluso reemplazar la señal de GPS [34]. **Supersense**, en tanto, es una tecnología de gran sensibilidad que permite usar GPS aún cuando su nivel de potencia es muy bajo por ejemplo en el interior de edificios y "cañones urbanos" [2].

La *identificación por radiofrecuencia* (usualmente abreviado como RFID) es un sistema de almacenamiento y recuperación de datos remoto que usa dispositivos denominados *etiquetas*. El propósito fundamental de la tecnología RFID es transmitir la identidad de un objeto (similar a un número de serie único) mediante ondas de radio. Los lectores RFID reciben los datos

[2]Se denomina "cañón urbano" a las zonas de los grandes centros urbanos cuyas calles están flanqueadas por edificios altos, imitando la forma del accidente geográfico cañón

emitidos por las etiquetas RFID mediante una frecuencia de radio definida y un protocolo para transmitir y recibir datos. Las etiquetas pueden ser pasivas o activas. Las etiquetas pasivas operan sin baterías, simplemente reflejan la señal RF transmitida y le agregan información modulando la señal reflejada, sus rangos de lecturas son muy limitados. Las etiquetas activas contienen un transmisor de radio y una batería para alimentarlo, logrando así un mayor rango de lectura. Uno de los sistemas que utilizan identificación por radiofrecuencias es **Landmarc**, el cual utiliza la intensidad de la señal recibida por los lectores RFID para calcular la posición actual y etiquetas de referencia de localización fijas para ayudar a la calibración del sistema [27].

Los sistemas basados en *ultrasonido* cuentan con un conjunto de puntos fijos, que emiten de alguna manera su posición, y se estima la posición de un objeto dado midiendo los tiempos de recepción de una o más señales ultrasónicas sincronizadas. En el caso del sistema **Bat** [6], la zona de interés se cubre con un conjunto de receptores ultrasónicos en posiciones conocidas, conectados a un nodo central. Los objetos móviles de interés cuentan con un código de identificación único y para conocer su ubicación emiten simultáneamente dos señales ultrasónicas en direcciones diferentes, para ser captada por la red de receptores para que el nodo central estime la posición del objeto. En el sistema **Cricket** [31], en cambio, cada objeto se encarga de estimar su posición a partir de los datos recibidos desde emisores fijos en posiciones conocidas. Cada uno de esos puntos fijos emite periódicamente una señal ultrasónica que es captada por los objetos móviles.

Un ejemplo de sistema de posicionamiento usando *señales de telefonía celular* es el sistema **CellSense** [20], que funciona para teléfonos GSM. El proceso está conformado por dos fases: una primer fase de contrucción de fingerprints[3] offline y una segunda fase de seguimiento online, dónde se utilizan las fingerprints para calcular la ubicación más probable.

A partir de la versión 1.1 de la especificación de Bluetooth (2002), es posible detectar la intensidad de la señal recibida. De este modo es posible aplicar los mismos principios descriptos para otras tecnologías en dispositivos que implementen el protocolo Bluetooth. En [26] se describe un método de posicionamiento basado en las intensidad de señal Bluetooth entre un dispositivo móvil y algunas bases estacionarias.

[3]Las fingerprints almacenan información acerca de la intensidad de recepción de la señal desde diferentes estaciones base en diferentes ubicaciones del área de interés.

2.2. Representación del espacio

La representación del espacio se realiza desde hace cientos de años [29] y de acuerdo a [33] la representación de los espacios indoor difieren de la representación de los espacios outdoor, al menos, desde lo estructural (un espacio indoor posee menos discontinuidades que uno outdoor) y desde lo funcional (los espacios indoor suelen tener un uso más estacionario que los outdoor).

Dado que el abordaje de la representación del espacio en el presente trabajo se realiza como apoyatura para brindarle caminos a los usuarios, se toma lo expresado en [15] donde se menciona que para este fin, el objetivo es similar tanto en un espacio outdoor como en uno indoor: llegar desde un origen hacia un destino y que los algoritmos de búsqueda de caminos, asumen la existencia de la representación del espacio tanto indoor como outdoor . Sin embargo, las diferencias del entorno hacen que la tarea de navegación sea más demandante en un espacio indoor. Una de estas diferencias es la disponibilidad de hitos de navegación y sus características. En el caso outdoor, estos hitos suelen ser objetos persistentes bien visibles (un lago, un edificio) mientras que en un espacio indoor los hitos son más pequeños y cambiantes (un cuadro, un número de habitación). En el espacio outdoor, los hitos brindan un sistema de referencia que permite al usuario ubicar el espacio local por el que se está transitando en el espacio global. Al contrario, en un espacio indoor la visibilidad es mucho más reducida por la misma estructura de las construcciones, lo que hace que el usuario dependa más de hitos locales. Es más fácil aprender una ruta particular dentro de un edificio que obtener una visión global de la configuración de los espacios dentro del mismo.

Entre las diferencias más grandes entre los espacios indoor y outdoor se encuentra las restricciones del propio espacio.

2.2.1. Representación del espacio indoor

De acuerdo a [23] un espacio indoor está determinada por las características de sus componentes arquitectónicos, tales como puertas, corredores, pisos, paredes, escaleras. Para analizar un espacio indoor, es necesario describir y entender esos componentes. Existen varios modelos de datos en las comunidades de arquitectura e ingeniería para hacer esto, sin embargo el foco de esos modelos no está en la información espacial.

Según [57], los modelos de espacios indoor pueden clasificarse en diferentes categorías, siendo las dos grandes divisiones los modelos semánticos y los modelos espaciales.

Los modelos semánticos representan los tipos de entidades contenidos dentro de un espacio indoor, así como sus propiedades y relaciones. Los modelos espaciales se puede categorizar a su vez en modelos topológicos, modelos geométricos y modelos híbridos. Los modelos topológicos tienen en cuenta las propiedades de conectividad del espacio y se enfocan en dos aspectos: el *espacio topográfico* (los componentes estructurales del espacio tales como habitaciones, pasillos y puertas) y el *espacio de caminos* entre los componentes del *espacio topográfico*. El *espacio topográfico* puede modelarse tanto en 3D como con un conjunto de capas 2D. El *espacio de caminos* se modela como un grafo cuyos nodos son regiones y la presencia de un arco implica que las regiones están conectadas de alguna manera.

Los modelos puramente geométricos suelen ser usados para el diseño y administración de edificios. Una estrategia de modelado es la *representación de fronteras*, que describe la topografía usando una representación geométrica (vértices, arcos, etc) de los límites del objeto [57]. Dentro de este tipo de modelos hay distintas variaciones y extensiones, para almacenar información extra además de la geometría. Esta estrategia de modelado también es usada para espacios outdoor.

Los modelos de espacios indoor más útiles son híbridos, combinando datos topológicos y geométricos [57]. Por ejemplo, es importante la definición de habitaciones o áreas y las relaciones entre ellas, además de otras propiedades tales como el uso que se le da a esas áreas, la cantidad de personas que las habitan, etc. Por otro lado, la geometría le agrega información sobre distancias y fronteras a la conectividad y accesibilidad provista por la topología.

Los espacios indoor no existen aislados, sino que forman parte del espacio global. Un objetivo importante es proveer al usuario una experiencia consistente y transparente al moverse entre espacios indoor y outdoor.

2.2.2. Representación del espacio outdoor

Como representación del espacio outdoor en este trabajo se toma la representación geográfica del espacio, la que refiere a la superficie de la

Tierra y será presentado de acuerdo a lo expuesto en [25]. La representación geográfica data de largo tiempo, destacándose como inicio relevante el siglo XV.

Una de las problemáticas con las que se trabaja en este tipo de representación, es la complejidad del mundo geográfico y la necesidad de hacer una representación finita del espacio.

Dependiendo del nivel de detalle de la información a representar y de su naturaleza, existen dos maneras fundamentales de representar el espacio geográfico, mediante objetos discretos o bien mediante campos continuos.

Mientras que los objetos discretos representan objetos del mundo real con limites bien definidos en su representación, los campos continuos representan objetos del mundo real como un conjunto finito de variables.

Dos métodos posibles para representar datos geográficos son el Raster y el Vector. En la representación Raster, se divide al mundo en celdas y se le asigna atributos a cada una de ellas, mientras que en la representación Vector, se emplean los elementos: líneas, puntos y polígonos para representar el espacio geográfico.

Para poder implementar el cálculo de caminos en este tipo de representación, es necesario contar con un modelo de red de circulación. Una red de Circulación es un grafo compuesto por puntos y líneas [11] que representa las calles, las intersecciones entre las mismas, puentes, y en algunos casos los sentidos de circulación, entre otra información.

La representación geográfica y el modelo de red de circulación, conllevan un proceso exhaustivo de análisis, representación, almacenamiento y mantenimiento de estructuras, lo que lo hace costoso y por ello se suele recurrir a reusar representaciones existentes cuando se incluyen en un proyecto.

2.3. Conclusiones

En este capítulo se presentaron diferentes técnicas de posicionamiento y se presento la representación del espacio desde la perspectiva de su uso para brindarle caminos a un usuario.

Capítulo 3

Modelo propuesto

En este capítulo se presenta, por un lado, una herramienta para definir visualmente espacios indoor, contemplando puntos de acceso y conexiones entre ellos, y por el otro una herramienta visual para lograr una integración transparente al usuario entre las representaciones indoor y outdoor al realizar búsqueda de caminos que involucren ambas representaciones. Para lograr este objetivo, el trabajo propuesto se basa en el uso de un modelo para representar el espacio indoor y su conexión con el espacio outdoor. Partimos del modelo planteado en [7] y lo extendemos con lo necesario para incluir la definición gráfica de los espacios indoor, la visualización de los caminos encontrados y la integración entre las representaciones indoor y outdoor.

3.1. El modelo existente

El modelo existente está planteado en [7] y tiene los siguientes objetivos:

- Buscar y brindar caminos a usuarios. Estos caminos pueden pertenecer sólo a representaciones de espacios indoor, o incluir tramos en espacios outdoor.
- Recalcular el camino en caso de que el usuario no siga el recorrido planteado originalmente.
- Determinar el área (dentro de una representación) en la que se encuentra el usuario en un momento dado.

- Permitir la representación de áreas de acceso restringido según el tipo de usuario.

El modelo original presentado en el trabajo citado está dividido en dos capas con responsabilidades bien diferenciadas: la capa *Geográfica* y la capa *Movilidad*. La capa *Geográfica* se encarga de representar objetos con información espacial, mientras que la capa *Movilidad* le agrega semántica a los datos representados y le aporta funcionalidad (e.g.: obtener caminos). Se debe destacar que el modelo adaptado contempla sólo objetos posicionados y esta línea de trabajo se continúa en nuestra propuesta. A continuación se explican cada una de las capas, las cuales serán presentadas sólo en términos de las clases relevantes para nuestra propuesta de trabajo, sin embargo en las Figuras se visualizarán todas las clases.

3.1.1. Capa Geográfica: Puntos y Áreas de Interés

En la capa Geográfica se modelan objetos posicionados del mundo real (e.g. una fuente en una plaza) con su ubicación espacial, representados mediante las clases *PuntoInteres* y *Ubicacion*, que pueden verse en la Figura 5.

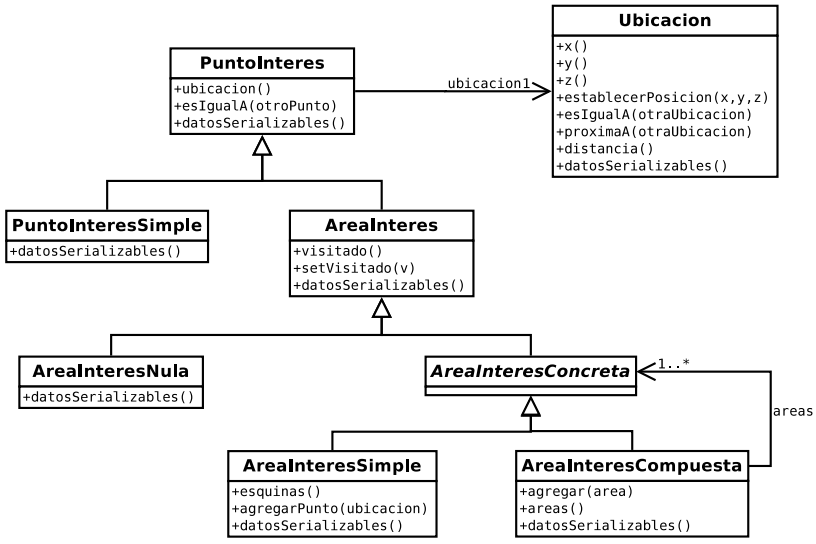


Figura 5: Diagrama de clases de la capa *Geográfica*

Representar como clases diferentes a *PuntoInteres* y *Ubicacion* permite abstraer la manera en que dicha ubicación o posición se representa y permite tener un comportamiento específico de la posición, como por ejemplo distancia [22].

La clase *PuntoInteres* tiene varias subclases, de acuerdo a la forma en que se necesite representar a un punto de interés. Para representar puntos de interés de los que no interesa su superficie en un dominio particular (e.g. una escultura en un museo o la ventanilla de la boletería de un teatro) se utiliza la clase *PuntoInteresSimple*. Para los objetos de interés de los que es necesario conocer su superficie (e.g. un aula en una escuela o un lago en un Parque Nacional), se usa la clase *AreaInteres*. Esta clase sigue el patrón *NullObject* [13], por lo que tiene dos subclases: *AreaInteresNula*, que se utiliza en los casos en que se necesita representar que aún no se tiene información sobre el área o que un área no existe, y *AreaInteresConcreta*. La clase *AreaInteresConcreta* sigue el patrón *Composite* descripto en [14], por lo que tiene dos subclases: *AreaInteresSimple*, para definir áreas de interés simples, y *AreaInteresCompuesta*, que permite definir áreas de interés que a su vez contengan otras áreas de interés más específicas.

La decisión de representar un objeto de interés como una instancia de *PuntoInteresSimple* o de *AreaInteres* depende del dominio en particular. Por ejemplo, si se está modelando la ubicación de los hospitales dentro de una ciudad, cada hospital será una instancia de *PuntoInteresSimple*. Sin embargo, si se necesita tener información sobre la ubicación de las habitaciones, consultorios y quirófanos dentro de un hospital particular, será necesario representarlo como una instancia de *AreaInteresCompuesta* con tantas instancias de *AreasInteresSimple* como habitaciones, consultorios y quirófanos tenga dicho hospital.

3.1.2. Capa Movilidad

La movilidad a la que hace referencia el nombre de esta capa es la posibilidad de asistir al usuario, encontrándole caminos hacia objetos de interés determinados, brindándole información acorde al contexto y manteniendo un historial de los lugares por donde pasó. Para una mayor fluidez en la presentación de esta capa, se la introduce mediante cuatro submodelos, organizados de acuerdo a su representatividad:

1. Movilidad propiamente dicha: a lo largo de este trabajo se define a la movilidad como el conjunto de segmentos conectados entre sí para ser usados en la búsqueda de caminos.

2. Conectividad entre movilidades: este submodelo provee la posibilidad de recorrer una misma área de interés con diferentes movilidades.

3. Caminos: conjunto de segmentos sugeridos o recorridos.

4. Modelo de usuario: contempla la representación del usuario propiamente dicho, sus preferencias, las zonas a las que tiene acceso y las que no, y su ubicación actual. Este submodelo no será abordado en este trabajo.

A continuación se describirán las clases pertenecientes a cada uno de los submodelos que serán utilizados en este trabajo.

3.1.2.1. Movilidad

En concordancia con lo antes expuesto, del submodelo Movilidad sólo explicaremos las clases relevantes para lo propuesto en la Sección 3.2. En la Figura 6 puede verse el diagrama de clases de este submodelo.

El submodelo Movilidad está formado por la clase *Movilidad* y sus subclases. La clase *Movilidad* tiene las siguientes subclases: *TransitarAreasInteres*, que se usa para buscar caminos entre áreas de interés de una representación indoor, y *RedCirculacionInterna*, que se utiliza para buscar caminos en una red de circulación de una representación indoor. Cada instancia de una subclase concreta de *Movilidad* colaborará con un conjunto de instancias de alguna subclase de *UnidadDesplazamiento*, las cuales serán las aristas del grafo que representa los posibles caminos dentro de cada movilidad. De esta manera, las instancias de la clase *TransitarAreasInteres* colaboran con instancias de la clase *ConectorAreaInteres*, y las de *RedCirculacionInterna* con *SegmentoCirculacionInterna*. Los nodos de dichos grafos serán instancias de las clases *AreaInteres* y *NodoCirculacionInterna*, respectivamente. Todas estas clases implementan el protocolo Visitable, formado por los métodos *visitado* (que indica si el nodo ya fue visitado por el algoritmo de búsqueda) y *setVisitado* (que le da un valor a esa propiedad), para facilitar el desarrollo de los algoritmos de búsqueda.

Si bien las clases *SegmentoRedCalles*, *NodoRedCalles*, *Calle* y *RedCalles* pertenecen al modelo original, no serán utilizadas en el trabajo propuesto, ya que, cómo se verá en la Sección 4.2.2 de este Capítulo, la búsqueda de caminos outdoor será responsabilidad de una entidad externa que provee, además, la representación de este tipo de espacios. Otro conjunto de clases que no será utilizado en este trabajo es el formado por *AreaLogica* (y sus subclases), *TransitarAreasLogicas* y *ConectarAreasLogicas*. En el modelo

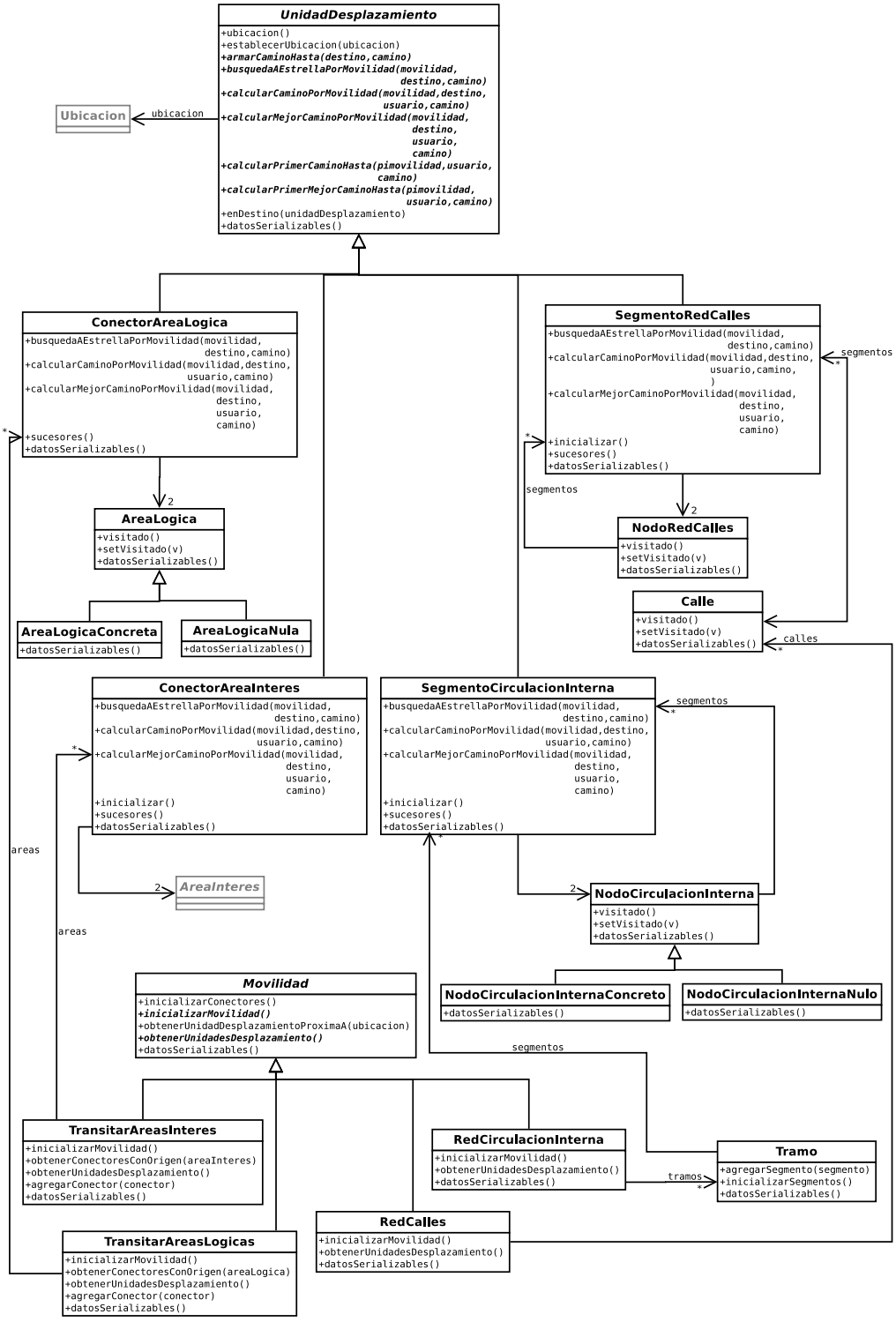


Figura 6: Diagrama de clases del submodelo “Movilidad”

original, las áreas lógicas son de creación volátil y varían más dinámicamente que las áreas de interés, por lo que no serán contempladas en este trabajo.

Las clases que aparecen grisadas no forman parte de esta subsección y serán explicadas en la subsección correspondiente, pero se incluyen por existir una relación de conocimiento hacia ellas. Esta misma metodología se mantendrá en el resto del Capítulo.

3.1.2.2. Conectividad entre movilidades

Este submodelo está organizado contemplando las clases que permiten conectar movilidades entre sí. La Figura 7 muestra su diagrama de clases.

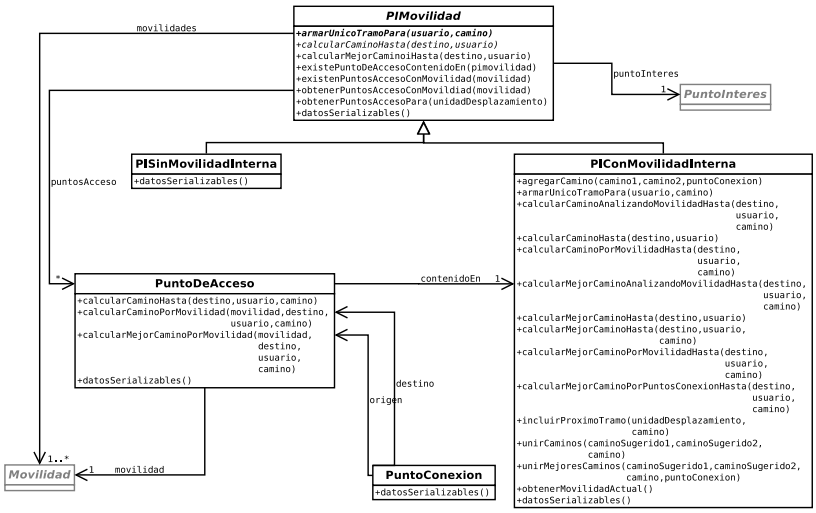


Figura 7: Diagrama de clases del submodelo
"Conectividad entre movilidades"

La jerarquía de clases *PIMovilidad* agrega las características de movilidad a los objetos de la capa geográfica. Las subclases que componen esta jerarquía definen el comportamiento que indica como deben ser recorridos los objetos de las diferentes subclases de *PuntoInteres*, constituyendo así un decorador [14] de dichas clases. De este modo, la clase *PISinMovilidadInterna* es un decorador de la clase *PuntoInteresSimple* y *PIConMovilidadInterna* es un decorador de la clase *AreaInteres*. Al momento de instanciar cada subclase de *PIMovilidad* se restringe la subclase de *PuntoInteres* a la que puede conocer. Esta abstracción posibilita que un mismo *PuntoInteres* pueda ser recorrido de diferentes formas ya que una instancia de *PIMovilidad* puede colaborar con varias instancias de la clase *Movilidad* (representado en el modelo de la Figura 7 mediante la relación llamada *movilidades*).

La clase *PuntoDeAcceso* representa la forma de acceder a un *PIMovilidad*, relacionándolo con una *UnidadDesplazamiento* y una *Movilidad*. Para

poder representar conexiones entre dos *PIMovilidad*, es necesario relacionar el *PuntoDeAcceso* que representa la salida de una *PIMovilidad* con el *PuntoDeAcceso* que representa la entrada a la otra. Esta relación está dada por la clase *PuntoConexion*.

3.1.2.3. Caminos

El último submodelo a describir para la capa de *Movilidad* es el denominado "Caminos". Mediante las clases de este submodelo es posible representar los caminos que se le sugieren a los usuarios, así como aquellos caminos que efectivamente realizaron. Las clases que lo conforman puede verse en la Figura 8.

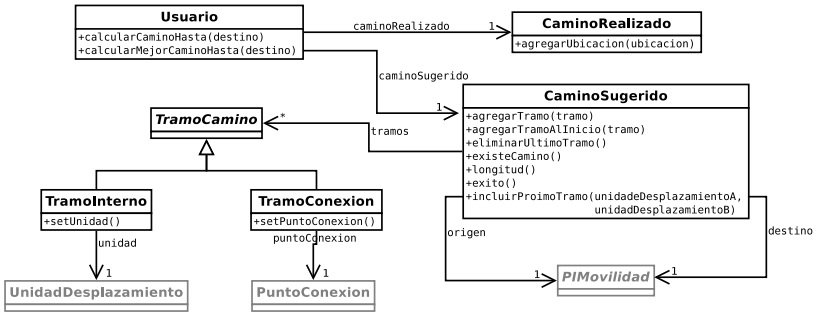


Figura 8: Diagrama de clases del submodelo "Caminos"

La clase *CaminoSugerido* representa el camino que se le ha calculado a un usuario para llegar a un destino determinado. Como puede verse en la Figura 8, el camino está formado por instancias de la clase *TramoInterno* (dentro de un mismo *PIMovilidad*) o de *TramoConexion* (cuando se pasa de un *PIMovilidad* a otro).

La clase *CaminoRealizado* es un punto de extensión expresado en el modelo original, el cual no es abarcado por el presente trabajo y representa el camino realizado por un usuario. Esto permitiría, entre otra cosas, validar si un usuario salió del camino que recibió y en ese caso recalcularlo en base a su ubicación actual.

3.2. Nuestra propuesta

La propuesta de este trabajo implica ampliar el modelo original para poder crear dos herramientas visuales, una para editar espacios indoor

("Editor de espacios indoor") y otra para buscar caminos indoor-outdoor ("Buscador de caminos indoor-outdoor"). En la Figura 9 se puede ver la forma en que estas herramientas colaboran, tanto con el modelo original como entre ellas, así como otras posibles herramientas y formas de comunicación que podrían plantearse de este modelo.

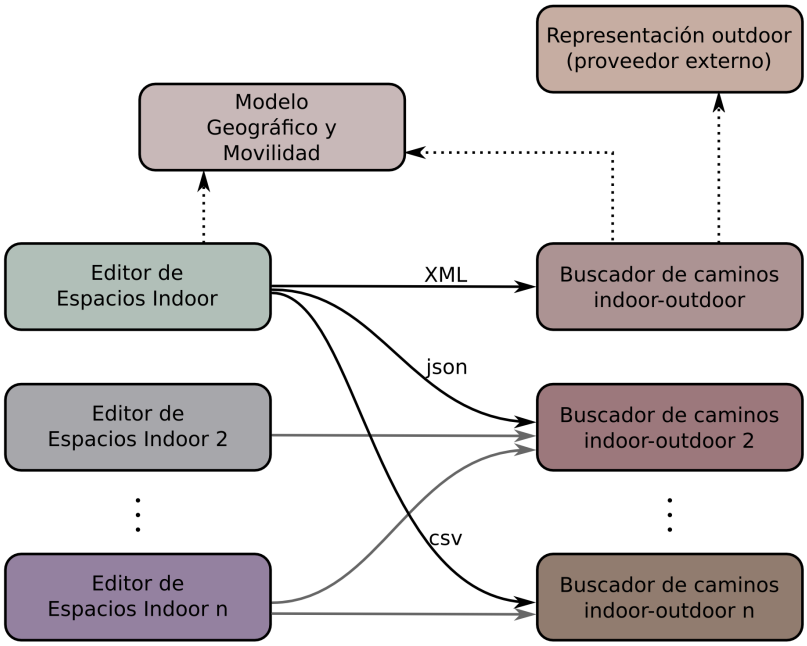


Figura 9: Vista esquemática de las herramientas propuestas y su relación con el modelo existente.

La Figura 9 muestra que la herramienta *Editor de espacios indoor* se relaciona con el modelo de base (modelos *Geográfico* y *Movilidad*) y que se podrían implementar diferentes editores. En particular en nuestro trabajo implementamos uno, al que denominamos simplemente **Editor**. Lo generado por un editor puede ser "consumido" por diferentes *Buscadores de caminos indoor-outdoor*, donde cada uno de ellos puede relacionarse o no con proveedores externos de representaciones outdoor. En este trabajo, el *Buscador de caminos indoor-outdoor* implementado (llamado **Buscador de Caminos**), además de relacionarse con un proveedor externo para realizar la búsqueda de caminos, utiliza el mismo modelo de base utilizado por el **Editor**.

En las secciones siguientes se detallarán las ampliaciones realizadas al modelo de base y los modelos particulares de cada herramienta.

3.2.1. Extensión al modelo existente

La extensión propuesta para el modelo existente consiste en dos herramientas visuales que hacen uso de lo definido por las capas *Geográfica* y *Movilidad* para proveer interfaces gráficas tanto para la edición de espacios indoor como para la búsqueda de caminos indoor-outdoor. En este trabajo definimos a un espacio indoor como aquel que involucra una o más áreas de interés y que puede tener en su interior puntos de interés. Por ejemplo, un espacio indoor puede ser la biblioteca de la Facultad de Informática, con un área simple que representa el total de la superficie y puntos de interés como el mostrador de atención a alumnos y cada una de las PCs disponibles. Este ejemplo se verá paso a paso, usando las herramientas visuales propuestas, en el Capítulo 5.

El concepto de espacio indoor está modelado en este trabajo por la clase *EspacioIndoor*, que consta principalmente de relaciones a todas las clases que representan el "contenido" de un espacio indoor, tal como puede verse en la Figura 10.

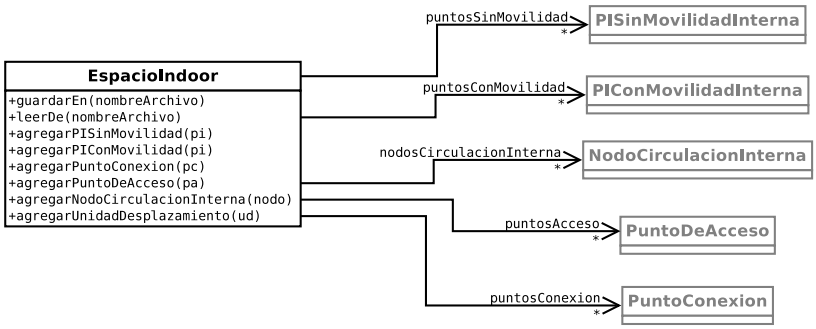


Figura 10: Diagrama de clases de EspacioIndoor.

Veremos en el Capítulo 4 la implementación de esta clase y su comportamiento específico.

3.2.2. Herramientas propuestas

Para el presente trabajo se desarrollaron dos herramientas que interactúan entre si para la búsqueda de caminos: el **Editor**, para definir gráficamente espacios indoor, y el **Buscador de Caminos**, para realizar la búsqueda de caminos indoor-outdoor y visualizar dicha búsqueda. El *espacio indoor*, compuesto por áreas y puntos de interés y las conexiones entre ellos son definidos en el **Editor** y pueden ser guardados por este como

una instancia de la clase *EspacioIndoor*, convertida a una representación adecuada, en un archivo usado para la interacción entre las dos herramientas. El **Buscador de Caminos** implementa la lectura del tipo de archivo generado por el **Editor** para obtener una definición de espacios indoor, y realiza la búsqueda del camino entre dos Puntos de Interés, de acuerdo a alguna estrategia implementada, y lo muestra sobre un mapa.

A continuación, a modo de introducción se explicará la forma de uso de cada herramienta y posteriormente se describirá el modelo de clases subyacente para cada una de ellas.

3.2.2.1. Editor

Esta herramienta es la que permite al usuario definir gráficamente la forma en la que el espacio indoor está dividido en áreas y puntos de interés, paso necesario antes de poder buscar caminos indoor en dicho espacio. Visualmente, la herramienta tiene capas de información para los distintos tipos de objetos que se pueden definir. En la Figura 11 se pueden ver esas capas desplegadas, las mismas han sido generadas con la herramienta implementada y superpuestas gráficamente con fines explicativos.

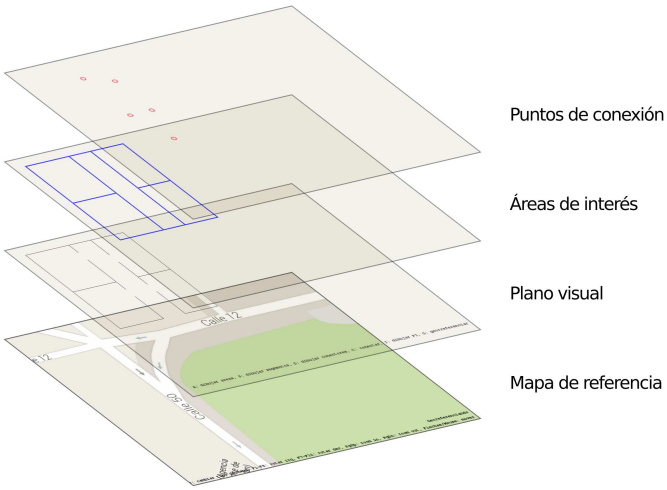


Figura 11: Superposición de capas visuales de información

Las dos capas inferiores, *Mapa de referencia* y *Plano visual*, no son editables sino que se usan para ayudar al usuario a definir los espacios indoor. La capa *Mapa de referencia* muestra un mapa que el usuario puede usar para georreferenciar (en caso de que exista) lo que está dibujando. Esta capa está visible solamente mientras se está ubicando geográficamente el

dibujo, durante la edición del mismo deja de verse. La capa *Plano visual* está visible de manera opcional, mostrando un plano, provisto por el usuario, que indique la arquitectura del espacio a representar, de modo que sirva de guía durante la edición.

Las dos capas superiores que se observan en la Figura 11 son las editables por el usuario. En la capa *Áreas de interés* se dibujan las áreas de interés definidas por el usuario. Sobre esta capa se ubica la capa *Puntos de conexión*, que es la capa donde se dibujan los conectores entre áreas y los puntos de acceso.

En el modelo propuesto para esta herramienta, las capas de información están modeladas por la clase *Capa*. Esta clase tiene como objetivo encapsular el mecanismo específico utilizado para dibujar en pantalla, exponiendo solamente los métodos para dibujar los elementos gráficos requeridos por la herramienta.

Todas las operaciones que se pueden realizar en el **Editor** están modeladas mediante la jerarquía de clases con raíz en *Herramienta*, que puede verse en la Figura 12. Esta clase es abstracta y sólo define el protocolo de las herramientas: tres métodos para manejar el movimiento del mouse (inicio, puntos intermedios y fin del movimiento), y uno para hacer efectiva la edición. Todas las herramientas conocen a un objeto de la clase *Lapiz*, que no tiene comportamiento propio sino que solamente contiene los parámetros necesarios en caso de que la herramienta tenga que mostrar algo en pantalla.

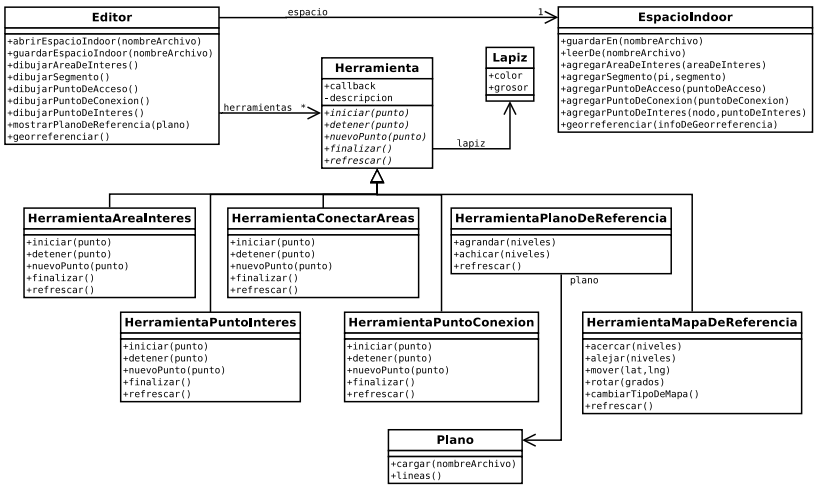


Figura 12: Jerarquía de clases de las herramientas del Editor

Se presentan tres funciones fundamentales sustentadas por el diagrama de la Figura 12:

- Asistencia al usuario durante la edición: definida por las clases *HerramientaPlanoDeReferencia* y *HerramientaMapaDeReferencia*.
- Definición de áreas y puntos de interés: definida por las clases *HerramientaAreaInteres* y *HerramientaPuntoInteres*.
- Conexión de áreas de interés entre si y con el exterior: definida por las clases *HerramientaPuntoConexion* y *HerramientaConectarAreas*.

A continuación se explican con más detalle cada una de las clases de este modelo.

HerramientaAreaInteres: Es la utilizada para dibujar un área de interés. Para ello el usuario define la forma del área indicando sus vértices. Cuando termina de dibujar, esta herramienta convierte la secuencia de puntos en una instancia de la clase *AreaInteresSimple*, que es agregada a la colección de áreas de interés pertenecientes al espacio indoor actual.

HerramientaPuntoInteres: Es la herramienta utilizada para dibujar un punto de interés. Funciona de manera similar a la herramienta anterior, pero en este caso el usuario solamente debe definir la posición dentro de espacio indoor que se está editando donde se encuentra el objeto de interés. Al igual que con las área de interés, una vez que se termina de definir el punto, esta herramienta convierte la posición en una instancia de *PuntoInteresSimple* que será agregada al espacio indoor actual.

HerramientaPlanoDeReferencia: Es la herramienta utilizada para mostrar un plano como fondo durante la edición del espacio indoor. El plano es provisto por el usuario indicando un archivo externo (no generado por el **Editor**) y contiene una representación del diseño del lugar sobre el que se está definiendo las áreas de interés que forman el espacio indoor.

HerramientaMapaDeReferencia: Se utiliza esta herramienta para georreferenciar el espacio indoor que se está creando. Para ello se muestra un mapa al que se puede rotar, trasladar, agrandar y achicar hasta que coincida con el espacio que se está editando.

HerramientaPuntoConexion: Esta herramienta permite al usuario definir la ubicación de los puntos de conexión entre áreas de interés.

HerramientaConectarAreas: Esta herramienta se usa para conectar efectivamente las áreas de interés utilizando los puntos de conexión definidos previamente. Para ello el usuario debe elegir el punto de conexión y las áreas de interés que vincula, luego de lo cual la herramienta crea una instancia de *PuntoConexion* y la agrega al espacio indoor que se está editando.

3.2.2.2. Buscador de Caminos

Esta es la herramienta que realiza la búsqueda de caminos entre dos Puntos de Interés e integra los tramos indoor con los tramos outdoor del resultado de la búsqueda, en caso de existir estos últimos. Este resultado se muestra sobre un mapa de la zona que contiene a los puntos de interés origen y destino. El **Buscador** debe leer un archivo creado por el **Editor** con la definición de un espacio indoor para poder listar los posibles puntos de interés que se pueden elegir.

El **Buscador de caminos** obtiene los tramos indoor del camino usando la funcionalidad provista por la capa *Movilidad*, mientras que los tramos outdoor se obtienen por una herramienta externa, por ejemplo Google Maps u Open Street Maps. Se siguió un diseño similar al planteado en [24]. La búsqueda de caminos y la integración de los tramos indoor y outdoor es responsabilidad de la clase *BuscadorIntegrado*. Como se observa en la Figura 13, para obtener los tramos indoor y outdoor del camino, la clase *BuscadorIntegrado* colabora con otras dos clases: *BuscadorIndoor* y *BuscadorOutdoor* y cada una de ellas tiene un grupo de subclases, siguiendo el patrón Strategy [14]. Por un lado están *BuscadorIndoor1* y *BuscadorIndoo2* como ejemplos de implementaciones concretas de buscadores indoor, y por otro están *BuscadorGoogleMaps* y *BuscadorOpenStreetMaps* como implementaciones concretas de buscadores outdoor que utilizan proveedores externos.

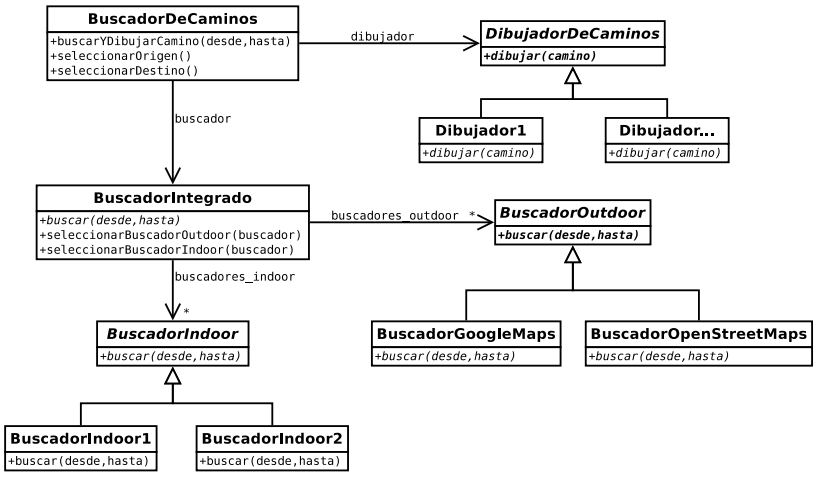


Figura 13: Diagrama de clases del **Buscador de Caminos**

Con el modelo propuesto es posible buscar caminos entre una posición del mapa (representación outdoor) y un punto de interés (dentro de una

representación indoor). Para esto, una instancia de la clase *BuscadorIntegrado* determina el punto de acceso del área de interés que contiene el punto de interés elegido más cercano a la posición dada y delega en una de sus instancias de *BuscadorOutdoor* la búsqueda del camino hasta allí. A continuación, una de las instancias de *BuscadorIndoor* completa el camino entre dicho punto de acceso y el punto de interés deseado. Al tener varias estrategias de búsquedas indoor y outdoor disponibles se obtienen varias ventajas: en el caso de los buscadores indoor, es posible elegir uno que minimice alguna métrica particular (por ejemplo, distancia recorrida). En el caso de los buscadores outdoor, está la posibilidad de tratar de obtener el camino con un buscador externo en particular y si no está disponible, intentar con otro.

Al momento de hacer la conexión entre los tramos indoor y outdoor es necesario transformar las coordenadas de los tramos internos para mostrar el camino completo adecuadamente sobre el mapa. Esta transformación es responsabilidad de la clase *BuscadorIntegrado*, que utiliza los datos que están almacenados en el archivo que contiene el espacio indoor leído por el **Buscador de caminos**. Como se mencionó en la sección 4.2.1, esta información es generada por el usuario desde el **Editor**.

Una vez calculado el camino, la clase *DibujadorDeCaminos* de la Figura 13 se encarga de dibujar los diferentes tramos sobre un mapa. Por ejemplo, se puede usar la API de Google Maps para dibujar los caminos outdoor e indoor. En este caso, tendríamos una subclase de *DibujadorDeCaminos* llamada *DibujadorGoogleMaps* que se encargaría de dibujar ambas partes del camino obtenido. Podrían existir tantas estrategias de dibujo como estrategias de búsqueda. Cabe aclarar que es necesario asegurar una cierta coherencia al momento de instanciar los distintos componentes del **Buscador** ya que una instancia de *DibujadorGoogleMaps* podría no ser capaz de dibujar correctamente los resultados de una búsqueda realizada mediante una instancia de *BuscadorOpenStreetMaps*.

El punto de acceso a la herramienta **Buscador de Caminos** es la clase *Buscador*, que provee métodos para conocer los puntos de interés disponibles y para buscar un camino hasta uno de ellos desde una coordenada geográfica dada o bien desde otro punto de interés del espacio indoor. Una instancia de *Buscador* conoce a una instancia de *BuscadorIntegrado* para realizar las búsquedas y a la instancia de *EspacioIndoor* que representa el espacio indoor sobre el que se está trabajando actualmente.

3.3. Conclusiones

En este capítulo se presentó una extensión al modelo propuesto en [7] con el comportamiento necesario para la definición visual de espacios indoor y para lograr una integración transparente entre la búsqueda de caminos indoor y outdoor. Se presentó el diseño de las dos herramientas implementadas para poner en práctica el funcionamiento de las extensiones realizadas al modelo.

Capítulo 4

Implementación propuesta

El modelo adoptado como base para el presente trabajo ha sido implementado previamente en [10]. Dicha implementación está enfocada mayormente en la búsqueda de caminos, y utiliza la representación de espacios outdoor tal como está en el modelo original presentado en [7], es decir, mediante clases del modelo original. Dicha implementación, escrita en Smalltalk [54], no cuenta con una interfaz gráfica para su uso, ya sea para definir los espacios indoor como para mostrar los caminos encontrados. En ella, los espacios se definen en el workspace del entorno, escribiendo sentencias Smalltalk para crear los objetos del modelo presentado en [7] y el resultado de la búsqueda de un camino es una instancia de la clase *CaminoSugerido* (del modelo), y hay que inspeccionarlo con la herramienta del entorno para conocer los tramos a recorrer.

El trabajo propuesto en esta tesis presenta un enriquecimiento del modelo original presentado en [7] y una implementación distinta a la antes mencionada, brindando en este caso una interface gráfica para la definición de los espacios indoor y para visualizar el resultado de la búsqueda de caminos. En este trabajo se prescindirá de la representación de espacios y recorridos outdoor presentes en [7], el objetivo será integrar la búsqueda de caminos indoor definida en la capa *Movilidad* del modelo con la búsqueda de caminos outdoor de un proveedor externo (como por ejemplo Google Maps). Otra característica de la implementación propuesta será el almacenamiento en disco del espacio indoor definido por el usuario.

La implementación realizada en este trabajo consiste en dos librerías y dos herramientas visuales. En la librería denominada `libgis` se implementaron las capas *Geográfica* y *Movilidad* descriptas en el Capítulo 3,

así como un mecanismo de persistencia de los datos. La otra librería contiene funcionalidad en común a las dos herramientas y fue implementada bajo el nombre de `libgispy`. Las herramientas son el **Editor** y el **Buscador de Caminos**, que ya fueron introducidos en el Capítulo 3 y hacen uso de la funcionalidad provista por ambas librerías. Al tener separadas las librerías de las herramientas, es posible reutilizarlas independientemente, por ejemplo para testear el funcionamiento de `libgis` sin tener en cuenta la interface visual.

4.1. Librerías

Las dos librerías implementadas para el presente trabajo (`libgis` y `libgispy`) fueron escritas en el lenguaje C++ [41]. La librería `libgis` se especializa en la representación del espacio indoor, su persistencia y en la búsqueda de caminos, mientras que la librería `libgispy` está orientada a dar soporte a las herramientas visuales, las cuales fueron desarrolladas en el lenguaje Python [52].

En las secciones siguientes se detallarán los aspectos más importantes de cada una de las librerías y herramientas.

4.1.1. Librería libgis

A diferencia de la implementación presentada en [10] que estaba escrita en Smalltalk, para este trabajo se realizó una implementación en C++, lo que nos permite compilar la lógica de las capas *Geográfica* y *Movilidad* en una librería que puede ser utilizada por diferentes herramientas implementadas en distintos lenguajes.

A causa del lenguaje de implementación usado en este trabajo, el manejo de memoria debe administrarse de una manera explícita. En la Sección 4.1.1.1 planteamos la problemática y la decisión de implementación adoptada.

Otra necesidad derivada del lenguaje de implementación elegido fue la persistencia explícita de los datos, la cual es abordada en la Sección 4.1.1.2.

4.1.1.1. Manejo de memoria

Uno de los puntos a tener en cuenta al programar en C/C++ es el manejo de la memoria. A diferencia de otros lenguajes como Java y Smalltalk, que poseen un garbage collector para encargarse de liberar automáticamente la memoria de los objetos que ya no son usados, el ambiente de ejecución estándar de C/C++ no provee el manejo automático de memoria, sino que el desarrollador debe definir explícitamente un mecanismo para tal fin, ya sea liberando manualmente los objetos o empleando alguna librería existente.

La dificultad en definir este mecanismo se da porque en general no es trivial asegurarse de que todos los objetos alocados en la heap sean eventualmente desalocados. En C++ el enfoque habitual para manejar el tiempo de vida de los objetos alocados dinámicamente es determinar quién es el dueño de cada objeto (ya que no siempre es el mismo objeto que lo crea). El dueño es el responsable de eliminarlo luego de asegurarse de que nadie más vaya a usar ese objeto. Sin embargo hay casos en donde no es posible elegir un dueño, por ejemplo cuando un mismo objeto puede estar en varias colecciones en un momento dado, y los tiempos de vida de dichas colecciones no coinciden. Esto sucede en la implementación de la librería `libgis` con las instancias de las clases *PIMovilidad* y *PuntoDeAcceso* por ejemplo.

En el lenguaje usado, para evitar que el programador tenga que decidir quién es el dueño de cada objeto, es habitual el uso de *punteros compartidos* (shared pointers), un caso particular de *puntero inteligente* (smart pointer), concepto introducido en [2]. Un puntero inteligente es un objeto que simula ser un puntero común, pero agregando funcionalidad adicional. En el caso de un puntero compartido lo que se agrega es el conocimiento de cuántos objetos están usándolo en un momento dado. Cuando el objeto apuntado no está siendo usado por ningún otro objeto, el puntero compartido lo destruye y se libera su memoria. De esta forma se evita que objetos sin referenciar vivan indefinidamente.

Debe tenerse en cuenta que aún es posible tener referencias ilegales. La implementación más común de punteros compartidos (y la que se usó en este trabajo) tiene un contador de referencias para determinar cuándo es seguro destruir a un objeto, pero si hay referencias cíclicas este método no es efectivo. En estos casos deben utilizarse *punteros débiles* (weak pointers), que se comportan como un puntero compartido pero que no fuerzan a mantener el objeto en memoria.

En este trabajo se utiliza la implementación de punteros inteligentes provista por la librería boost [38].

4.1.1.2. Persistencia

A causa del lenguaje de implementación elegido, fue necesario realizar la persistencia explícita de los datos, a diferencia de la implementación preexistente [10], realizada en Smalltalk, donde alcanza con guardar una imagen del sistema para persistir el estado del mismo [16]. En cambio, en la implementación en C++ es necesario programar explícitamente cómo deben persistirse los objetos, por lo cual se implementó un módulo de persistencia para que todas las clases tengan un método que les permita guardar y leer su estado.

Al momento de elegir un mecanismo de persistencia las dos opciones que se tuvieron en cuenta fueron usar una base de datos espacial[1] o escribir en un archivo.

Usando una base de datos espacial se tiene acceso a un conjunto de operaciones para realizar consultas con criterios "espaciales", además de los operadores relacionales tradicionales, así como el manejo de transacciones y el acceso concurrente a los datos [17]. Sin embargo, nuestro modelo ya tiene planteadas las operaciones de búsqueda y habría que realizar varias modificaciones para reemplazarlas por consultas a una base de datos. Por otro lado, las herramientas implementadas en este trabajo no requieren acceso concurrente a los datos ni soporte para operaciones transaccionales, motivo por el cual se puede prescindir de una base de datos espacial como podría ser PostgreSQL [48] con el módulo PostGIS [47].

Para llevar a cabo la persistencia, se decidió utilizar la funcionalidad de serialización de objetos provista por la librería boost.serialization [39]. En este contexto se entiende a la serialización como la conversión de un conjunto de objetos a una secuencia de bytes, que puede ser usada a su vez para reconstruir dichos objetos. Previo a la elección de la librería mencionada, se analizaron algunas alternativas: MFC (Microsoft Foundation Classes), GNU CommonC++ y s11n, o implementar un mecanismo de serialización ad hoc. Las características de las cuatro librerías son similares, a excepción

[1] Una Base de Datos Espacial es una base de datos pensada para el almacenamiento y consulta de datos que representan objetos definidos en el espacio.

de MFC, que solamente compila en Windows. Se eligió a boost.serialization por pertenecer a la familia de librerías boost [37].

Las características más destacadas de la librería boost.serialization [39] son:

- Almacenamiento y recuperación de los datos referenciados con punteros. Esta librería permite persistir objetos que contengan referencias a otros objetos alocados dinámicamente (i.e.: en la heap).
- Portabilidad, tanto del código como de los archivos generados. La librería asegura que los archivos generados en un sistema operativo puedan ser leídos correctamente en otro, sin necesidad de que el programador tenga que codificar las conversiones que pudieran ser necesarias.
- Existencia de diversos formatos de archivos. Hay dos formatos de salida predefinidos por la librería boost.serialization tiene (XML y binario) y se pueden definir los que sean necesarios.
- Ausencia de restricciones en los tipos de datos que se pueden persistir. Se puede persistir todos los tipos de datos primitivos (int, bool, arreglos, etc) así como las clases y estructuras definidas por el usuario o librerías externas, independientemente de su complejidad.
- Versionado de objetos. En caso de que se modifique la definición de una clase (agregando o quitando variables de instancia), esta librería asegura que los archivos con contenido generado previamente podrán ser leído sin problemas. Esto lo logra almacenando un número de versión en cada objeto persistido, que se incrementa antes cambios en la definición de una clase.
- No ser intrusiva en el modelo existente. Esta propiedad permite persistir tanto clases propias como clases sobre las que el programador no tiene control, por ejemplo aquellas que pertenezcan a librerías de terceros.

La librería boost.serialization permite definir la información que se va a persistir de cada clase de dos maneras diferentes. Si el programador tiene control sobre la clase a persistir, alcanza con implementar un método en esa clase para definir los datos que deben guardarse para poder luego recuperar el estado del objeto. En caso de que la clase esté fuera del control del

programador (por ejemplo, en una librería de un tercero), existe la opción de definir una función global con una signatura específica para realizar la misma tarea que el método interno. Como teníamos control sobre todas las clases que necesitábamos persistir (en nuestro rol de programadores), en este trabajo se optó por la primera opción.

Al momento de elegir la representación usada en el archivo hay que tener en cuenta que dicha representación determina el espacio en disco que ocupará y la facilidad de lectura por una persona. En general, las representaciones más legibles por un ser humano son formatos textuales como por ejemplo XML o JSON, pero ocupan más espacio en disco que una representación binario de los mismos datos. En [32] se dice que la única justificación para usar una codificación binaria es si el volumen de datos es muy grande y debe aprovecharse al máximo el espacio disponible o si debe asegurarse que el tiempo y el esfuerzo para que el programa interprete los datos sea el mínimo. Para el resto de los casos lo recomendable es usar una representación textual.

La librería boost.serialization provee dos tipos de representaciones preestablecidas para el archivo de salida: formato binario o formato XML, y (como ya se mencionó) existe la posibilidad de definir formatos propios. Para este trabajo elegimos la representación XML predeterminada, lo que nos facilita la lectura de los archivos para corroborar que el contenido sea el esperado.

La clase *EspacioIndoor*, introducida en el Capítulo 3 como representación del espacio indoor definido por el usuario, es la responsable de realizar la persistencia de dicho espacio. Esta clase mantiene colecciones de instancias de las clases que forman un espacio indoor: *PISinMovilidadInterna*, *PIConMovilidadInterna*, *NodoCirculacionInterna*, *PuntoDeAcceso* y *PuntoConexion*. Al momento de persistir una instancia de *EspacioIndoor* se le piden mediante su protocolo los "datos serializables" necesarios para poder luego reconstruir su estado. Para hacer esto, en la instancia de *EspacioIndoor* se recorre cada una de las colecciones nombradas anteriormente recolectando los datos serializables de cada elemento de las colecciones. Cada elemento, a su vez, obtiene sus datos serializables recorriendo recursivamente sus propias colecciones internas.

El mecanismo para inicializar una instancia de *EspacioIndoor* a partir de los datos de un archivo es similar. Se van consumiendo los datos serializables contenidos en el archivo y se van generando las instancias que forman

las colecciones propias de *EspacioIndoor* y recursivamente el contenido de cada una de ellas.

El mismo problema de referencias cíclicas que enfrentamos en el manejo de la memoria (descripto en 4.1.1.1), lo tuvimos al momento de determinar la información que se va a persistir en las instancias de cada clase. En este caso, la persistencia de objetos referenciados como *punteros débiles* no está a cargo de las instancias de las clases a las cuales pertenecen, sino que es la clase *EspacioIndoor* la responsable de recopilarlos y persistirlos adecuadamente.

4.1.2. Librería libgispy

Esta librería cumple dos funciones: brindar acceso desde Python a los objetos C++ del modelo y proveer a las herramientas visuales la posibilidad de leer archivos DXF.

Las herramientas visuales presentadas en este trabajo fueron implementadas en Python, aprovechando la flexibilidad de este lenguaje, su completa librería estándar y su sistema de tipos dinámico, lo que en conjunto permite escribir rápidamente prototipos que se pueden ir refinando progresivamente [1]. Para que estas herramientas puedan manipular los objetos del modelo (implementado en C++) es necesario establecer una interface de comunicación entre los modelos definidos en ambos lenguajes. Esta interface de comunicación se realizó mediante un binding entre C++ y Python, el cual se describe en la sección 4.1.2.1.

Uno de los objetivos de las herramientas visuales a desarrollar es facilitar al usuario la definición de los puntos y áreas de interés. Una manera de lograrlo es mostrando un plano sobre el que el usuario tenga una guía al dibujar los distintos elementos. En la Sección se describirá el mecanismo usado para leer el plano desde un archivo en formato DXF.

4.1.2.1. Binding C++ → Python

Python provee nativamente una API en C para escribir módulos de extensión [50]. Esta API es adecuada si el módulo de extensión se escribe en C, en caso de escribirlo en C++ hay que realizar un trabajo adicional. En cualquiera de los dos casos, la funcionalidad expuesta por esta API es de

bajo nivel, el programador del módulo de extensión debe asegurarse de no invalidar el estado del intérprete de Python durante la ejecución del código del módulo de extensión.

Existen varias herramientas y librerías para simplificar la tarea de binding entre C++ y Python, haciendo más seguro el manejo de la memoria y del tiempo de vida de los objetos que "viajan" de un lenguaje a otro. Las herramientas más populares son SWIG, SIP y Boost.python. Para este trabajo elegimos Boost.python, a continuación describiremos las características de las tres y presentaremos el análisis realizado sobre cada una.

SWIG [55] es una herramienta open source para conectar una herramienta escrita en C o C++ con otra escrita en uno de los muchos lenguajes soportados, Python entre ellos. Para exportar una clase escrita en C++ a Python son necesarios tres pasos. En primer lugar se debe escribir un conjunto de directivas (en un lenguaje propio de SWIG) en un archivo indicando qué funcionalidad de la clase se va a exportar. A continuación debe ejecutarse la herramienta SWIG con ese archivo como entrada, generándose un archivo cpp[2] con todo el código necesario para exportar la clase. Finalmente debe compilarse ese archivo (y todos los que se hayan creado si se exporta más de una clase) para obtener el módulo de extensión de Python.

Por otro lado, *SIP* [53] es una herramienta que funciona de manera similar a SWIG, pero está diseñada para conectar C++ únicamente con Python. Al igual que SWIG, es necesario definir archivos especificando las clases a exportar, que son tomados por la herramienta para generar archivos cpp. Estos archivos finalmente se compilan para obtener el módulo de extensión de Python. A diferencia de SWIG, SIP provee un módulo Python con funcionalidad de soporte para el módulo de extensión generado.

Como se ha visto, ambas herramientas requieren de tres pasos para generar el módulo de extensión Python: la definición de las clases a exportar en archivos de especificación, el procesamiento de un grupo de archivos de especificación y finalmente la compilación de los archivos cpp creados en el paso previo. En cambio, Boost.python [40] tiene un enfoque diferente para la forma de especificar las clases a exportar, lo que permite omitir el paso de procesar archivos de especificación para generar archivos cpp. Aprovechando las características de metaprogramación de C++ y su preprocesador,

[2]Llamaremos "archivo cpp" a un archivo con extensión ".cpp", que contiene código fuente escrito en C/C++ y que es procesado por el compilador para generar código objeto.

Boost.Python logra que la especificación se haga usando una sintáxis simple, directamente en código C++. Esto permite compilar directamente el módulo Python, sin pasos intermedios.

La especificación de las clases a exportar se realiza en uno o más archivos cpp. Para cada clase exportada se indica qué funciones serán visibles en Python. La librería se encarga de determinar el tipo de retorno (de existir) y de los parámetros (de existir); estos serán comprobados en ejecución para evitar invocaciones inválidas. Dado que el tipado en Python es dinámico, el chequeo de tipos solo puede hacerse en ejecución.

Para mostrar la forma en que se exporta una clase, tomaremos como ejemplo a la clase *Plano* del modelo propuesto (Figura 12).

```
class_<Plano>("Plano")
  .def("cargarDXF", &Plano::cargar)
  .def("lineas", &Plano::lineas,
      return_value_policy<reference_existing_object>()
  );
```

Este código especifica que la clase *Plano* implementada en C++ será visible en Python como una clase con el mismo nombre. A continuación se definen los dos métodos que se exportan: *cargar* y *lineas*, indicando en primer lugar el nombre con el que se conocerá al método en Python y en segundo lugar el método de la clase C++ que se invocará. No es necesario indicar nada acerca de los parámetros. En la especificación del método *lineas*, puede verse el uso de un tercer parámetro, que indica la forma en que se manejará el valor de retorno de la función. En este caso se especifica *reference_existing_object*, lo que implica que se está retornando una referencia a un objeto ya existente y cuyo tiempo de vida está manejado por la clase *Plano* de C++. Las otras dos opciones son: *return_by_value*, en los casos en que se quiere retornar una copia o *manage_new_object*, para indicar que se retorna una referencia a un objeto nuevo, pero cuyo tiempo de vida debe ser manejado del lado de Python.

4.1.2.2. Lectura de archivos DXF

El formato DXF se usa como formato de intercambio entre diferentes programas de diseño asistido por computadora (CAD) [35]. La idea es que

si uno cuenta con un plano en este formato, puede usarlo como base para definir las áreas de interés, los puntos de interés y las distintas movilidades "dibujando" sobre el mismo.

Para leer los archivos DXF se utiliza la librería open source llamada *dime* [42]. Esta librería permite leer un archivo DXF y representar la información contenida en estructuras de datos propias. Dichas estructuras de datos pueden ser escritas como un archivo DXF. En este trabajo, la clase *Plano*, introducida en la Sección 3.2.2.1, usa la funcionalidad de la librería *dime* para leer un archivo DXF y a continuación convierte la información obtenida en las líneas que serán usadas para dibujar el plano.

4.2. Herramientas visuales

A partir de la funcionalidad ofrecida por las librerías de base descriptas en las Sección 4.1 se implementaron dos herramientas visuales en Python: el **Editor**, implementado como una aplicación de escritorio que sirve para definir los espacios indoor con sus puntos y áreas de interés, entre otros elementos del modelo; y el **Buscador de caminos**, implementado como una aplicación web, para visualizar el camino encontrado entre dos puntos de interés.

A continuación detallamos cada una de estas herramientas.

4.2.1. Editor

El objetivo de esta herramienta es permitir que el usuario cree los objetos de las capas *Geográfica* y *Movilidad* y las relaciones entre ellos.

El **Editor** es una aplicación de escritorio implementada en Python que usa la funcionalidad provista por `libgispy` para manejar la interacción con `libgis`, que a su vez provee lo necesario para la definición de los espacios indoor y para la persistencia de objetos. Además hace uso de la API de Google Maps para el georreferenciamiento de las áreas definidas.

Como se explicó en el Capítulo 3, la información presentada por esta herramienta está dividida en capas visuales, representadas como instancias de la clase *Capa*. Esta clase encapsula el mecanismo específico utilizado

para dibujar en pantalla, que en este trabajo se realiza mediante la librería de Python llamada *pygame* [49]. Ésta es una librería diseñada para escribir video juegos y aplicaciones multimedia, haciendo énfasis en facilitar la manipulación de gráficos y sonido pero sin perder eficiencia. Esto lo logra delegando las operaciones más complejas a otra librería, escrita en C, llamada *SDL* (Simple DirectMedia Layer) [51]. Ambas librerías son multiplataforma y con licencias LGPL. La clase *Capa* expone solamente la funcionalidad de *pygame* requerida por el **Editor** para armar los elementos de su interface (rectángulos, círculos, elipses, líneas y mapas de bits) y para el manejo de eventos de teclado y mouse.

Durante la edición es posible tener un plano como fondo para facilitar la tarea de dibujar los elementos que forman el espacio indoor, tal como se mencionó en el Capítulo 3. En esta implementación en particular, los planos deben estar en formato DXF. La clase *Plano* se encarga de la lectura de este tipo de archivos y de convertirlos en una representación adecuada para que una instancia de la clase *HerramientaPlanoDeReferencia* los pueda dibujar en pantalla.

En la Figura 14 puede verse a la instancia del **Editor** en proceso de edición, con un plano de fondo leído desde un archivo DXF (en color negro) y dos áreas de interés (una ya creada en color azul y una en proceso de creación en color verde). La parte inferior derecha de la pantalla provee

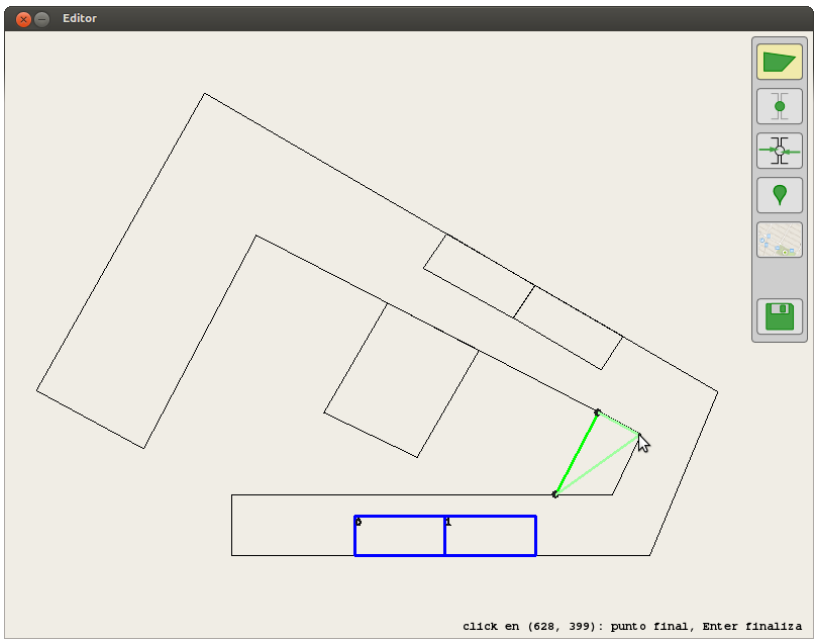


Figura 14: **Editor** en proceso de edición de áreas de interés.

información relevante para el usuario según la operación que esté realizando. En este caso, muestra las coordenadas donde se encuentra el cursor del mouse y la acción que se ejecutará al hacer click (definir un nuevo punto del área) o presionar Enter (finalizar la creación del área).

A medida que el usuario define los elementos que conforman un espacio indoor, se les asigna la ubicación correspondiente al espacio cartesiano visible en el **Editor**. Pero para poder hacer el nexo entre los tramos indoor de los caminos y los tramos outdoor, es necesario saber a qué coordenadas geográficas se corresponden las coordenadas cartesianas de los puntos de acceso. Para definir ese mapeo existe la clase *HerramientaMapaDeReferencia*, que permite mostrar un mapa como fondo del editor al que se lo puede rotar, trasladar y hacer zoom hasta hacer coincidir la representación del espacio indoor con el mundo exterior. La transformación de las coordenadas cartesianas del espacio indoor en coordenadas geográficas va a depender de la proyección que utilice el proveedor del mapa de referencia. En nuestra implementación este proveedor es Google Maps, el cual usa en sus mapas el sistema de coordenadas WGS 84 [28] con una variante de la proyección Mercator [56] [44]. En este caso, la API de Google Maps nos provee tanto de los mapas de referencia como de las funciones de transformación de coordenadas.

En la Figura 15 se aprecia el **Editor** mostrando el plano de la planta baja de la Facultad de Informática sobre el mapa de la zona correspondiente provisto por Google Maps.

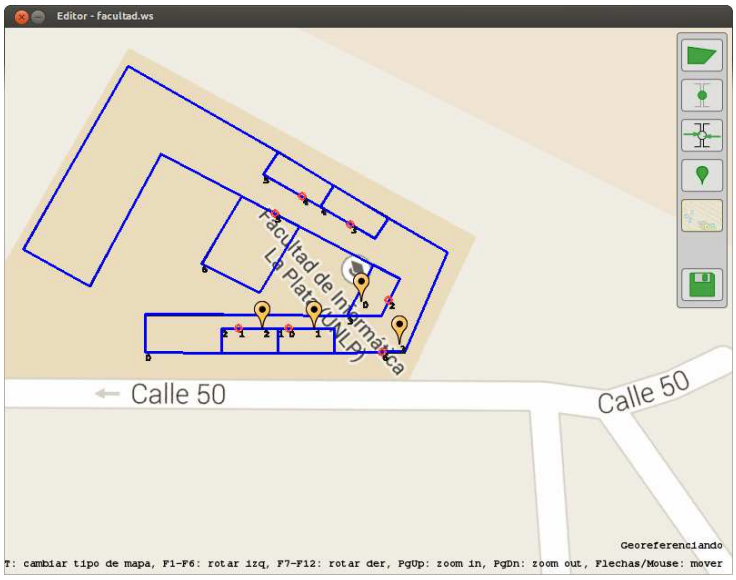


Figura 15: Georreferenciando un espacio indoor

Los datos de georreferenciamiento obtenidos en este proceso son persistidos junto con el resto de los objetos del espacio indoor para poder ser mostrados correctamente sobre un mapa de Google posteriormente. Los datos guardados son: el grado de rotación que se aplicó al mapa original para hacerlo coincidir con el espacio indoor y los niveles de zoom correspondientes (uno de mayor granularidad provisto por Google Maps y uno de menor granularidad propio de nuestras herramientas). De no contar con estos datos, no será posible dibujar el espacio indoor sobre un mapa y, por lo tanto, tampoco será posible realizar la búsqueda de caminos indoor-outdoor.

La finalidad de esta herramienta es generar un archivo con el espacio indoor definido por el usuario. Este archivo podrá ser usado en otro momento por el mismo **Editor** para continuar la edición del espacio indoor, o por el **Buscador de Caminos** para obtener la información necesaria acerca de los espacios indoor, tal como se describirá en la sección siguiente.

El **Editor** actualmente solo maneja una movilidad *Transitar Áreas Interés* para todo el espacio indoor que se está definidiendo. Esta movilidad se va actualizando automáticamente a medida que el usuario define los diferentes puntos de conexión. y conecta las áreas de interés.

4.2.2. Buscador de Caminos

El **Buscador de Caminos** es una herramienta escrita en Python que combina la funcionalidad de búsqueda de caminos indoor implementada en la librería `libgis` con la búsqueda de caminos outdoor provista por la librería JavaScript [45] de Google Maps [43], obteniendo caminos que pueden atravesar indistintamente espacios outdoor e indoor.

Esta herramienta fue implementada como una aplicación web para que sea fácilmente accesible desde un dispositivo que cuente con un navegador web con JavaScript activado. Además, al tener la interface como una página HTML, podemos utilizar la librería JavaScript provista por Google Maps para interactuar con sus mapas y caminos. Esta librería permite, entre otras cosas, dibujar caminos definidos manualmente por el programador, lo que nos posibilita usar el mismo código JavaScript para dibujar los tramos indoor y outdoor de los caminos encontrados por nuestra herramienta.

En la Sección 4.2.2.1 explicamos los detalles de implementación del servidor web que proporciona las funcionalidades de la aplicación, y el modo de uso de la misma en la Sección 4.2.2.2.

4.2.2.1. Servidor web

Como se decidió que la interface de la herramienta fuera web, creamos un servidor web utilizando la funcionalidad provista por la librería estándar de Python. La decisión de escribir nuestro propio servidor se tomó por la poca complejidad de los requerimientos que maneja este herramienta, lo que no justifica usar un servidor web existente, como Apache [36] o lighhtpd [46].

La librería estándar de Python cuenta con varios módulos relacionados con diferentes aspectos de la comunicación vía HTTP. Para el caso más básico de atender peticiones HTTP y generar la respuesta que será mostrada por un browser, se tiene el módulo *BaseHTTPServer*. Dentro de este módulo están las dos clases que nos interesan: *BaseHTTPRequestHandler* y *HTTPServer*. Las instancias de *HTTPServer* se encargan de crear sockets HTTP y escuchar hasta que llegue un requerimiento para despacharlos a un manejador. Nuestro manejador de requerimientos es una subclase de *BaseHTTPRequestHandler* llamada *BuscadorRequestHandler*.

Para iniciar la ejecución del **Buscador de Caminos**, se instancia la clase *HTTPServer* pasándole como parámetro una instancia de *BuscadorRequestHandler*. Esta instancia será la encargada de determinar a qué función de la API de la clase *Buscador de Caminos* (explicada en la Sección 4.2.2 del Capítulo 3 e implementada como un Singleton [14] en este trabajo) invocar de acuerdo a la URL y los parámetros recibidos en cada requerimiento. También se encarga de generar el código HTML y JavaScript necesario para mostrar los resultados de las operaciones del **Buscador de Caminos**.

4.2.2.2. Modo de uso

Al acceder al **Buscador de Caminos** desde un navegador web, se muestra el plano del espacio indoor correspondiente sobre el mapa provisto por Google Maps y un formulario con las diferentes opciones de interacción:

- Buscar un camino entre dos puntos de interés del espacio indoor representado
- Buscar un camino desde un punto de interés del espacio indoor representado hasta un punto externo al espacio indoor

- Buscar un camino desde un punto externo al espacio indoor hasta un punto de interés del espacio indoor representado

El *espacio indoor* es presentado sobre el mapa usando las clases *Polyline* y *Circle* provistas por la librería de JavaScript de Google Maps. Las líneas representan los límites de las áreas de interés y los segmentos de circulación interna (con diferente color), y los círculos representan puntos de interés, puntos de conexión y puntos de acceso.

Como la representación del espacio indoor utiliza coordenadas cartesianas, es necesario aplicar una transformación de coordenadas para que, a partir de la información de georreferencia del espacio indoor, se puedan obtener las coordenadas geográficas que necesitan las instancias de *Polyline* y *Circle* para dibujar sobre el mapa. Como explicamos en la Sección 4.2.1, en nuestra implementación las funciones de transformación de coordenadas son provistas por la librería de Google Maps.

Para que el usuario pueda elegir el origen y el destino, la interface cuanta con una lista de puntos de interés indoor y un campo autocompletable por Google para ingresar los puntos de interés outdoor, como puede verse en la Figura 16. En el caso de realizar una búsqueda *Indoor-Indoor* se muestran dos listas con puntos de interés indoor.

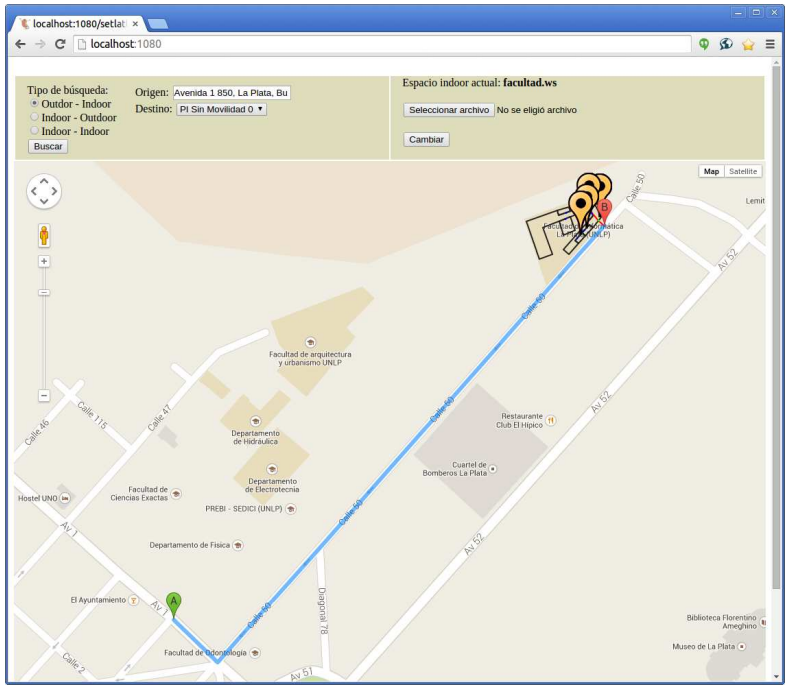


Figura 16: El **Buscador de caminos** mostrando un camino

En la Figura 16 también se puede ver como se muestran los resultados de una búsqueda de caminos usando un navegador web, considerando que el origen es un punto externo y el destino está dentro de una representación indoor. Los globos amarillos que aparecen sobre el el espacio indoor son los puntos de interés indoor. El globo rojo etiquetado como **B** por Google Maps indica la unión entre la parte outdoor y la parte indoor del camino.

Si tanto el origen como el destino del camino buscado pertenecen al mismo espacio indoor, el **Buscador de Caminos** se limita a usar la librería `libgis` para obtener el camino y luego aplica la transformación de coordenadas correspondiente para dibujar el camino sobre el mapa, usando la clase *Polyline* de la librería de Google Maps.

Si el origen o el destino es un punto externo a la representación indoor, se utiliza una instancia de la clase *DirectionsService* de la librería de Google Maps para encontrar y dibujar un camino entre el punto externo solicitado y el punto de acceso al espacio indoor más cercano. Luego se procede como en el caso anterior, esta vez entre dicho punto de acceso y el punto de interés buscado.

4.3. Conclusiones

En este capítulo hemos presentado la implementación del modelo y las herramientas introducidas en el Capítulo 3. Una de las herramientas presenta una interface gráfica para que el usuario defina espacios indoor, mientras que la otra integra de manera transparente al usuario la búsqueda de caminos indoor y outdoor y su visualización.

Capítulo 5

Casos de ejemplo de uso de las herramientas visuales

En este capítulo se presenta el funcionamiento de las herramientas implementadas mediante ejemplos de uso particulares de las mismas. En primer lugar se mostrará cómo crear una representación de un espacio indoor y a continuación se verá cómo realizar y visualizar la búsqueda de caminos entre puntos de dicho espacio indoor y del espacio outdoor.

5.1. Editor

Como explicamos en el Capítulo 4, el **Editor** provee la funcionalidad para mostrar un archivo en formato DXF que sirva de guía para definir el espacio indoor. Esto se logra indicando la ubicación del archivo como primer parámetro al ejecutar la aplicación. Por ejemplo, para abrir el **Editor** usando el archivo *facultad.dxf* como guía, se usa el siguiente comando en una consola:

```
python editor.py facultad.dxf
```

La Figura 17 muestra cómo se ve el **Editor** mostrando un archivo DXF con los planos de una Facultad de Informática simplificada.

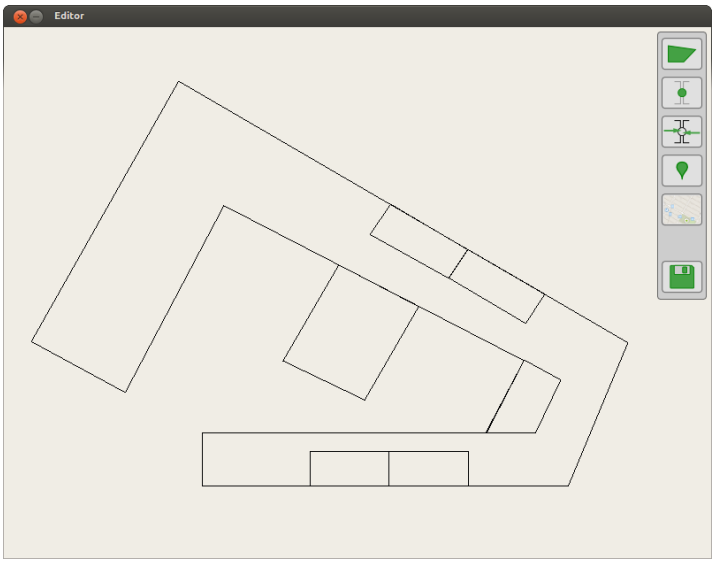
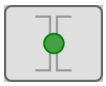
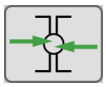


Figura 17: El **Editor** mostrando un plano en formato DXF.

A la derecha de la imagen de la Figura 17 puede observarse la barra de herramientas del **Editor**, la cual cuenta con las siguientes funciones:


Crear áreas de interés


Crear puntos de conexión


Conectar áreas de interés


Crear puntos de interés


Georreferenciar


Guardar

Al hacer click sobre cada una de ellas, el **Editor** entra en el modo correspondiente.

En el modo de *creación de áreas de interés*, cada click del botón izquierdo del mouse define un vértice del área de interés a crear. Cuando se hayan definido todos los vértices, el usuario debe apretar la tecla Enter para crear efectivamente el área de interés. En la Figura 18 pueden verse algunas áreas de interés ya creadas, en azul, y otra en proceso de creación,

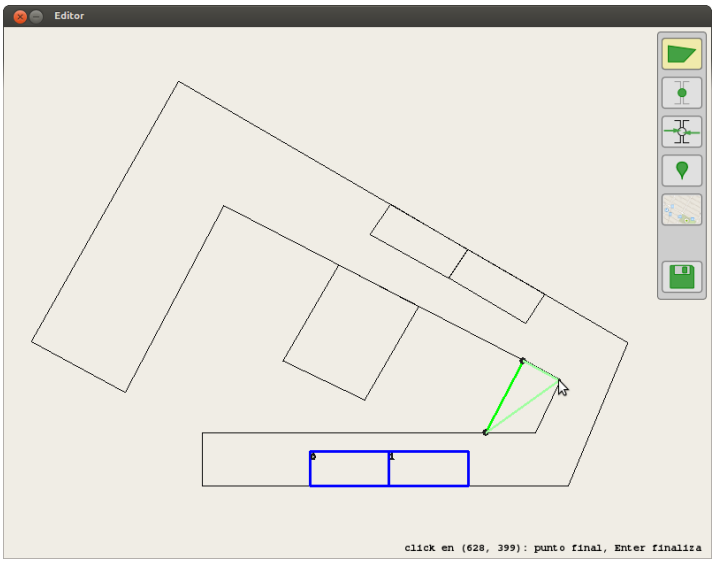


Figura 18: Creación de áreas de interés en el **Editor**.

en verde. Las áreas azules ya son instancias de la clase *AreaInteresSimple* y están almacenadas en la instancia de la clase *EspacioIndoor* que crea automáticamente el **Editor** al iniciarse.

En el modo de *creación de puntos de conexión*, cada click del mouse crea un punto de conexión, tal como se muestra en la Figura 19. Los puntos no se convierten en instancias de clases del modelo de *Movilidad* hasta que no sean usados efectivamente para conectar dos áreas de interés.

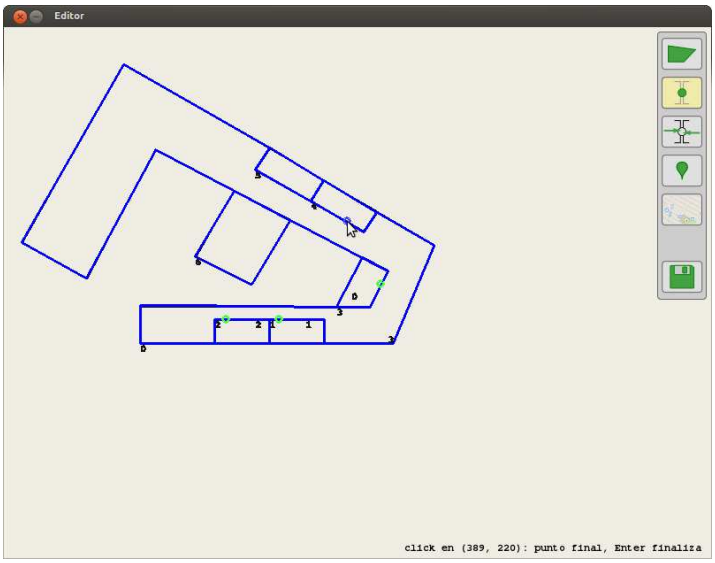


Figura 19: Creación de puntos de conexión en el **Editor**.

Una vez creados los puntos de conexión, el usuario debe indicar explícitamente qué áreas de interés están conectando, lo que se logra en el modo

de *conexión de áreas de interés*. En este modo el usuario debe usar el mouse para indicar primero el conector y luego las áreas de interés a conectar. Para facilitar esta tarea, el **Editor** irá resaltando los conectores o áreas de interés por las que pase el mouse, según corresponda. En la Figura 20 se puede ver cómo está resaltada una de las áreas a conectar. Al finalizar

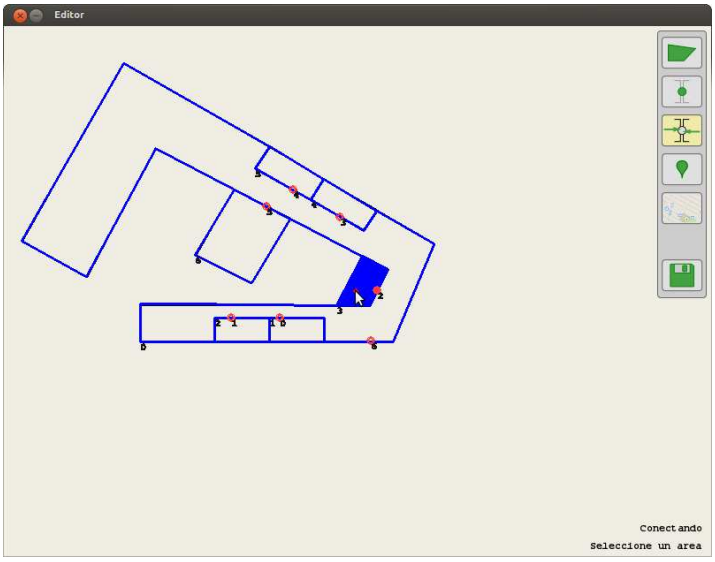


Figura 20: Conexión de áreas de interés en el **Editor**.

esta operación se crean las instancias de las clases *PuntoConexión* y se las relaciona con las áreas de interés conectadas.

En la Figura 21 puede verse al **Editor** en el modo de *creación de puntos*

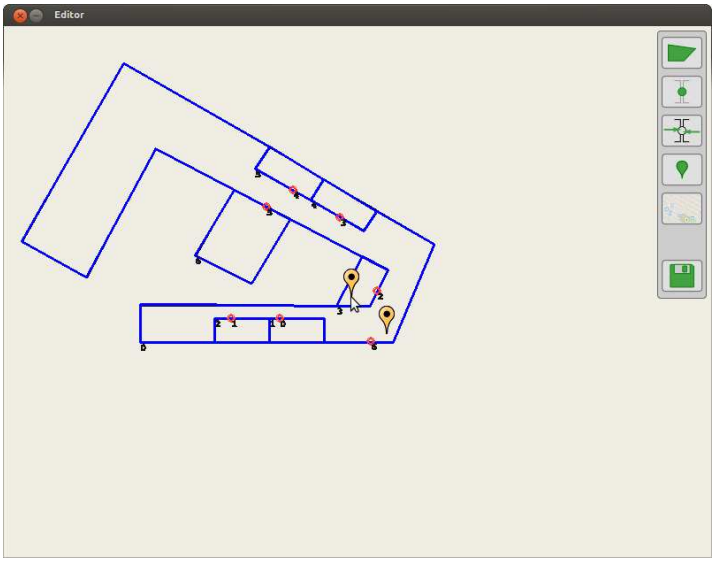


Figura 21: Creación de puntos de interés en el **Editor**.

de interés. En este caso, cada click del mouse creará una nueva instancia de la clase *PuntoInteres*, que se visualizará en la interface como un globo amarillo de Google.

Este prototipo del **Editor** solamente crea una movilidad del tipo *Transitar Áreas Interés* para todo el ejemplo. Esta movilidad se va actualizando automáticamente a medida que se conectan las áreas y puntos de interés.

En el espacio indoor de este ejemplo, los puntos de interés son el "escritorio de atención al público" de la Biblioteca, la ventanilla de "Atención a alumnos", la ventanilla de "Mesa de entradas" y "Mesa de información".

El modo de *georreferenciación* es el que permite darle coordenadas geográficas al espacio indoor. En este modo el **Editor** obtiene un mapa de Google Maps y lo muestra bajo el espacio indoor que estamos creando, como se observa en la Figura 22.

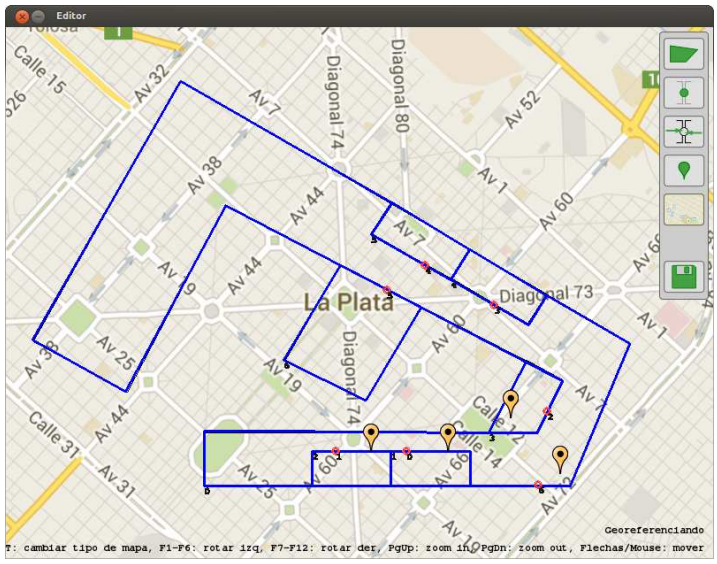


Figura 22: El **Editor** mostrando un mapa para georreferenciar el espacio indoor.

En la Figura 22 ya están todas las áreas y los puntos de interés creados, y se establecieron las conexiones entre ellos. Para georreferenciar el espacio indoor es necesario hacerlo coincidir con su ubicación en la representación outdoor utilizada. Para esto el **Editor** provee una serie de funciones para manipular el mapa de referencia:

- Cambiar el tipo de mapa: cada vez que se presione la letra T del teclado, el tipo de mapa de referencia cambia entre los disponibles por Google Maps (normal, satelital, híbrido y terreno).

- Rotar el mapa: usando las teclas de Función del teclado el usuario puede rotar el mapa de referencia. Las teclas F1 a F6 rotan el mapa en sentido antihorario, cada una de ellas con diferente magnitud (siendo F1 mayor ángulo de rotación y F6 menor). Las teclas F7 a F12 rotan el mapa en sentido horario, también con diferente magnitud (siendo F12 mayor ángulo de rotación y F7 menor).
- Cambiar el zoom del mapa: las teclas RePág y AvPág incrementan y decrementan, respectivamente, el nivel de zoom del mapa.
- Escalar el espacio indoor: con las teclas + y - del teclado se puede ajustar el nivel de escala que se aplica al espacio indoor.
- Mover el mapa: el mapa puede moverse haciendo drag&drop con el mouse o usando las teclas de dirección del teclado.

En la Figura 23 se puede ver cómo, luego de utilizar las funciones antes descriptas, se logra que el espacio indoor quede con una ubicación georreferenciada acorde al lugar en el mapa de Google Maps en el que se encuentra aproximadamente la Facultad de Informática.

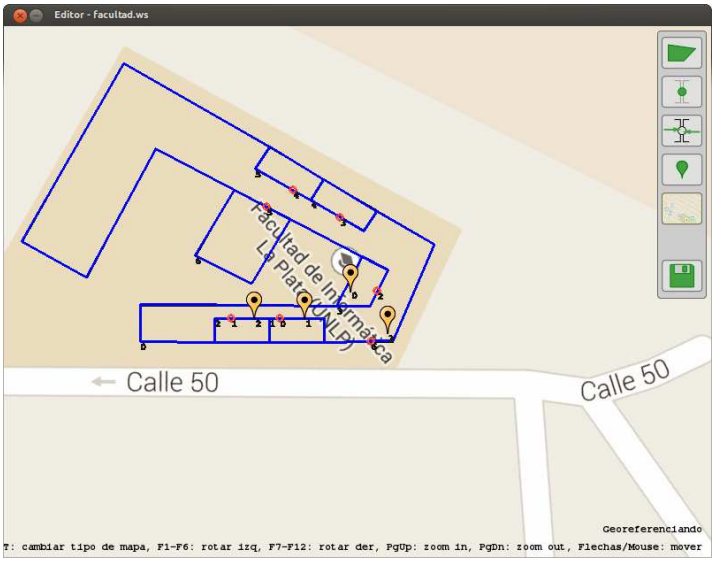


Figura 23: Espacio indoor georreferenciado en el **Editor**.

Finalmente, el último botón de la barra de herramientas nos permite guardar el espacio indoor creado en un archivo XML, para su uso posterior en el **Buscador de Caminos** o para seguir editándolo en el mismo **Editor**.

5.2. Buscador de Caminos

Para configurar el **Buscador de Caminos** con una representación de un espacio indoor hay dos alternativas. La primera es al momento de iniciar el servidor web, indicando como primer parámetro de la línea de comandos el archivo con la representación de un espacio indoor. Por ejemplo, para iniciar el servidor web con un espacio indoor por defecto, cuya representación esté guardada en el archivo *facultad.xml*, se usa la siguiente línea de comando:

```
python webserver.py facultad.xml
```

La interface del **Buscador de Caminos**, presentada en la Figura 24, tiene tres secciones: en la primera, arriba a la izquierda, se configuran los parámetros de la búsqueda. En la segunda, arriba a la derecha, se muestra el nombre del archivo con la representación del espacio indoor que se está usando y se puede elegir otro para reemplazar al actual. Por último, en el resto de la pantalla se muestra el mapa con la representación del espacio indoor actual y los caminos encontrados, en caso de haberse realizado una búsqueda.

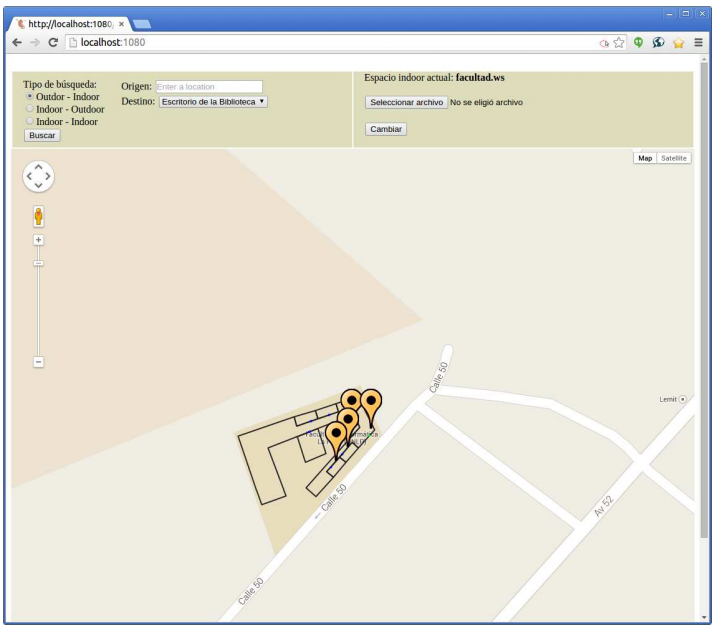


Figura 24: Estado inicial del **Buscador de Caminos**.

Como puede verse en la Figura 24, hay un botón etiquetado "Seleccionar archivo" mediante el cual el usuario puede seleccionar un archivo

con una representación de un espacio indoor. Con el botón "Cambiar" se hace efectiva la selección, reemplazando el espacio indoor existente con el indicado anteriormente y reflejándose este cambio visualmente.

En la Figura 24 pueden verse las tres secciones con los siguientes datos: el tipo de búsqueda que se va a realizar es *Outdoor-Indoor*, el espacio indoor utilizado es el que se encuentra en el archivo llamado *facultad.xml* y en el área principal solamente se ve dicho espacio indoor posicionado sobre el mapa, sin ningún camino porque no se ha realizado una búsqueda aún.

Para los tipos de búsqueda *Outdoor-Indor* e *Indoor-Outdoor*, el usuario debe elegir de una lista el punto de interés perteneciente a la parte indoor de la búsqueda e indicar en el campo de texto el extremo outdoor del camino deseado. Para facilitar esta tarea, el campo de texto provee la funcionalidad de autocompletado, usando el soporte que provee Google Maps, como puede verse en la Figura 25.

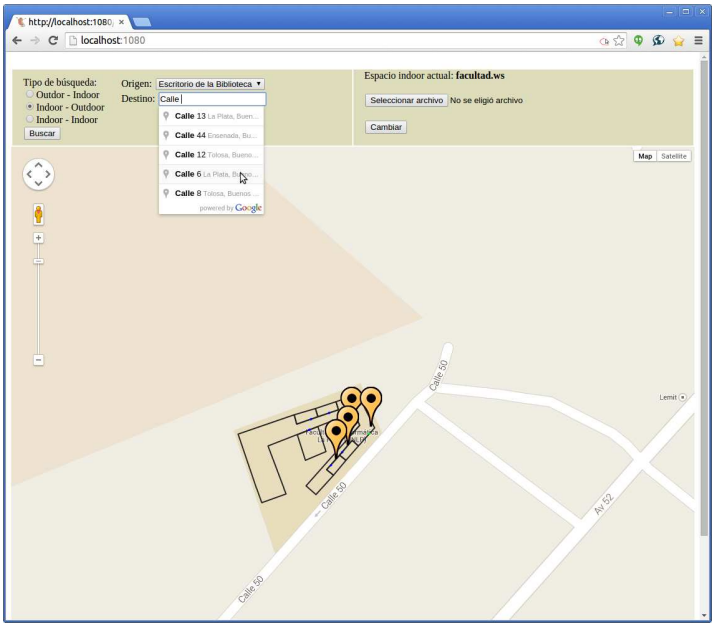


Figura 25: Autocompletado de opciones outdoor.

Para mostrar el funcionamiento del **Buscador de Caminos**, se buscará un camino entre las calles 1 y 49 y el escritorio de atención al público de la Biblioteca de la Facultad de Informática, que fue creado como un punto de interés del espacio indoor.

El usuario debe elegir el "Tipo de búsqueda" Outdoor-Indoor, completar la dirección deseada en el campo de texto "Origen" y en la lista de "Destino" elegir el Escritorio de la Biblioteca. Cuando presione el botón

"Buscar", el **Buscador de Caminos** buscará y mostrará el camino deseado, como se ve en la Figura 26.

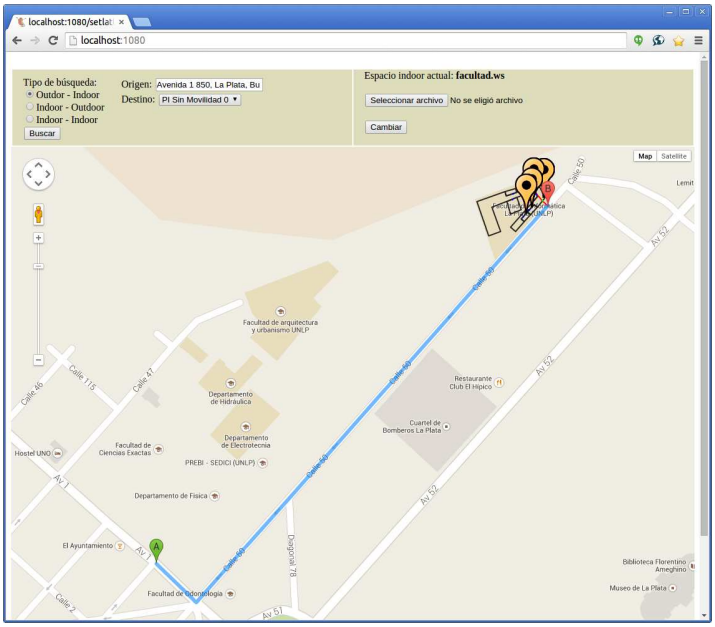


Figura 26: Camino Outdoor-Indoor y puntos de interés indoor.

En la Figura 27 se puede apreciar en más detalle cómo el **Buscador de Caminos** conectó la búsqueda outdoor provista por Google Maps con

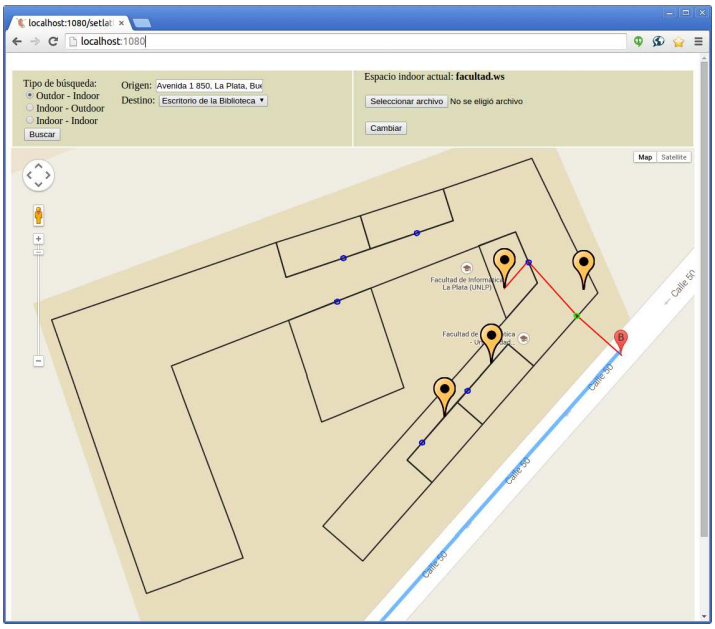


Figura 27: Conexión entre la parte outdoor y la parte indoor del camino.

la búsqueda indoor provista por **libgis**, logrando que el camino "ingrese" al edificio de la Facultad por uno de los puntos de acceso disponibles para el espacio indoor. La parte indoor del camino está representada en rojo mientras que la parte outdoor es azul. El usuario puede hacer zoom sobre el mapa utilizando la funcionalidad de Google Maps para ver en más detalle cualquiera de los dos tipos de caminos.

Por último, mostraremos cómo se realiza una búsqueda entre dos puntos de interés dentro del mismo espacio indoor. Por ejemplo, entre el escritorio de la Biblioteca de la Facultad de Informática y el escritorio de la mesa de informes de la Facultad. La Figura 28 muestra el resultado obtenido luego de haber seleccionado el "Tipo de búsqueda" Indoor-Indoor y los puntos de interés correspondientes en las listas de "Origen" y "Destino".

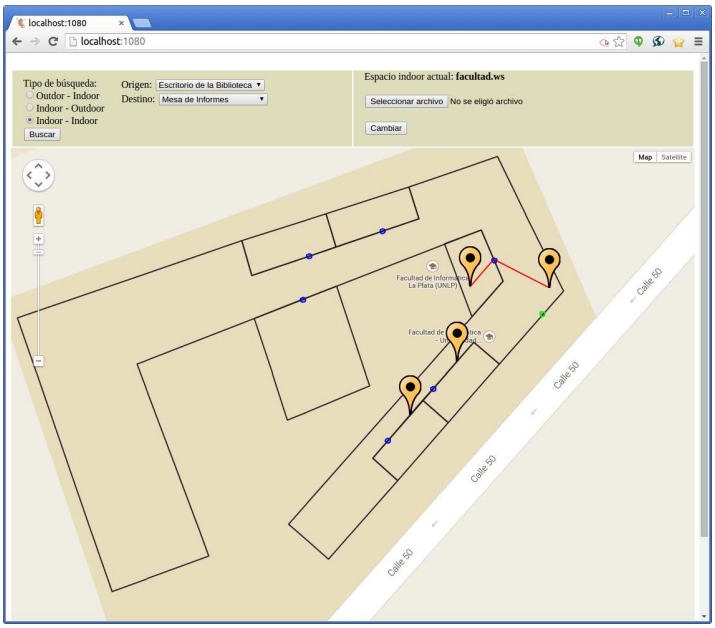


Figura 28: Camino Indoor-Indoor encontrado por el **Buscador de Caminos**.

En este caso, el **Buscador de Caminos** realiza la búsqueda empleando solamente la movilidad definida para el espacio indoor, mediante la funcionalidad provista por **libgis**. Al igual que en los otros tipos de búsqueda, el camino obtenido se muestra mediante la interface de Google Maps, lo que permite mover el mapa y hacer zoom en él.

5.3. Conclusiones

En este capítulo presentamos, mediante un caso de ejemplo, la forma de utilización de las herramientas visuales propuestas. En primer lugar se definió usando el **Editor** un *espacio indoor* compuesto por siete áreas de interés simples, con conexiones entre algunas de ellas, y cuatro puntos de interés. A continuación usamos dicho *espacio indoor* en el **Buscador de Caminos** para realizar dos tipos de búsquedas diferentes: una con origen en un punto de interés indoor y destino en un punto outdoor y otra búsqueda con origen y destino en puntos de interés indoor. En el primer caso se vió la manera en que la herramienta visual integra la parte outdoor del camino encontrado con la parte indoor. En el segundo caso se pudo ver que se puede utilizar solamente la funcionalidad de búsqueda de caminos indoor presente en el modelo implementado.

Capítulo 6

Conclusiones y Trabajos futuros

En este Capítulo se exponen las conclusiones y se presentan los trabajos futuros a realizar sobre el presente trabajo.

6.1. Conclusiones

En este trabajo se mostró cómo extendiendo el modelo presentado en [7] se desarrollaron dos herramientas prototípicas: una para definir visualmente espacios indoor y otra para la búsqueda y visualización de caminos entre posiciones que pueden estar ubicados indistintamente en espacios outdoor como indoor.

Las extensiones involucraron la definición de un mecanismo de representación y manipulación visual de los objetos del modelo geográfico y de movilidad, así como la creación de un módulo de persistencia para dichos objetos. Además se reemplazó la representación del espacio outdoor propia del modelo para utilizar la búsqueda de caminos outdoor suministrados por proveedores externos.

La modularización de la implementación del modelo original y de las extensiones propuestas en este trabajo permite tener independencia a la hora de implementar cada una, así como el intercambio de implementaciones de ser necesario. También facilita el desarrollo de extensiones a alguno de los módulos o de nuevas herramientas utilizando la funcionalidad existente.

6.2. Trabajos futuros

A continuación se nombran algunos de los puntos de extensión a partir de lo desarrollado en este trabajo.

En relación a la herramienta visual **Editor**:

- Implementar en el **Editor** los otros tipos de movilidades descriptas en el modelo (*TransitarAreasInteres* y *RedCirculacionInterna*).
- Integrar la funcionalidad de carga de archivos DXF desde la interface visual del **Editor**
- Agregar otro mecanismo de persistencia del modelo subyacente, por ejemplo en formato JSON, archivos binarios o bases de datos.
- Implementar una herramienta equivalente al **Editor** pero como una aplicación web, lo que permite que la edición sea colaborativa y poder acceder a la funcionalidad de edición desde diversos dispositivos.
- Incorporar otros formatos de archivos para visualizar planos, como por ejemplo el formato Shapefile.
- Incluir las herramientas necesarias para que se puedan realizar representaciones de espacios indoor multiniveles

En relación a la herramienta visual **Buscador de Caminos**:

- Incorporar otros proveedores externos para el cálculo de caminos outdoor. Actualmente el **Buscador de Caminos** utiliza sólo a Google Maps como proveedor para las búsquedas de caminos outdoor.
- Definir estrategias para determinar por qué punto de acceso ingresar o salir de un espacio indoor. Durante la búsqueda de caminos, en caso de que haya varios puntos de acceso al punto de interés destino, es necesario elegir uno de ellos al momento de realizar la unión entre los tramos indoor y outdoor de un camino.
- Incluir la elección del medio de locomoción en los tramos de búsqueda outdoor. Los proveedores de búsqueda de caminos outdoor brindan la posibilidad de elegir el medio de locomoción para el que se va a calcular el camino.

- Ampliar el comportamiento de la clase *BuscadorIntegrado* para que incluya la estrategia de búsquedas de puntos de interés contenidos en diferentes representaciones indoor.
- Desarrollar la algoritmia necesaria para soportar la búsqueda de caminos en espacios indoor multiniveles.
- Generalizar la interface visual para que el usuario no tenga que definir explícitamente el tipo de búsqueda (indoor-indoor, indoor-outdoor, outdoor-indoor).

Bibliografía

[1] Abrahams, D., Grosse-Kunstleve, R., "Building Hybrid Systems with Boost.Python", Boost Consulting. 2003.

[2] Alexandrescu, A. , "Modern C++ Design: Generic Programming and Design Patterns Applied", Addison-Wesley Professional. 2001.

[3] Avila-Rodriguez, J., Wallner, S., Hein, G. W., "How to Optimize GNSS Signals and Codes for Indoor Positioning". 2006.

[4] Bauss, W., "Radio Navigation Systems for Aviation and Maritime Use, a comparative study", Pergamon Press. 1963.

[5] Broumandan, A., Nielsen, J., Lachapelle, G. "Indoor GNSS Signal Acquisition Performance using a Synthetic Antenna Array". 2011.

[6] "The Bat System", Computer Laboratory, Digital Technology Group, Universidad de Cambridge. 2006.

[7] Challiol, C., Lliteras, A. "Movilidades indoor-outdoor para aplicaciones GIS". Reporte Técnico. LIFIA, Facultad de Informática, UNLP. 2007.

[8] Curran, K., Furey, E., Lunney, T., Santos, J., Woods, D. "An Evaluation of Indoor Location Determination Technologies". 2011.

[9] Cypriani, M., Lassabe, F., Canalda, P., Spies, F. "Wi-Fi-Based Indoor Positioning: Basic Techniques, Hybrid Algorithms and Open Software Platform". In International Conference on Indoor Positioning and Indoor Navigation (IPIN). Zurich, Suiza. 2010.

[10] Durante, L., Ilarragorri, C., "Implementación de movilidad Indoor-Outdoor para aplicaciones GIS basada en en modelo Orientado a Objetos". Tesina de grado. Facultad de Informática, UNLP. 2008

[11] Feng, J., Watanabe, T. "Road Network Model. In Index and Query Methods in Road Networks" (pp. 41-69). Springer International Publishing. 2015.

[12] Figueiras, J., Frattasi, S., "Mobile positioning and tracking: From conventional to cooperative techniques". John Wiley & Sons, 2011.

[13] Fowler, M, "Refactoring. Improving the Design of Existing Code", Addison-Wesley. 1999.

[14] Gamma E., Helm R., Johnson R., Vlissides J., "Design Patterns, Elements of Reusable Object-Oriented Software", Addison-Wesley Professional. 1994 .

[15] Giudice, N. A., Walton, L. A., Worboys, M., "The informatics of indoor and outdoor space: a research agenda". In Proceedings of the 2nd ACM SIGSPATIAL International Workshop on Indoor Spatial Awareness (pp. 47-53). ACM. 2010.

[16] Goldberg A., Robson D. "Smalltalk-80 The Language and its implementation", Addison-Wesley, 1983.

[17] Güting, R. H., "An Introduction to Spatial Database Systems", Fern Universität Hagen, Germany. 1994.

[18] Hofmann-Wellenhof B., Lichtenegger H., Wasle E., "GNSS - Global Navigation Satellite Systems: GPS, GLONASS, Galileo & more". Springer. 2007.

[19] Hsu, L.T., Lin, Y.H., Tsai, W.M., Jan, S.S. "Development of Indoor Positioning and Location Based Information Test Bed". 2009.

[20] Ibrahim, M., Youssef, M. "CellSense: A Probabilistic RSSI-based GSM Positioning System", Nile University, Egipto. 2010.

[21] Kazemi, P. L. "Development of New Filter and Tracking Schemes for Weak GPS Signal Tracking". pp 1-8. 2010.

[22] Leonhardt, U. "Supporting Location-Awareness in Open Distributed Systems" Ph.D. Thesis, Dept. of Computing, Imperial College, London. 1998.

[23] Li, K.. "Indoor space: A new notion of space." En Web and wireless geographical information systems, pags. 1-3. Springer Berlin Heidelberg, 2008.

[24] Lliteras A., Challiol C., Mostaccio M. y Gordillo S.. "Representaciones enriquecidas para la navegación indoor-outdoor en aplicaciones móviles". CACIC 2011. Octubre de 2011. Facultad de Informática, UNLP. Con referato. ISBN 978-950-34-0756-1 Pags. 867-876. 2011

[25] Longley, P. "Geographic information systems and science". John Wiley & Sons. 2005

[26] Muñoz-Organero, M., Muñoz-Merino, P. J., Delgado Kloos, C. "Using Bluetooth to Implement a Pervasive Indoor Positioning System with Minimal Requirements at the Application Level", Universidad Carlos III de Madrid, España. 2012

[27] Ni, L., Liu, Y., Lau, Y. C., Patil, A. "LANDMARC: Indoor Location Sensing Using Active RFID". Wireless Networks 10. 2004.

[28] National Imagery and Mapping Agency "World Geodetic System 1984" Technical report, United States Department of Defense. 2000.

[29] Nossum, A. S., Midtbø, T., Haakonsen, T. A., Nordan, R. P. V. "Are indoor positioning systems mature for cartographic tasks?". In Proceedings of AutoCarto 2012. 2012.

[30] Park, J., Charrow, B., Curtis, D., Battat, J., Minkov, E., Hicks, J., Teller, S., Ledlie, J. "Growing an organic indoor location system." In Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services (MobiSys 10). ACM, New York, NY, USA, pp 271-284. 2010.

[31] Priyantha, N. B., "The Cricket Indoor Location System". Massachusetts Institute of Technology. 2005.

[32] Raymond, E., "The Art of UNIX Programming", Addison-Wesley Professional. 2003.

[33] Richter, K. F., Winter, S., Ruetschi, U. J. "Constructing hierarchical representations of indoor spaces". In Mobile Data Management: Systems, Services and Middleware. MDM'09. Tenth International Conference (pp. 686-691). IEEE. 2009.

[34] Rizos, C., Roberts, R., Barnes, J., Gambale, N. "Locata: A new high accuracy indoor positioning system", International Conference on Indoor Positioning and Indoor Navigation (IPIN), Zürich, Switzerland. 2010.

[35] Srinivasan V. "Standardizing the specification, verification, and exchange of product geometry: Research, status and trends". 2007.

[36] `http://httpd.apache.org/docs/`, fecha de última visita: 25 de Septiembre de 2014.

[37] `http://www.boost.org/`, fecha de última visita: 25 de Septiembre de 2014.

[38] Colvin G., Dawes b., `http://www.boost.org/doc/libs/1_55_0/libs/smart_ptr/smart_ptr.htm`. 2012, fecha de última visita: 25 de Septiembre de 2014.

[39] Ramey R., `http://www.boost.org/doc/libs/1_55_0/libs/serialization/doc/`. 2004, fecha de última visita: 25 de Septiembre de 2014.

[40] Abrahams D., `http://www.boost.org/doc/libs/1_55_0/libs/python/doc/`. 2003, fecha de última visita: 25 de Septiembre de 2014.

[41] `http://www.open-std.org/jtc1/sc22/wg21/`, fecha de última visita: 25 de Septiembre de 2014.

[42] `https://bitbucket.org/Coin3D/dime`. 2012, fecha de última visita: 25 de Septiembre de 2014.

[43] `https://developers.google.com/maps/documentation/javascript/`, fecha de última visita: 25 de Septiembre de 2014.

[44] `https://developers.google.com/maps/documentation/javascript/maptypes#MapCoordinates`, fecha de última visita: 25 de Septiembre de 2014.

[45] `https://developer.mozilla.org/en-US/docs/Web/JavaScript`, fecha de última visita: 25 de Septiembre de 2014.

[46] `http://redmine.lighttpd.net/projects/lighttpd`, fecha de última visita: 25 de Septiembre de 2014.

[47] `http://postgis.net/`, fecha de última visita: 25 de Septiembre de 2014.

[48] `http://www.postgresql.org/`, fecha de última visita: 25 de Septiembre de 2014.

[49] `http://www.pygame.org/wiki/about`, fecha de última visita: 25 de Septiembre de 2014.

[50] `https://docs.python.org/2/c-api/`, fecha de última visita: 25 de Septiembre de 2014.

[51] `http://www.libsdl.org/index.php`, fecha de última visita: 25 de Septiembre de 2014.

[52] `https://www.python.org/`, fecha de última visita: 25 de Septiembre de 2014.

[53] `http://www.riverbankcomputing.com/software/sip/intro`, fecha de última visita: 25 de Septiembre de 2014.

[54] `http://www.smalltalk.org`, fecha de última visita: 25 de Septiembre de 2014.

[55] `http://www.swig.org/Doc3.0/SWIG.html`, fecha de última visita: 25 de Septiembre de 2014.

[56] Weisstein, Eric W. "Mercator Projection." From MathWorld–A Wolfram Web Resource. `http://mathworld.wolfram.com/MercatorProjection.html`, fecha de última visita: 25 de Septiembre de 2014.

[57] Worboys, M. "Modeling indoor space." En Proceedings of the 3rd ACM SIGSPATIAL International Workshop on Indoor Spatial Awareness, pags. 1-6. ACM. 2011.

[58] Worth, H., Warren, M. "TRANSIT TO TOMORROW Fifty Years of Space Research at The Johns Hopkins University Applied Physics Laboratory", The Johns Hopkins University Applied Physics Laboratory, 2009.

[59] Yang, L., Worboys, M., "Similarities and differences between outdoor and indoor space from the perspective of navigation". Poster presentado en COSIT 2011.

[60] Yilin Zhao, "Mobile Phone Location Determination and Its Impact on Intelligent Transportation Systems", IEEE Transactions on intelligent transportation systems, Vol. 1, No. 1. 2010.

[61] Zumberge and Gendt "The demise of selective availability and implications for the international GPS service network". 2000.

Apéndice A

Ejecución y Compilación

En este apéndice se describe el contenido del CD entregado junto a este trabajo, que consiste en el código fuente de las herramientas visuales presentadas y los archivos ejecutables de las mismas. Se incluye además, un texto describiendo cómo ejecutar las herramientas visuales una vez compiladas. Finalmente, se detallan las dependencias de software necesarias para compilar y ejecutar las herramientas visuales y se describen los pasos para compilar el código fuente.

A.1. Estructura de carpetas

A continuación se muestra la estructura de carpetas del CD entregado. Cabe aclarar que para compilar o ejecutar las herramientas es recomendable copiar el contenido a una carpeta local.

```
/
    bin/
        gis/
            libgis.so
            libgispy.so
        gispycommon/
            gispycommon/
                <scripts de Python>
        editor/
            editor/
```

```
            <scripts de Python>
        run.py
    buscador/
        buscador/
            <scripts de Python>
        run.py
ejemplos/
    facultad.dxf
    facultad.xml
src/
    buscador/
    editor/
    gis/
    gispy/
    gispycommon/
doc/
    mendoza_tarantola.pdf
```

Dentro de la carpeta `bin` se encuentran las librerías y herramientas visuales compiladas en una máquina con sistema operativo Ubuntu Linux de 64 bits y listas para ejecutar en un ambiente de similares características.

En la carpeta `ejemplos` está el archivo DXF usado para los ejemplos del Capítulo 5, así como el XML con el espacio indoor generado en ese mismo capítulo.

Dentro de la carpeta `src`, en tanto, se encuentra el código fuente escrito para este trabajo. Dentro de esta carpeta hay cinco subcarpetas (`gis`, `gispy`, `editor`, `buscador` y `gispycommon`) cuyo contenido se detalla a continuacion.

La carpeta `gis` contiene el código fuente en C++ necesario para implementar el concepto de *espacio indoor* y la funcionalidad de las capas *Geográfica* y *Movilidad*. Al compilar el contenido de esta carpeta se obtiene la librería **libgis**.

La carpeta `gispy` contiene el código fuente en C++ necesario para hacer el binding entre **libgis** y las herramientas visuales escritas en Python, además de proveer funcionalidad extra a dichas herramientas: lectura de archivos DXF y conversión a un formato manejable por las herramientas

visuales. Al compilar el contenido de esta carpeta se obtiene la librería **libgispy**.

La carpeta `editor` contiene el código fuente en Python del **Editor**. Este código es ejecutable directamente por el intérprete de Python, pero depende de la presencia de la librería dinámica **libgispy**.

La carpeta `buscador` contiene el código fuente en Python del **Buscador de Caminos**. Este código es ejecutable directamente por el intérprete de Python, pero depende de la presencia de la librería dinámica **libgispy**.

La carpeta `gispycommon` contiene el código fuente en Python de la funcionalidad compartida entre el **Editor** y el **Buscador de Caminos**.

Por último, la carpeta `doc` contiene el pdf con este trabajo.

A.2. Compilación

En esta sección explicaremos como configurar el ambiente para compilar **libgis** y **libgispy**, para poder luego ejecutar las dos herramientas visuales (**Editor** y **Buscador de caminos**). En primer lugar se detallarán las dependencias para la compilación de las librerías, nombrando los paquetes de Ubuntu que es necesario instalar para cada uno. Finalmente se explicará cómo compilarlas.

A.2.1. Dependencias

El proceso de compilación de **libgis** y **libgispy** se controla mediante la herramienta *CMake* (versión 2.8), que es necesario tener instalada antes de comenzar el proceso. En Ubuntu alcanza con instalar el paquete *cmake*. También es necesario instalar un compilador de C++, en nuestro caso utilizamos *gcc* versión 4.7.2 (En Ubuntu, paquete *gcc*), pero puede usarse cualquier compilador que cumpla con el estándar ISO/IEC 14882:2003 (informalmente conocido como "C++03").

Para poder compilar correctamente a **libgis** es necesario tener instaladas las siguientes librerías:

- Librería de base de la Boost versión 1.49 o superior (en Ubuntu, paquete *libboost-dev*).

- Librería de serialización de la Boost versión 1.49 o superior (en Ubuntu, paquete *libboost-serialization-dev*).

Para poder compilar correctamente a **libgispy** es necesario tener instaladas las siguientes librerías:

- Librería de base de la Boost versión 1.49 o superior (en Ubuntu, paquete *libboost-dev*).
- Encabezados y librerías para desarrollo de Python versión 2.7 (en Ubuntu, paquete *python-dev*).
- Librería *Boost.Python* para desarrollo versión 1.49 o superior (en Ubuntu, paquete *libbost-python-dev*).
- Librería de serialización de la Boost versión 1.49 o superior (en Ubuntu, paquete *libboost-serialization-dev*).
- Librería *dime* para desarrollo versión 0.20030921-2 (en Ubuntu, paquetes *libdime-dev* y *libdime1*)

Las dos herramientas visuales están escritas en Python. Para ejecutarlas es necesario tener instalado un intérprete de la versión 2.7 de Python (en Ubuntu, paquete *python*). Por último, para ejecutar la herramienta visual **Editor** es necesario tener instalada además la librería *pygame* versión 1.9.1 o superior (En Ubuntu alcanza con instalar el paquete *python-pygame*).

A.2.2. Compilación

Una vez que todas las dependencias estén instaladas correctamente puede procederse a compilar las librerías **libgis** y **libgispy** (para poder luego ejecutar las dos herramientas visuales) mediante los siguientes pasos:

- Abrir una terminal de línea de comando.
- Posicionarse en la carpeta con los fuentes, por ejemplo `~/tesis/src/`.
  ```
  $cd ~/tesis/src
  ```
- Para mayor prolijidad, crear una carpeta para contener los archivos compilados (llamada por ejemplo "build") y entrar en ella.
  ```
  $mkdir build
  $cd build
  ```

- Ejecutar el comando para configurar la compilación.
  `$cmake ..`
- Ejecutar el comando para comenzar la compilación.
  `$make`
- Una vez que el comando anterior termine exitosamente, las dos librerías compiladas quedarán en *gis/libgis.so* y *gispy/libgispy.so*.
- Finalmente ejecutar:
  `$sudo make install.`

De esta manera se compilarán e instalarán en el sistema las dos librerías (**libgis** y **gispy**) lo que permitirá su uso desde cualquier ubicación.

Si bien el **Editor** y el **Buscador de Caminos** pueden ejecutarse directamente, también es posible instalarlos para poder accederlos desde cualquier ubicación. Para ello deben seguirse los siguientes pasos:

- Abrir una terminal de línea de comando.
- Posicionarse en la carpeta `gispycommon`, por ejemplo `~/tesis/src/gispycommon`.
  `$cd ~/tesis/src/gispycommon`
- Ejecutar el comando para instalar el paquete **gispycommon**.
  `$sudo python setup.py install`
- Posicionarse en la carpeta `editor`, por ejemplo `~/tesis/src/editor`.
  `$cd ~/tesis/src/editor`
- Ejecutar el comando para instalar el paquete **editor**.
  `$sudo python setup.py install`
- Posicionarse en la carpeta `buscador`, por ejemplo `~/tesis/src/buscador`.
  `$cd ~/tesis/src/buscador`
- Ejecutar el comando para instalar el paquete **buscador**.
  `$sudo python setup.py install`

De esta manera quedarán instalados en el sistema las dos herramientas visuales (**Editor** y **Buscador de Caminos**), pudiendo ser ejecutados sin necesidad de estar situados en una carpeta específica.

A.3. Ejecución

En esta sección se explicará como ejecutar con un sistema operativo Ubuntu de 64 bits las dos herramientas visuales que están en la carpeta `bin`. También se indica la forma de ejecutarla en caso de haber compilado e instalado las librerías y herramientas.

A.3.1. Editor

Para ejecutar el **Editor** basta con dirigirse, en una terminal de línea de comando, hasta la carpeta `bin/editor` mencionada en la Sección A.1. Una vez allí escribir:

```
$./run.py [<archivo_dxf>] [<archivo_xml>]
```

Si la aplicación fue instalada como se indica en la Sección A.2.2, se puede ejecutar escribiendo lo siguiente en una terminal de línea de comando:

```
$python -m editor [<archivo_dxf>] [<archivo_xml>]
```

El parámetro *archivo_dxf* es opcional e indica el archivo que contiene el plano que se va a usar como guía para el usuario. El parámetro *archivo_xml* también es opcional e indica un archivo que contiene un espacio indoor definido por esta herramienta visual para continuar con su edición.

Debe tenerse en cuenta que el **Editor** escribe el espacio indoor que genera en un archivo xml en la carpeta de ejecución, por lo que el usuario debe tener permiso de escritura en dicha carpeta.

A.3.2. Buscador de Caminos

Para ejecutar el **Buscador de Caminos** basta con dirigirse, en una línea de comando, hasta la carpeta `bin/buscador` mencionada en la Sección A.1. Una vez allí escribir:

```
$./run.py [<archivo_xml>]
```

Si la aplicación fue instalada como se indica en la Sección A.2.2, se puede ejecutar escribiendo lo siguiente en una terminal de línea de comando:

```
$python -m buscador [<archivo_xml>]
```

El parámetro *archivo_xml* especifica el archivo que contiene un espacio indoor definido por el usuario que el **Buscador de Caminos** va a usar inicialmente. Una vez arrancado el servidor, el usuario puede indicar otro espacio indoor desde la propia interface visual de la herramienta.

En cualquiera de los dos casos, el comando arranca un servidor que quedará escuchando peticiones en el puerto 1080. Para acceder al **Buscador de Caminos** hay que dirigirse, en un navegador web, a la url *http://localhost:1080*.