TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

# UNIVERSIDAD DE ALMERIA

## ESCUELA POLITÉCNICA SUPERIOR Y FACULTAD DE CIENCIAS EXPERIMENTALES

# JOSM – Urban

## Un plugin De Consultas XQuery sobre Núcleos Urbanos Para JOSM


**Curso 2013/2014**

**Alumno/a:**

Juan Carlos Fernández Zamora

**Director/es:**

Antonio Becerra Terón

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

UNIVERSIDAD DE ALMERIA

ESCUELA POLITÉCNICA SUPERIOR Y
FACULTAD DE CIENCIAS EXPERIMENTALES

# JOSM – Urban

## Un plugin De Consultas XQuery sobre Núcleos Urbanos Para JOSM


**Curso 2013/2014**

**Alumno/a:**
Juan Carlos Fernández Zamora

**Director/es:**
Antonio Becerra Terón

# *Agradecimientos*

*Quisiera agradecer el apoyo y comprensión a mi familia, que siempre ha estado a mi lado cuando la he necesitado, especialmente a mi madre por su apoyo incondicional.*

*Asimismo, expresar mi más sincero agradecimiento al Director de este Proyecto Fin de Carrera, Doctor D. Antonio Becerra Terón, por su ayuda, dedicación y disponibilidad.*

*Por último a la Universidad de Almería y a todos sus profesores por las enseñanzas aportadas.*

*Juan Carlos Fernández Zamora*

## *Resumen*

El principal objetivo de este PFG es la creación de un plugin que permita realizar consultas urbanas complejas en XQuery para el programa JOSM.

Este plugin permite la utilización de los lenguajes de consulta XPath y XQuery para realizar las consultas, además de incluir la librería de operadores topológicos propuesta por A. Becerra y J. Almendros.

El plugin será desarrollado en lenguaje Java, usando diferentes librerías. Seguirá una arquitectura cliente-servidor, utilizando como servidor de XQuery el programa BaseX. Seguirá las directrices marcadas por IPO en la interfaz gráfica, desarrollado bajo el patrón arquitectónico **ModeloVistaControlador**

En el proyecto se siguen las diferentes etapas de desarrollo del software (planificación, modelado, codificación, pruebas y documentación) explicando el trabajo desarrollado en cada una de las fases.

Entre las principales funcionalidades se encuentran, configurar la información de la conexión por el usuario, una **previsualización de consultas** y su ejecución rápida, guardando las soluciones generadas, además de mostrar las consultas en forma de lista y otras funcionalidades básicas como eliminar, añadir y modificar consultas.

Dentro del sistema existe, un tutorial explicativo sobre su utilización, Este tutorial muestra la información detallada acerca de sus funcionalidades.

# *Abstract*

The main target of this TFG is a plugin development that allows performing complex urban queries for the JOSM program.

This plugin allows the use of XQuery and XPath query languages to execute queries, besides including the topological operators library proposed by A. Becerra and J. Almendros.

The plugin is developed in Java using different libraries. It follows a Client-Server arquitecture, using BaseX as an XQuery server. The GUI follows the guidelines set by IPO. In addition, uses a View Model Controller pattern arquitecture.

The different stages of development that are followed in this project (planning, modeling, construction, test and documentation) explaining the developed work in each one of the stages.

The main functionalities that can be found: organize the information about the user's connection, show up a query preview, save the solution generated by the program, besides showing up the queries on a list and many other basic functionalities such as delete, create and modify queries.

It exists inside the system an explanatory tutorial guide. This tutorial shows the detailed information about its functionalities.

## Tabla de contenido

# 1. Introducción

La realización de consultas sobre mapas y operaciones topológicas, es un tema de actualidad en el mundo de la informática. Desde hace unos años, se vienen desarrollando varios proyectos que pretenden ofrecer funcionalidades a los usuarios. Entre otras, visualizar museos, restaurantes, determinar rutas óptimas cuando tenemos diferentes alternativas de paso por varios puntos de una ciudad, etc.

Dada la amplia versatilidad de este tipo de herramientas, cada vez son más usuarios los que reclaman este tipo de funcionalidades. Con la finalidad de ofrecer un amplio conjunto de las funcionalidades antes mencionadas, nace la comunidad **OpenStreetMap** (OSM) [RW 01].

Se trata de una iniciativa cuyo objetivo es crear y proporcionar datos geográficos libres (en formato **XML**), tales como callejeros y mapas de carreteras, a toda la comunidad virtual.

La comunidad OSM esta mantenida por la **OSM Foundation** (OSMF), esta es una organización internacional sin ánimo de lucro. OSMF tiene como principal objetivo sustentar y apoyar a la comunidad OSM, realiza estas labores desde el punto de vista económico, jurídico y organizativo. OSM ofrece diferentes alternativas de uso y edición de mapas espaciales.

La alternativa más sencilla es utilizar la propia interfaz web de OSM, aunque sólo permite realizar consultas básicas (i.e. valores de las etiquetas). La opción más interesante de edición y consulta de mapas espaciales es **JOSM** (**Editor Java de OSM**) [RW02].

JOSM permite realizar una edición de mapas espaciales de más alto nivel (i.e. relaciones y agrupaciones entre entidades, dibujar de forma precisa en los mapas, crear entidades espaciales) y generar consultas sobre mapas espaciales utilizando básicamente operadores booleanos sencillos sobre etiquetas o valores de etiquetas de los archivos XML que representan los mapas espaciales.

Con esta perspectiva y **con la idea de permitir consultas más complejas sobre datos espaciales, los autores A. Becerra y J. Almendros**, han propuesto un conjunto de operadores espaciales sobre mapas que representen núcleos urbanos de poblaciones [RW03].

Estos operadores espaciales están basados en los operadores topológicos definidos por Clementini [RB01], extensión del modelo topológico definido por Egenhofer [RB02].

Los operadores espaciales urbanos se han definido utilizando el lenguaje de consulta **XQuery** [RW04], diseñado para colecciones de datos XML y desarrollado por el grupo de consulta XML de **W3C**.

## 1.1 Justificación

Actualmente cada vez mas personas utilizan programas de mapas, por lo que los proyectos como OSM son cada vez mas usados por miles de personas. Existe una gran cantidad de plugins desarrollados para la plataforma JOSM, aunque la mayoría de ellos se centran en solucionar problemas de renderizado, conversión, análisis, planificación de ruta entre otros, pero el problema de realizar consultas urbanas todavía sigue sin solución. Únicamente se permite realizar consultas urbanas sencillas tales como realizar búsquedas sobre etiquetas o valores de etiqueta.

Debido a esto se ha tomado la iniciativa de desarrollar un plugin que de solución a este problema.
El trabajo se ha centrado en desarrollar un plugin para JOSM que nos permita utilizar el conjunto de operadores urbanos espaciales definidos en el estudio técnico y utilizarlos en los mapas que se visualizan usando esta herramienta.
Para ello, necesitaremos un gestor de documentos XML con un editor de XQuery. La solución por la que hemos optado ha sido **BaseX** [RW05] por su versatilidad y facilidad de uso.

El desarrollo del plugin se lleva a cabo en el lenguaje Java, siguiendo una metodología ágil, y realizando los pasos de análisis, diseño, implementación y pruebas que permitan ofrecer un producto de calidad al usuario final.

Hay dos aspectos a remarcar en el desarrollo del proyecto, dada la naturaleza del mismo:

- Por una parte, dado que se trata de un proyecto que se va a liberar, es necesario asegurar cuestiones de mantenimiento, por lo que se va implementar siguiendo el patrón de diseño **ModeloVistaControlador** (MVC) [RW06]

- Además, dado que va a ser un proyecto que contiene una interfaz gráfica, se van a utilizar las recomendaciones necesarias para garantizar una adecuada usabilidad e interacción con el usuario.

## 1.2 Objetivos

El objetivo del proyecto es la creación de un producto software, que dé solución al problema de realizar consultas complejas sobre mapas urbanos.

Dicho producto software consiste en un plugin para la plataforma JOSM, este plugin permite la ejecución de código XQuery para realizar las consultas.

- Los sub objetivos que se contemplan son los siguientes:
  - Desarrollar un proyecto que pueda ser mantenido por la comunidad JOSM.
  - Generar la documentación necesaria para contribuir con futuros desarrollos.

Los objetivos de este proyecto se han acotado debido a la gran cantidad de mejoras que se le pueden implementar.

## 1.3 Estado del arte

Actualmente, el interés sobre los mapas urbanos y la geografía está creciendo. Este crecimiento viene promovido por las nuevas tecnologías, debido a esto, cada vez más empresas y fundaciones se embarcan en proyectos para satisfacer las necesidades de los usuarios.

De estas necesidades nacen proyectos como Google Maps [RW07], creado por Google en 2005, y otras desde fundaciones sin ánimo de lucro como OSM. Además, estos nuevos proyectos no solamente nos ofrecen mapas geográficos, también nos permiten funcionalidades, tales como planificar una ruta, o indicarnos los puntos interesantes en un mapa.

Muchos proyectos nacidos desde la iniciativa privada, como Google Maps obligan a los programas que quieran utilizar sus mapas a utilizar la API de Google Maps, de forma que solamente puedes utilizar las funciones que proporciona dicha API, siendo Google el dueño de sus derechos.

Desde el punto de vista de OSM, la comunidad se ha esforzado por proporcionar la API de OpenStreetMap, la cual dispone del código abierto y permite que personas de todo el mundo puedan desarrollarla en común.

En estos momentos se están desarrollando muchas herramientas para OSM, las principales tareas que las herramientas son capaces de llevar a cabo son edición, exportación, renderizado, conversión, análisis, planificación de ruta, navegación, pero se le ha prestado muy poca atención a las consultas.

Las consultas se pueden ver desde muchos puntos de vista, desde planificar una ruta para llegar a un lugar determinado, hasta hacer cálculos estadísticos sobre un mapa.

La mayoría de las herramientas son capaces de consultar OSM con comandos muy simples: buscando por etiquetas y nombres relacionados.

También existen otras como la API extendida de OSM que ofrece búsquedas de consultas en OSM con XPath [RW08]. La API Overpass o OSM3S [RW09] es una extensión que selecciona partes en un mapa de datos OSM, esta actúa como una base de datos sobre la web, en la

que el cliente envía una consulta a la API y recibe el dato que corresponde a la consulta.

Como lenguajes de consulta esta XQuery, este es un lenguaje de programación propuesto por la W3C [RW10] como estándar para el manejo de documentos XML [RW11] . Es un lenguaje funcional donde las expresiones FLOWR (FOR-LET-WHERE-ORDER BY-RETURN) son capaces de cruzar documentos XML. XQuery tiene un sublenguaje, llamado XPath, cuyo papel es añadir nodos al mapa XML. XPath es propiamente un lenguaje de consultas equipado con condiciones booleanas y varios operadores.

GQuery es una opción para añadir operadores espaciales a XQuery. XQuery lleva a cabo la manipulación de árboles y subárboles, mientras que el procesamiento espacial es llevado a cabo utilizando funciones geométricas que usan JTS.

La aproximación GeoXQuery [RW12] amplía el proceso Saxon XQuery [RW13] con librerías funcionales que proporcionan operaciones geoespaciales. GeoXQuery también está basado en Java Topology Suite [RW14] y proporciona una librería de transformación de Graph Modelling Language (GML) [RW15] para los procesos XQuery, y para poder mostrar los resultados de las consultas. Existen varios programas que permiten ejecutar sentencias en XQuery tales como BaseX (licencia BSD), SAXON XSLT (licencia OSS) [RW16], Oracle XML DB [RW17] (licencia comercial), entre otros.

El tratamiento de los datos espaciales puede ser realizado por sistemas de gestión de bases de datos relacionales (RDBMS) como: SpatialSQL [RW18] , GeoSQL, Oracle Spatial y PostGIS [RW19]. Estos sistemas están basados en extensiones del modelo relacional para almacenar objetos

y extensiones del lenguaje de consultas SQL para la recuperación de consultas espaciales.

# 2. Marco Teórico

## 2.1 Lenguajes de consulta en XML

Actualmente utilizamos, almacenamos, representamos, gran cantidad de información en páginas web (XHTML), mensajes web entre aplicaciones (SOAP), libros y artículos, datos de negocio, representación de bases de datos relacionales, interfaces de usuario (XUL), transacciones financieras y bancarias, partidas de ajedrez, recetas de cocina, trazas del sistema, gráficos vectoriales (SVG), archivos de configuración, documentos de texto (OpenOffice.org [RW20]) y se realiza bajo uno de los formatos más conocidos XML (Extensible Markup Language) [RW21].

XML es una cadena de texto donde cada uno de los datos está delimitado por etiquetas de la forma <T> ... </T> o se incluye como atributo de una etiqueta de la forma <T A="..."> ... </T>. Mediante esta representación es posible expresar también la semántica de cada uno de los datos, estructurando las etiquetas en forma de árbol.

XML se ha convertido en una herramienta de uso cotidiano en los entornos de tratamiento de información y de programación, sin embargo, a medida que aumenta su complejidad, un árbol con un conjunto de datos en XML, no es la herramienta adecuada para manejar grandes y complejas colecciones de datos en XML.

El principal problema a la hora de procesar colecciones de datos en XML a bajo nivel es la necesidad de escribir una gran cantidad de código para realizar el procesamiento, complicándose a medida q la consulta se amplia, necesitando un mayor número de almacenamientos temporales de datos, más si añadimos el problema

de escribir desde el principio, el código para cada vez q la consulta sufra un cambio.

El código resultante es muy poco reutilizable añade complejidad y la exigencia de tener que cargar los datos en memoria antes de comenzar el recorrido, lo cual puede resultar imposible para grandes volúmenes de datos.

Para apaliar esta situación encontramos XQuery un lenguaje declarativo, que permite definir de forma rápida consultas o recorridos complejos sobre colecciones de datos en XML los cuales devuelvan todos los nodos que cumplan ciertas condiciones.

Por último indicar que el XQuery añade expresividad al XPath proporcionando mecanismos para fusionar varios documentos XML.

XQuery tiene un sublenguaje llamado XPath, cuyo papel es añadir nodos al mapa XML. XPath es un lenguaje de consultas equipado con condiciones booleanas y varios operadores.

### 2.1.1 XPath

#### *2.1.1.1 Definición*


Es un lenguaje que permite seleccionar nodos de un documento XML y calcular valores a partir de su contenido. Forma junto con XSLT [RW22] y XSL-FO [RW23] una familia de lenguajes llamada XSL, diseñada para acceder, trasformar y dar formato de salida a documentos XML, proporcionando facilidades básicas para manipulación de cadenas, números y booleanos.

XPath está diseñado a través de un subconjunto natural para comprobar si un nodo encaja con un patrón o no, siendo su construcción sintáctica básica la expresión.

Las expresiones son evaluadas para producir un objeto, que tendrá uno de los siguientes cuatro tipos básicos:

- Conjunto de nodos (colección desordenada de nodos sin duplicados)
- Booleano (verdadero o falso)
- Número (un número en punto flotante)
- Cadena (una secuencia de caracteres UCS)

La evaluación de expresiones tiene lugar respecto a un contexto. El contexto consiste en:

- Un nodo (el nodo contextual ).
- un par de enteros positivos no nulos ( la posición contextual y el tamaño contextual).
- un conjunto de asignaciones de variables.
- una biblioteca de funciones.

- el conjunto de declaraciones de espacios de nombres aplicables a la expresión.
- La posición contextual es siempre menor o igual que el tamaño contextual.
- El valor de una variable es un objeto, que puede ser de cualquiera de los tipos posibles para el valor de una expresión y puede también ser de tipos adicionales.
- La biblioteca de funciones consiste en una correspondencia de nombres de funciones a funciones. Cada función toma cero o más argumentos y devuelve un único resultado.
- Un tipo importante de expresión es el camino de localización.
- Un camino de localización selecciona un conjunto de nodos relativo al nodo de contexto.

El resultado de evaluar una expresión que sea un camino de localización es el conjunto de nodos seleccionados por el camino de localización.

Los caminos de localización pueden contener recursivamente expresiones utilizadas para filtrar conjuntos de nodos.

Las expresiones se analizan dividiendo primero la cadena de caracteres a analizar en "tokens" (caracteres que tiene un significado coherente) y a continuación analizando la secuencia de tokens resultante. Se puede usar libremente espacio en blanco entre tokens.

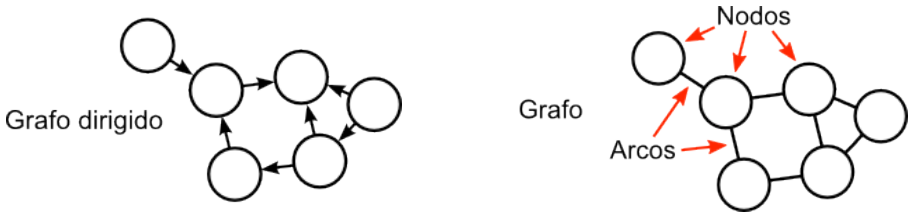
### *2.1.1.2 Características generales*

#### Árbol de nodos

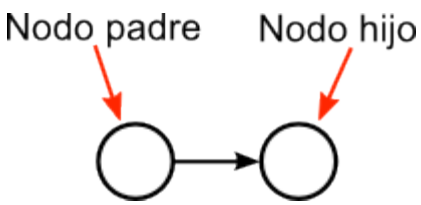
XPath considera un documento XML como un árbol de nodos. Un árbol es una estructura de datos que equivale a un árbol matemático. En Matemáticas un árbol es un caso particular de grafo. Hay diferentes tipos de nodos, incluyendo nodos elemento, nodos atributo y nodos texto.

Los siguientes términos definidos en teoría de grafos, se utilizan también en Informática y en XPath:

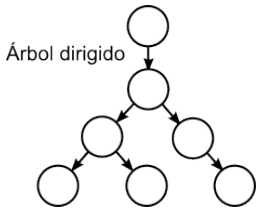
- Un **grafo** es un conjunto de objetos llamados nodos o vértices unidos por enlaces llamados arcos o aristas. Un **grafo dirigido** es un grafo en el que los arcos tienen dirección.

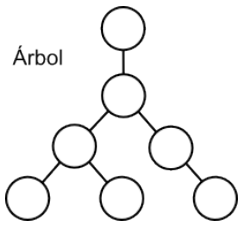



- Cuando dos nodos están unidos por un arco con dirección, el **nodo padre** es el nodo del que parte el arco y el **nodo hijo** es el nodo al que llega el arco.

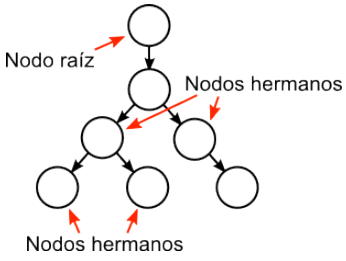


- Un **árbol** es un grafo en el que cualquier pareja de vértices están conectada por un único camino (es decir, que no hay ciclos). Un **árbol dirigido** es un árbol en el que las aristas tienen dirección y todos los nodos menos uno tienen un único padre.

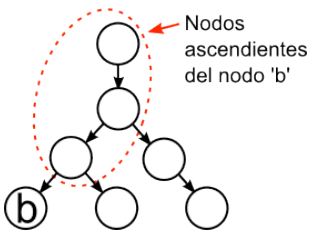


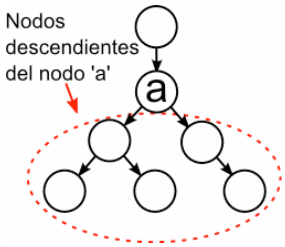




- El **Nodo Raíz** de un árbol dirigido es el único nodo sin padre. Los **nodos hermanos** son los nodos que tienen el mismo padre.




- Los **nodos descendientes** de un nodo son todos los nodos a los que se llega desde el nodo: los hijos, los hijos de los hijos, etc. Los **nodos ascendientes** de un nodo son todos los nodos de los que un nodo es descendiente: el padre, el padre del padre, etc.

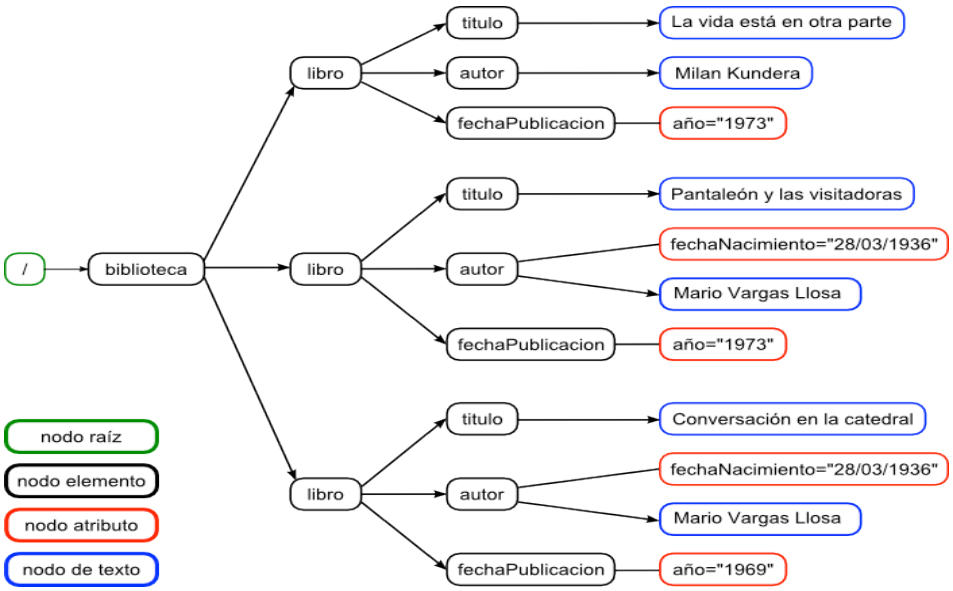


## Tipos de nodos en XPath

Un documento XML puede representarse como un árbol dirigido, considerando por ejemplo los elementos como nodos y que un elemento es padre de los elementos que contiene. Pero en XPath no sólo los elementos son nodos, en realidad hay siete tipos de nodos:

- Raíz
- Elemento
- Atributo
- Texto
- Comentario
- Instrucción de procesamiento
- Espacio de nombres




- Los nodos atributos y de texto no son como los nodos elemento.
- Los nodos atributo y de texto no pueden tener descendientes. El nodo atributo ni siquiera se considera como hijo, sino como una etiqueta adosada al elemento.

- El texto contenido por una etiqueta sí que se considera hijo del elemento, aunque las expresiones XPath suelen trabajar con nodos elementos y para referirse a los atributos o al texto se utilizan notaciones especiales.

Para cada tipo de nodo, hay una forma de determinar un **valor de cadena**. Para unos, el valor de cadena es parte del nodo; para otros, el valor de cadena se calcula a partir del valor de cadena de nodos descendientes.

El **orden de documento**, definido sobre todos los nodos del documento corresponde con el orden del primer carácter de la representación XML

- El nodo raíz será el primer nodo.
- Los nodos elemento aparecen antes de sus hijos.
- El orden de documento ordena los nodos elemento, en el orden de aparición de sus etiquetas de apertura en el XML (tras la expansión de entidades).
- Los nodos atributo y los nodos espacio de nombres de un elemento aparecen antes que los hijos del elemento.
- Los nodos espacio de nombres aparecen por definición antes que los nodos atributo.
- El orden relativo de los nodos espacio de nombres depende de la implementación.
- El **Orden inverso de documento** es el inverso del orden de documento.
- Los nodos raíz y los nodos elemento tienen una lista ordenada de nodos hijo.
- Los nodos nunca comparten un hijo: si un nodo no es el mismo nodo que otro, entonces ninguno de los hijos del primer nodo será el mismo nodo que ninguno de los hijos del otro nodo.
- Todos los nodos salvo el nodo raíz tienen exactamente un **padre**, que es o bien un nodo elemento o bien el nodo raíz.

- Un nodo raíz o un nodo elemento es el padre de cada uno de sus nodos hijo.
- Los **descendientes** de un nodo son los hijos del nodo y los descendientes de los hijos del nodo.

## Nodos Raíz

- El nodo raíz es la raíz del árbol.
- El nodo elemento del elemento de documento es hijo del nodo raíz.
- El nodo raíz tiene también como hijos los nodos instrucción de procesamiento y comentario correspondientes a las instrucciones de procesamiento y comentarios que aparezcan en el prólogo y tras el fin del elemento de documento.
- El valor de cadena del nodo raíz es la concatenación de los valores de cadena de todos los nodos texto descendientes del nodo raíz en orden de documento.
- El nodo raíz no tiene nombre expandido.

### Nodos elemento

Los nodos elemento tienen un nombre expandido calculado expandiendo el QName [RW24] del elemento especificado en la etiqueta de acuerdo con la Recomendación de Espacios de Nombres XML [XML Names] [RW25].

El URI [RW26] de espacio de nombres del nombre expandido del elemento será nulo si el QName no tiene prefijo y no hay un espacio de nombres por defecto aplicable.

- Hay un nodo elemento por cada elemento en el documento.

- Los hijos de un nodo elemento son los nodos elemento, nodos comentario, nodos instrucción de procesamiento y nodos texto que contiene.
- Las referencias de entidades tanto a entidades internas como externas son expandidas.
- Las referencias de caracteres son resueltas.
- El valor de cadena de un nodo elemento es la concatenación de los valores de cadena de todos los nodos texto descendientes del nodo elemento en orden de documento.

### Identificadores únicos

Los nodos elemento pueden tener un identificador único (ID). Este es el valor del atributo que se declara en el DTD [RW27] como de tipo ID.

No puede haber dos elementos en un documento con el mismo ID.

Si un procesador XML informa de la existencia de dos elementos de un documento con el mismo ID (solo posible si el documento es no valido) entonces el segundo elemento en orden de documento debe ser tratado como si no tuviese ID.

Si un documento no tiene DTD, entonces ningún elemento del documento tendrá ID.

### Nodos atributo

Cada nodo elemento tiene asociado un conjunto de nodos atributo, donde el elemento, es el padre de cada uno de esos nodos atributo y los nodos atributo no son hijos de su elemento padre.

Los elementos nunca comparten nodos atributo: Si dos nodos elemento son distintos, entonces ninguno de los nodos atributo del primer

elemento será el mismo nodo que ningún nodo atributo del otro nodo elemento.

**El operador =** comprueba si dos nodos tienen el mismo valor, *no* si son el mismo nodo. El atributos de dos elementos diferentes pueden ser comparados como iguales utilizando =, a pesar de que no son el mismo nodo.

Un atributo tomado por defecto se trata igual que uno especificado. Algunos atributos, tal como xml:lang [RW28] y xml:space [RW29], tienen la semántica de ser aplicables a todos los elementos que sean descendientes del elemento que lleva el atributo, salvo que sean redefinidos por una instancia del mismo atributo en otro elemento descendiente.

Los nodos atributo tienen un nombre expandido y un valor de cadena. El nombre expandido se calcula expandiendo el QName especificado en la etiqueta en el documento XML.

Los nodos atributo tienen un valor de cadena. El valor de cadena es el valor normalizado tal como se especifica en la Recomendación XML [RW30]. Un atributo cuyo valor normalizado es una cadena de longitud cero no es tratado de una forma especial: da lugar a un nodo atributo cuyo valor de cadena es una cadena de longitud cero.

### Nodos espacio de nombres

Cada elemento tiene un conjunto asociado de nodos espacio de nombres, uno para cada uno de los distintos prefijos de espacio de nombres con efecto sobre el elemento (incluyendo el prefijo xml, que está implícitamente declarado según la Recomendación de Espacios de Nombres XML siendo las principales:

- El elemento es el padre de cada uno de los nodos espacio de nombres
- Los nodos espacio de nombres no son hijos de su elemento padre.
- Los elementos nunca comparten nodos espacio de nombres
- Si dos nodos elemento son distintos, entonces ninguno de los nodos espacio de nombres del primer elemento será el mismo nodo que ningún nodo espacio de nombres del otro nodo elemento. Esto significa que los elementos tendrán un nodo espacio de nombres
- Los nodos espacio de nombres tienen un nombre expandido: la parte local es el prefijo de espacio de nombres (este es vacío si el nodo espacio de nombres corresponde al espacio de nombres por defecto); el URI de espacio de nombres es siempre nulo.
- El valor de cadena de un nodo espacio de nombres es el URI de espacio de nombres que se está asociando al prefijo de espacio
- Una expresión XPath que dependa del valor de cadena de uno de tales nodos de espacio de nombres no es interoperable.

### Nodos instrucción de procesamiento

Hay un nodo instrucción de procesamiento para cada instrucción de procesamiento, salvo para aquellas que aparezcan dentro de la declaración de tipo de documento.

Las instrucciones de procesamiento tienen un nombre expandido: la parte local es el destinatario de la instrucción de procesamiento; el URI de espacio de nombres es nulo.

El valor de cadena de un nodo instrucción de procesamiento es la parte de la instrucción de procesamiento que sigue al destinatario y todo el espacio en blanco. No incluye la terminación ?>.

### Nodos comentario

Hay un nodo comentario para cada comentario, salvo para aquellos que aparezcan dentro de la declaración de tipo de documento.

El valor de cadena de los comentarios es el contenido del comentario sin incluir el inicio <!-- ni la terminación -->.

### Nodos texto

Los datos de carácter se agrupan en nodos texto. En cada nodo texto se agrupan todos los datos de carácter que sea posible, por lo que un nodo texto nunca tiene un hermano inmediatamente anterior o siguiente que sea nodo texto.

El valor de cadena de los nodos texto son los datos de carácter. Los nodos texto siempre tienen al menos un carácter.

Los caracteres dentro de comentarios, instrucciones de procesamiento y valores de atributos no producen nodos texto.

El espacio en blanco fuera del elemento de documento no produce nodos de texto.

### Caminos de Localización [LocationPath]

Todo camino de localización [RW35] se puede expresar utilizando una sintaxis directa. Hay también ciertas abreviaturas sintácticas que permiten expresar casos frecuentes con concisión. La semántica de los caminos de localización utilizan la sintaxis no abreviada.

Ejemplos de caminos de localización utilizando la sintaxis no abreviada:

- **child::** para selecciona los elementos para hijos del nodo contextual
- **child::*** selecciona todos los elementos hijos del nodo contextual
- **child::text()** selecciona todos los nodos texto hijos del nodo contextual
- **child::node()** selecciona todos los hijos del nodo contextual, cualquiera que sea su tipo de nodo
- **attribute::name** selecciona el atributo name del nodo contextual
- **attribute::*** selecciona todos los atributos del nodo contextual
- **descendant::**para selecciona los elementos para descendientes del nodo contextual
- **descendant-or-self::**para selecciona los elementos para descendientes del nodo contextual y, si el nodo contextual es un elemento para , el nodo contextual también
- **self::**para selecciona el nodo contextual si este es un elemento para , en otro caso no selecciona nada
- **child::chapter/descendant::**para selecciona los elementos para descendientes de los elementos chapter hijos del nodo contextual
- **child::*/child::**para selecciona todos los nietos para del nodo contextual
- / selecciona la raíz del documento (que es siempre el padre del elemento de documento)

### Tipos de caminos

Hay dos tipos de caminos de localización: caminos de localización relativos y caminos de localización absolutos.

Un camino **de localización relativo** consiste en una secuencia de uno o más pasos de localización separados por /.

... Los pasos en un camino de localización relativo se componen de izquierda a derecha.

- Cada paso selecciona un conjunto de nodos relativos a un nodo contextual.
- La secuencia inicial de pasos selecciona un conjunto de nodos relativos a un nodo de contexto.
- Cada nodo de ese conjunto se usa como nodo de contexto para el siguiente paso.
- Los distintos conjuntos de nodos identificados por ese paso se unen.
- El conjunto de nodos identificado por la composición de pasos es dicha unión.

Un camino de **localización absoluto** consiste en / seguido opcionalmente por un camino de localización relativo.

- Una / por si misma selecciona el nodo raíz del documento que contiene al nodo contextual.
- Si es seguida por un camino de localización relativo, entonces el camino de localización selecciona el conjunto de nodos que seleccionaría el camino de localización relativo al nodo raíz del documento que contiene al nodo contextual.

***Location Paths***

**LocationPath** ::= **RelativeLocationPath**
| **AbsoluteLocationPath**

**AbsoluteLocationPath** ::= '/' **RelativeLocationPath** ?
| **AbbreviatedAbsoluteLocationPath**

**RelativeLocationPath** ::= **Step**

| RelativeLocationPath '/' Step

| AbbreviatedRelativeLocationPath

### Pasos de localización

Un paso de localización tiene tres **partes:**

- Un eje, que especifica la relación jerárquica entre los nodos seleccionados por el paso de localización y el nodo contextual.
- Una prueba de nodo, que especifica el tipo de nodo y el nombre-expandido de los nodos seleccionados por el paso de localización.
- Cero o más predicados, que usan expresiones arbitrarias para refinar aún más el conjunto de nodos seleccionado por el paso de localización.

La **sintaxis** del paso de localización es:

- … el nombre de eje y prueba de nodo.
- … separados por dos caracteres de dos puntos.
- … seguido de cero o más expresiones.
- … cada una entre paréntesis cuadrados.

… **Location Steps**

**Step** ::= **AxisSpecifier *NodeTestPredicate* ***

| **AbbreviatedStep**

**AxisSpecifier** ::= **AxisName** '::'

| **AbbreviatedAxisSpecifier**

### Ejes

Se describen algunos de los ejes [RW36] disponibles:

- El **eje child** contiene los hijos del nodo contextual
- El **eje descendant** contiene los descendientes del nodo contextual; un descendiente es un hijo o el hijo de un hijo, etc; de forma que nunca contiene nodos atributo o espacio de nombres
- El **eje parent** contiene el padre del nodo contextual, si lo hay
- El **eje ancestor** contiene los ancestros del nodo contextual; en el padre del nodo contextual y el padre del padre, etc, incluyendo siempre al nodo raíz, salvo que el nodo contextual sea el nodo raíz
- El **eje following-sibling** contiene todos los siguientes hermanos del nodo contextual; si el nodo contextual es un nodo atributo o un nodo espacio de nombres, el eje following-sibling está vacío
- El **eje preceding-sibling** contiene todos los hermanos precedentes del nodo contextual; si el nodo contextual es un nodo atributo o un nodo espacio de nombres, el eje preceding-sibling está vacío
- **El eje following** contiene todos los nodos del mismo documento que el nodo contextual que están después de este según el orden del documento, excluyendo los descendientes y excluyendo nodos atributo y nodos espacio de nombres
- El **eje preceding** contiene todos los nodos del mismo documento que el nodo contextual que están antes de este según el orden del documento, excluyendo los ancestros y excluyendo nodos atributo y nodos espacio de nombres
- El **eje attribute** contiene los atributos del nodo contextual; el eje estará vacío a no ser que el nodo contextual sea un elemento

- El **eje namespace** contiene los nodos espacio de nombres del nodo contextual; el eje estará vacío a no ser que el nodo contextual sea un elemento
- **El eje self** contiene simplemente el propio nodo contextual
- El **eje descendant-or-self** contiene el nodo contextual y sus descendientes
- El **eje ancestor-or-self** contiene el nodo contextual y sus ancestros; así, el eje ancestor-or-self siempre incluirá el nodo raíz

```
Axe        AxisName  ::=  'ancestor'
                          | 'ancestor-or-self'
                          | 'attribute'
                          | 'child'
                          | 'descendant'
                          | 'descendant-or-self'
                          | 'following'
                          | 'following-sibling'
                          | 'namespace'
                          | 'parent'
                          | 'preceding'
                          | 'preceding-sibling'
                          | 'self'
```

## Predicados

Los ejes están orientados hacia adelante o hacia atrás.

- Un eje que sólo puede contener el nodo contextual o nodos que están a continuación del nodo contextual según el orden de documento es un **eje hacia adelante**.
- Un eje que sólo puede contener el nodo contextual o nodos que están antes del nodo contextual según el orden de documento es un **eje hacia atrás**.

Los ejes ancestor, ancestor-or-self, preceding, y preceding-sibling son ejes hacia atrás; todos los demás ejes son hacia adelante.

La **posición de proximidad** de un miembro de un conjunto de nodos con respecto a un eje se define como la posición del nodo en el conjunto ordenado según el orden de documento si el eje es hacia adelante y según el orden inverso de documento si el eje es hacia atrás. La primera posición es 1.

Un predicado filtra un conjunto de nodos con respecto a un eje para producir un nuevo conjunto de nodos. Por cada nodo en el conjunto de nodos a filtrar, la PredicateExpr es evaluada con dicho nodo como nodo contextual, con el número de nodos en el conjunto de nodos como tamaño contextual, y con la posición de proximidad del nodo en el conjunto de nodos respecto al eje como posición contextual; si PredicateExpr se evalúa como verdadera para ese nodo, el nodo se incluye en el nuevo conjunto de nodos; en otro caso, no se incluye.

Un PredicateExpr se evalúa evaluando la Expr y convirtiendo el resultado en un booleano. Si el resultado es un número, se convertirá en verdadero si el número es igual a la posición contextual y se convertirá en falso en cualquier otro caso.

Si el resultado no es un número, el resultado se convertirá igual que con una llamada a la función boolean. Así un camino de localización para[3] es equivalente a para[position()=3] .

***Predicates***

**Predicate ::= '[' PredicateExpr ']'**

**PredicateExpr ::= Expr**

## Booleanos

Un objeto de tipo booleano puede tener uno de dos valores, verdadero o falso.

- Una expresión **or** se evalúa para cada operando y convirtiendo su valor en booleano como si se aplicase la función boolean. El resultado es verdadero si alguno de los dos valores es verdadero y falso en otro caso. El operando de la derecha no se evalúa si el operando de la izquierda se evalúa como verdadero.
- Una expresión **and** se evalúa para cada operando y convirtiendo su valor en booleano como si se aplicase la función boolean. El resultado es verdadero si ambos valores son verdaderos y falso en otro caso. El operando de la derecha no se evalúa si el operando de la izquierda se evalúa como falso.

Las comparaciones que involucran conjuntos de nodos se definen en términos de comparaciones que no los involucran; esto se define uniformemente para =, !=, <=,< , >= y >. Las comparaciones que no involucran conjuntos de nodos se definen para = y !=. Las comparaciones que no involucran conjuntos de nodos se definen para <= ,<, >= y >.

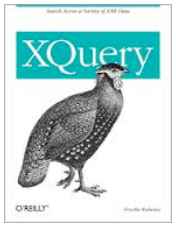
### *2.1.1.3 Estructura Léxica*

En la "tokenización", siempre se devuelve el token más largo posible. Para facilitar la lectura, se pueden utilizar espacios en blanco en las expresiones aunque no estén explícitamente permitidos por la gramática. Se puede añadir libremente ExprWhitespace [RW40] en las expresiones antes o después de cualquier ExprToken .

Las siguientes reglas especiales de "tokenización" se deben aplicar en el orden especificado para romper la ambigüedad de la gramática de la expresión ExprToken :

- Si existe un token anterior y este no es @, ::, ( ,[, , o un Operator , entonces un * se deberá reconocer como un MultiplyOperator [RW41] y un NCName [RW42] se deberá reconocer como un OperatorName [RW43].
- Si el carácter que sigue a un QName (quizá tras la interposición de ExprWhitespace [RW44]) es (, entonces el token se deberá reconocer como un NodeType [RW45] o un FunctionName [RW46].
- Si los dos caracteres siguientes a un NCName (quizá tras la interposición de ExprWhitespace ) son ::, entonces el token se deberá reconocer como un AxisName [RW47].
- En otro caso, el token no se deberá reconocer como un MultiplyOperator, un OperatorName , un NodeType, un FunctionName , o un AxisName.

### 2.1.2 XQuery




#### *2.1.2.1 Definición*

XQuery [RW48] es un lenguaje de consulta diseñado para escribir consultas sobre colecciones de datos expresadas en XML. Abarca desde archivos XML hasta bases de datos relacionales con funciones de conversión de registros a XML.

Su principal función es extraer información de un conjunto de datos organizados como un árbol de etiquetas XML.

- XQuery es independiente del origen de los datos.
- XQuery es un lenguaje funcional, en vez de ejecutar una lista de comandos, cada consulta es una expresión que es evaluada y devuelve un resultado.
- Combina de manera flexible diversas expresiones para crear nuevas expresiones más complejas y de mayor potencia semántica.

XQuery es el estándar de consultas sobre documentos XML. De manera rápida podemos definir XQuery con un símil en el que XQuery es a XML lo mismo que SQL es a las bases de datos relacionales. Ha sido construido sobre la base de XPath, un lenguaje declarativo para la localización de nodos y fragmentos de información en árboles XML.

XQuery se basa en este lenguaje para realizar la selección de información y la iteración a través del conjunto de datos.

Es un lenguaje declarativo. Al igual que SQL [RW49] hay que indicar que se quiere, no la manera de obtenerlo. Independiente del protocolo de acceso a la colección de datos. Una consulta en XQuery debe funcionar igual al consultar un archivo local que al consultar un servidor de bases de datos que al consultar un archivo XML en un servidor web.

- Las consultas y los resultados han de respetar el modelo de datos XML.

- Trabaja con independencia de la estructura del documento, sin necesidad de conocerla.

- Soportar tipos simples, como enteros y cadenas, y tipos complejos, como un nodo compuesto por varios nodos hijos.

- Las consultan deben soportar cuantificadores universales (para todo) y existenciales (existe).

También, las consultas soportan operaciones sobre jerarquías y secuencias de nodos .Una consulta puede combinar información de múltiples fuentes. Las consultas son capaces de manipular los datos independientemente del origen de estos.

Mediante XQuery se definen consultas que transforman las estructuras de información originales y se pueden crear nuevas estructuras de datos.

### *2.1.2.2 Características generales*

**Principales operaciones de consultas** **For, Let, Where, Order y Return (FLWOR)**

Una consulta en XQuery es una expresión que lee una secuencia de datos en XML y devuelve como resultado otra secuencia de datos en XML.

Las expresiones y los valores que devuelven son dependientes del contexto. Por ejemplo, los nodos que aparecerán en el resultado dependen de los namespaces, de la posición, donde aparezca la etiqueta raíz del nodo (dentro de otra, por ejemplo), etc.

En XQuery las consultas pueden estar compuestas por cláusulas de hasta cinco tipos distintos.

Las consultas siguen la **norma FLWOR** (leído como flower), siendo FLWOR las siglas de For, Let, Where, Order y Return.

**----------For----------**

Vincula una o más variables a expresiones escritas en XPath, creando un flujo de tuplas en el que cada tupla está vinculada a una de las variable.

**----------Let----------**

Vincula una variable al resultado completo de una expresión añadiendo esos vínculos a las tuplas generadas por una cláusula for o, si no existe ninguna cláusula for, creando una única tupla que contenga esos vínculos.

**----------Where---------**

Filtra las tuplas eliminando todos los valores que no cumplan las condiciones dadas.

**----------Order by-------**

Ordena las tuplas según el criterio dado.

**----------Return-----------**

Construye el resultado de la consulta para una tupla dada, después de haber sido filtrada por la cláusula where y ordenada por la cláusula order by.

- En XQuery, cuando usamos el térmico tupla, nos estamos refiriendo a cada uno de los valores que toma una variable:

  - For y Let ….. Crean las tuplas
  - Where ….. Filtra las tuplas
  - Order by …. Ordena las tuplas
  - Return …. Trasforma las tuplas

### *2.1.2.3 Estructura léxica*

**Reglas de obligado cumplimiento en cualquier consulta.**

- For y let sirven para crear las tuplas con las que trabajará el resto de las cláusulas de la consulta.

- For y let pueden usarse tantas veces como se desee en una consulta, incluso dentro de otras cláusulas.

- Solo puede declararse una única cláusula where.
- Solo puede declararse una única cláusula order by.
- Solo puede declararse una única cláusula return.
- Ninguna de las cláusulas FLWOR es obligatoria en una consulta.
- Es posible especificar varios criterios de ordenación en la cláusula order by, separándolos por comas.
- Los criterios de ordenación se aplican por orden de izquierda a derecha.

### Diferencias entre las cláusulas for y let.

- La cláusula **for** vincula una variable con cada nodo que encuentre en la colección de datos.
- La cláusula **let**, en cambio, vincula una variable con todo el resultado de una expresión.

Si una cláusula let aparece en una consulta que ya posee una o más cláusulas for, los valores de la variable vinculada por la cláusula let se añaden a cada una de las tuplas generadas por la cláusula for.

Si en la consulta aparece más de una cláusula for (o más de una variable en una cláusula for), el resultado es el producto cartesiano de dichas variables, es decir, las tuplas generadas cubren todas las posibles combinaciones de los nodos de dichas variables.

## Principales aplicaciones

XQuery es una tecnología emergente con grandes expectativas en el mundo de la programación, del tratamiento y manipulación de información.

Sus principales aplicaciones se pueden resumir en tres grandes grupos:

- En primer lugar, **recuperar información** a partir de conjuntos de datos XML.

- Mediante un lenguaje sencillo, potente y flexible, es posible recorrer los nodos de un conjunto de datos XML, filtrando aquellos que nos interesen y transformándolos para mostrar la información deseada con la estructura adecuada.

- En segundo lugar, **transformar unas estructuras de datos** XML en otras estructuras que organicen la información de forma diferente, siendo sencillo y fácil obtener a partir de la jerarquía de datos almacenada.

- Y en tercer lugar, ofrecer una alternativa a XSLT para realizar **transformaciones de datos** en XML a otro tipo de representaciones, como HTML o PDF.

- Resulta bastante sencillo escribir una consulta XQuery para que genere código HMTL.

## Ejemplos de utilización de XQuery

En este punto se describe un ejemplo de utilización de la clausula Where, dicha clausula se probara en un archivo xml llamado libros.xml.

```
for $b in doc("libros.xml")//libro
return
        <libro>
                { $b/titulo }
                {
                        for $a at $i in $b/autor
                        where $i <= 2
                        return <autor>{string($a/last), '', '',
                        string($a/first)}</autor>
                }
                {
                        if (count($b/autor) > 2)
                        then <autor>et al.</autor>
                        else ()
                }
        </libro>
```

El resultado de esta consulta se sería el siguiente.

```
<libro>
        <titulo>TCP/IP Illustrated</titulo>
        <autor>Stevens, W.</autor>
</libro>
<libro>
        <titulo>Advanced Programming in the Unix Environment</titulo>
        <autor>Stevens, W.</autor>
</libro>
```

```
<libro>
        <titulo>Data on the Web</titulo>
        <autor>Abiteboul, Serge</autor>
        <autor>Buneman, Peter</autor>
        <autor>et al.</autor>
</libro>
```

La cláusula where de una consulta permite establecer el ámbito filtrando las tuplas que aparecerán en el resultado.
Por otro lado una expresión condicional nos permitiría crear una u otra estructura de nodos dependiendo de los valores que hemos filtrado en el resultado.

En XQuery a diferencia de la mayoría de los lenguajes, la cláusula else es obligatoria y debe aparecer siempre en la expresión condicional.
Por este motivo toda expresión en XQuery debe devolver un valor y si no existe ningún valor a devolver al no cumplirse la cláusula if, devolvemos una secuencia vacía con 'else ()'.

En este otro ejemplo se haría va a poner una consulta que utilice los operadores urbanos propuesto por A.Becerra y J.Almendros, los cuales serán explicados en el punto 2.3, esta consulta estaría tal y como la podríamos usar en el programa.

```
Import module namespace osm = "osm" at
"osmXQueryLibraryDataBase.xqy";

import module namespace osm_database = "osm_database" at
"osmDataBaseTransformationLibrary.xqy";

let $tags := <tags>
<tag>osm:AllCrossing</tag>
```

```
<tag>osm:AllEndingFrom</tag>
<tag>osm:AllEndingTo</tag>
</tags>

let $document1 := osm_database:_osm2OneWay()

let $string1 := "Calzada de Castro", $idRecursion := 0
return osm:iteratorQuery($string1, $idRecursion, $document1,$tags/tag)
```

En esta consulta nos devolvería todas las calles que acaben en Calzada de Castro, las calles en las que Calzada de castro acaba y todas las calles que se cruzan con Calzada de Castro.

## 2.2 Operadores Espaciales. Egenhofer

# Análisis Espacial

Max J. **Egenhofer**

### 2.2.1 El modelo topológico

Para realizar un análisis espacial, es necesario contar con un modelo de interacción visual que nos ayude a encontrar las relaciones existentes entre los objetos.

También hay que contar con un lenguaje, que nos permita capturar el tipo de consulta que aplicaremos sobre los datos, en un lenguaje visual, para Consultas en SIG´s [RW50].

El Modelo de relaciones topológicas binarias se define en términos de invariantes topológicas de cuatro intersecciones entre objetos espaciales homogéneos. Las relaciones son definidas de acuerdo a los resultados obtenidos, identificando ocho relaciones topológicas entre dos objetos espaciales.

Para determinar la relación entre dos objetos se debe buscar la existencia de las siguientes cuatro intersecciones, las cuales **retornan "0" cuando no existe la intersección y "¬0" cuando existe.**

1. Existen límites comunes entre los dos objetos $(\partial \cap \partial)$.

2. Existe interiores en común (°∩°).

3. Parte del límite de un objeto que coincide con parte del interior del otro objeto ($\partial \cap °$).

4. Parte del interior de un objeto coincide con el límite del otro objeto (°∩∂).

$$R_r(A.B) \Leftrightarrow I(A.B) = \begin{pmatrix} \partial \cap \partial & \partial \cap ° \\ ° \cap \partial & ° \cap °. \end{pmatrix}$$

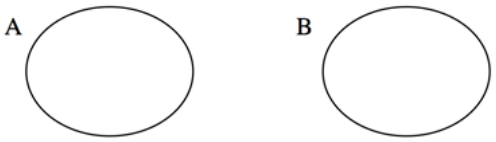
### 2.3.2 Relaciones interseccionales

Las ocho relaciones que se pueden detectar, a partir de los resultados obtenidos de estas cuatro intersecciones son:

1. **No toca a (Disjoint)**

Si las cuatro intersecciones entre todas las caras no existen, entonces las dos áreas están separadas. La representación sería:

$$R_{disjoint}(A.B) \Leftrightarrow I(A.B) = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$$

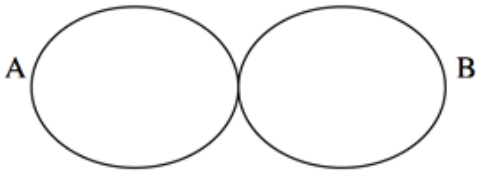



Relación "No toca a"

### 2. Toca a (Meet)

Si la intersección entre los límites de los objetos existe y las otras tres intersecciones no se cumplen entonces estos objetos se tocan:

$$R_{meet}(A.B) \Leftrightarrow I(A.B) = \begin{pmatrix} \neg 0 & 0 \\ 0 & 0 \end{pmatrix}$$

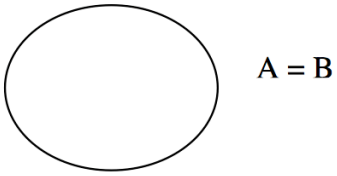



Relación "Toca a"

### 3. Igual a (Equal)

Dos regiones son iguales, si existen ambas intersecciones de límites e interiores.

$$R_{equalst}(A.B) \Leftrightarrow I(A.B) = \begin{pmatrix} \neg 0 & 0 \\ 0 & \neg 0 \end{pmatrix}$$

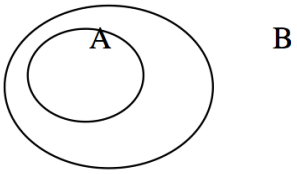



Relación "Igual a"

### 4. Dentro de (Inside)

Una región A esta dentro de otra región B si: 1) A y B comparten el mismo interior pero no los límites. 2) Si el límite de A es un subconjunto del interior de B. 3) y el límite de B no intersecta con ninguna parte del interior de A.

$$R_{inside}(A.B) \Leftrightarrow I(A.B) = \begin{pmatrix} 0 & \neg 0 \\ 0 & \neg 0 \end{pmatrix}$$

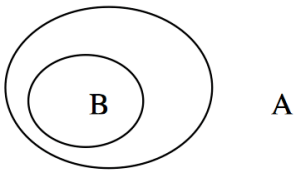



Relación "Dentro de"

### 5. Contiene a (Contain)

Una región A contiene a otra región B, ver figura 4.5. Si 1) A y B comparten el mismo interior, pero no tienen límites en común. 2) Si el límite de B es un subconjunto del interior de A, 3) el límite de A no intersecta con el interior de B.

$$R_{contain}(A.B) \Leftrightarrow I(A.B) = \begin{pmatrix} 0 & 0 \\ \neg 0 & \neg 0 \end{pmatrix}$$

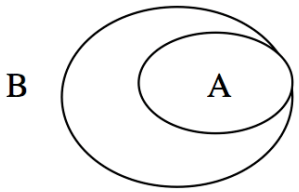



Relación "Contiene a"

### 6. Es cubierto por (Covered by)

Una región es cubierta por otra región B si ambas regiones tienen partes en común en sus límites e interiores. Si parte del interior de A intersectan con parte de los límites de B y si el interior de B intersecta con parte del límite de A.

$$R_{\text{covered by}}\ (A\,.\,B) \Leftrightarrow I(A\,.\,B) = \begin{pmatrix} \neg 0 & \neg 0 \\ 0 & \neg 0 \end{pmatrix}$$

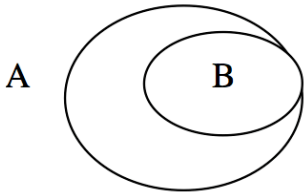



Relación "Es cubierto por"

### 7. Cubre a (Covers)

Una región A cubre a otra región B, si ambas regiones comparten el mismo límite e interior; Si el interior de B intersecta con el límite de A; y si el interior de A es parte del límite de B.

$$R_{\text{covers}}\ (A\,.\,B) \Leftrightarrow I(A\,.\,B) = \begin{pmatrix} \neg 0 & 0 \\ \neg 0 & \neg 0 \end{pmatrix}$$

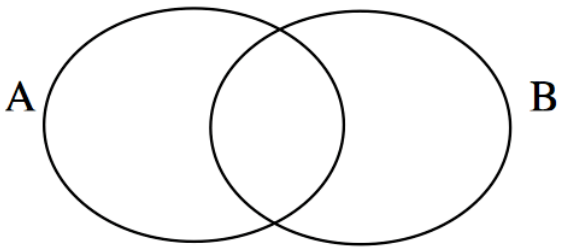



Relación "Cubre a"

### 8. Cubre parte de (Overlap)

Una región cubre parte de otra, si tienen límites e interiores en común y ambos límites intersectan con el interior del otro objeto.

$$R_{overlap}\ (A\ .\ B) \Leftrightarrow I(A\ .\ B) = \begin{pmatrix} \neg 0 & \neg 0 \\ \neg 0 & \neg 0 \end{pmatrix}$$




Relación “Cubre parte de”

Obteniendo con este enfoque la relación que existe entre dos objetos, siendo, solo empleado para obtener relaciones entre superficies.

## 2.3 Operadores Espaciales. Clementini

Eliseo Clementini

### 2.3.1 El modelo topológico

Hay dos modelos comunes en la física del espacio [RW51] :

**"Orientado a Campos y Orientada a Objetos"**

Desde una perspectiva de la base de datos, los modelos orientados a objetos son la mejor opción. Los modelos orientados a objetos tratar el espacio físico, identificable y espacialmente referenciados.

Para la integración de los aspectos geométricos, en un modelo de datos, es necesario para representar objetos espaciales (en el sentido de la aplicación) como \ objetos "(en el sentido de la DBMS) que tiene al menos un atributo de \ geométrica".

Esto significa que el modelo de datos debe apoyar, además de los tipos de datos comunes (por ejemplo, integer, float, string, etc), y tipos de datos geométricos, como Point, LineString, polígono, multipunto, MultiLineString, MultiPolygon, etc.

Su uso permite la formulación de consultas que se mezclan tanto espaciales como predicados no espaciales.

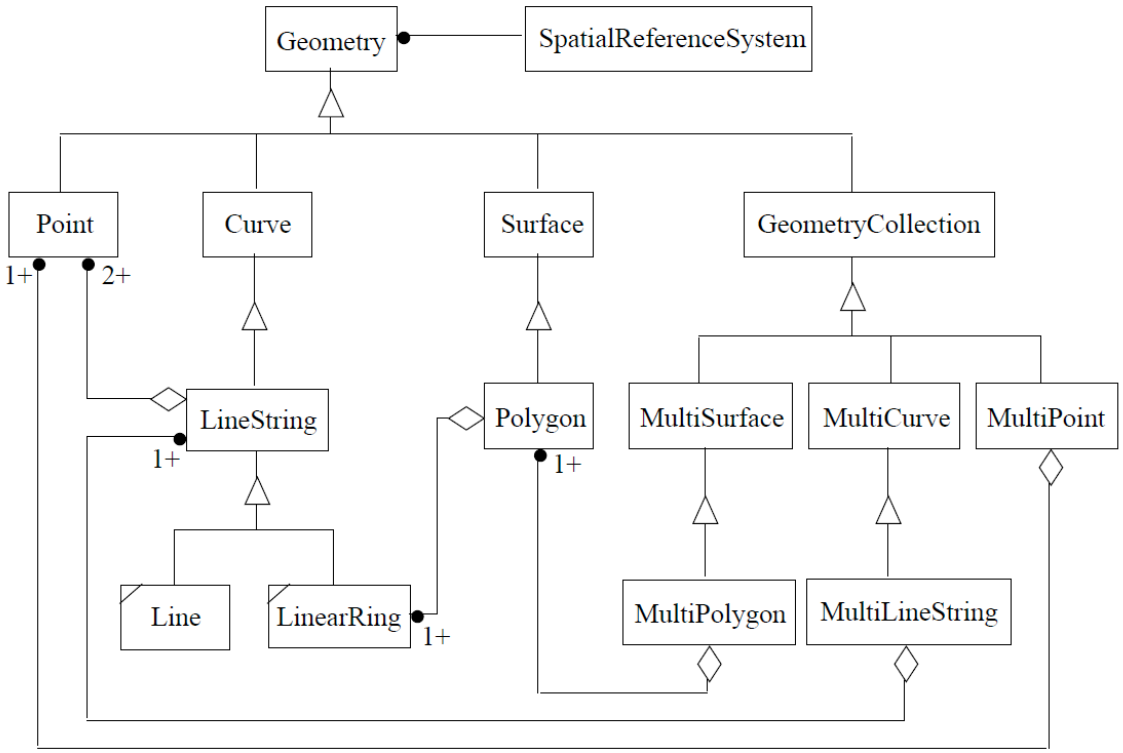
Se han usado modelos de datos espaciales a la medida y por lo tanto el límite, de los objetos espaciales se determina con precisión. Esto conduce a objetos de base de datos que tienen geometría exacta (comúnmente llamado: objetos con un límite definido o simplemente objetos nítidos).

Esta simplificación de la realidad no es aceptable en muchos casos, creando una **situación de incertidumbre** [RW52] en los datos espaciales: incompleto, inconsistencia, vaguedad, imprecisión y error.

Se necesita un nuevo modelo geométrico que gane a los modelos actuales de bases de datos espacial y que tradicionalmente son una colección de líneas (puntos, polilíneas y polígonos).

La propuesta de numerosos trabajos de investigación recientes es introducir límites amplios y sustituir los límites generales.

## Jerarquía de tipos de datos geométrica



### 2.3.5 Operadores Topologicos

Los operadores espaciales se utilizan para capturar las propiedades geométricas de los objetos en el espacio físico y las relaciones entre ellos, así como para realizar el análisis espacial.

Y que se aplican a todos los tipos de datos primitivos geométricos (es decir, los puntos, las líneas y las regiones) definiéndose en un nivel más general, mientras que otros operadores son específicos de tipos de datos derivados (por ejemplo, redes).

- El **tipo más básico** de los operadores son los operadores-orientado a establecer **(intersección, unión, diferencia**) también presentes en las primeras propuestas de lenguajes de consulta espaciales.

Los operadores espaciales de más alto nivel se basan en **tipo primarios**, que pueden aplicarse al agregar datos de tipos de redes (por ejemplo) y puede ser utilizada para realizar más tipos complejos de análisis de datos espaciales.

Se puede distinguir entre los operadores espaciales en unario y binario, que se aplican, respectivamente, para evaluar las propiedades de los objetos o las relaciones individuales entre los objetos.

En general, pueden ser estructurados a lo largo de las tres dimensiones ortogonales \ "propuesta por Clementini y Di Felice [RB03] aunque en este trabajo los operadores métricos y los proyectivos no se van a tratar, únicamente se van a tratar los operadores topológicos.

**Operadores topológicos,** a través de que la topología pueda expresar predicados acerca de la conexión, el número de componentes, la presencia / ausencia de agujeros, así como relaciones topológicas (que

describen si dos objetos intersectan o no, y, en el primer caso, ¿cómo se cruzan?).

Las propiedades topológicas son invariables a las transformaciones topológicas (1-a-1 bicontinuo funciones), son propiedades que no cambian después de transformaciones como rotación, traslación, escala.

La topología puede ser considerada el tipo más primitivo de información espacial, ya que un cambio en la topología implica un cambio en otros aspectos geométricos.

Los operadores topológicos pueden ser estructurados jerárquicamente en varios niveles, donde el nivel base corresponde a operadores capaces de detectar detalladamente las relaciones topológicas entre las regiones con un amplio límite usando el modelo de intersección y cuanto más alto es el nivel o más abstracto, los operadores permiten a los usuarios consultas espaciales inciertas de datos de forma independiente de la geometría del modelo de datos.

Se establece un marco conceptual de referencia para las propiedades topológica, proponiendo un conjunto de siete invariantes lógicas topográficas caracterizando completamente las relaciones topológicas entre los objetos binarios en el plano y las invariantes topológicas con estructuración jerárquica, aunque no son de carácter general (tales como el contenido de intersección).

- **Operadores Básicos (Basic operators)**:
  - **SpatialReference** Devuelve el sistema de referencia geométrica
  - **Envelope** operadores de sobres El rectángulo delimitador mínimo de la geometría

- **Export** Exporta la geometría introducida a una presentación diferente
- I**sEmpty** Prueba si la geometría es un conjunto vacío o no
- I**sSimple** Devuelve True si la geometría es simple
- **Boundary** Devuelve el perímetro de la geometría

## Operadores topológicos (Topological operators)

- **Equal**- Pruebas de Igualdad topológicos (si las geometrías son espacialmente iguales )
- **Disjoint**- Comprueba sí las geometrías son distintas
- **Intersect** - Intersección. Comprueba si las geometrías se cruzan
- **Touch** – Toque: Compruebas si las geometrías se tocan entre ellas
- **Cross**–Cruce: Comprueba si las geometrías se cruzan entre sí
- **Within:** Comprueba si la geometría está dentro de otra geometría dada
- **Contains** - Contiene: Comprueba sí dicha geometría contiene otra geometría dada
- **Overlap** - Superposición: Comprueba si se superpone a otra geometría dada
- **Relate** – Relacionar: Devuelve verdadero(returs True) si la relación espacial especificada se relaciona con la matriz

## 2.3 Modelo de relaciones topológicas. Almendros-Becerra.

### 2.3.1 El modelo topológico

Jesús Almendros y Antonio Becerra.

Utilizan estructuras de datos topológicas que incluyen los siguientes elementos:

- **Nodos**: Puntos geográfica, almacenados como coordenadas (par latitud y longitud) acorde a WGS84 [RW53]. Son usados entre otras formas, para describir características de un mapa sin la magnitud.
- **Caminos**: Lista ordena de nodos, representando una polilínea, o posiblemente un polígono si forman un círculo cerrado.
- **Relaciones:** Listas ordenadas de nodos, caminos y relaciones. Las relaciones son utilizadas para representar la afinidad que existe entre varios nodos y caminos.
- **Etiquetas:** son un par clave-valor (ambos cadenas de caracteres arbitrarios). Las utilizan para almacenar metadatos sobre objetos del mapa (como su tipo, su nombre y sus propiedades físicas). Las etiquetas conectan un nodo, un camino, una relación o a un miembro de una relación.

### 2.3.2 Operadores urbanos

El trabajo realizado por los profesores **Jesús Almendros y Antonio Becerra** se centra en consultas sobre caminos y nodos, aunque está especialmente diseñado para calles y puntos.

Se diseñan un conjunto de Operadores Urbanos [RW54] apto para OSM, cuyo fundamento es el conocido operador espacial definido por Clementini.

A continuación se detalla la proposición de Operadores Urbanos, basados en los operadores de Clementini, mostrando los nombres de los operadores, sus definiciones y tipos, así como el uso de operadores espaciales para su implementación.

| Nombre | Definición | Tipo | Operador Espacial |
|---|---|---|---|
| inWay(p,s) | Devuelve verdadero cada vez que p está en s | Booleano | Contains |
| inPointWays(p) | Devuelve las calles donde está p | Colección | Contains |
| inSameWays(p1,p2) | Devuelve verdadero cada vez que p1 y p2 están en la misma calle | Booleano | InWayPoint, Equals |
| isCrossing(s1,s2) | Devuelve verdadero cada vez que s1 cruza s2 | Booleano | Crosses |
| AllCrossing(s) | Devuelve las calles que cruzan s | Collection | isCrossing |
| isParallel(s1,s2) | Devuelve verdadero cada que s1 es paralelo a s2 | Booleano | Disjoint |
| AllParallels(s) | Devuelve las calles paralelas a s | Collection | isParallel |
| isEnding(s1,s2) | Devuelve verdadero vada vez que s1 termina en s2 | Booleano | Touches (ni es punto inicial ni final) |
| EndingTo(s) | Devuelve las calles donde termina s | Collection | Touches (ni es punto inicial ni final) |
| EndingFrom(s) | Devuelve las calles que terminan en s | Collection | isEnding |
| isContinuation(s1,s2) | Devuelve verdadero cada vez que s1 es continuación de s2 | Booleano | Touches (es punto inicial o punto final) |
| ContinuationTo(s) | Devuelve las calles que son continuacion de s | Collection | Touches (es punto inicial o punto final) |
| ContinuationFrom(s) | Devuelve las calles cuya continuación es s | Collection | isContinuation |

## Tipos de Operadores

Detallan dos tipos de operadores:

- **Operadores Booleanos** que devuelven verdadero o falso.
- **Operadores de colección** que devuelven un conjunto de elementos urbanos.

Este repertorio propuesto de operadores urbanos, está diseñado para cubrir la mayoría de las consultas que conciernen a puntos y calles, que normalmente queremos saber:

- En qué calle está un punto dado.
- Si dos puntos están en la misma calle.
- Así como, si dos calles cruzan un punto.
- Si dos calles son paralelas.
- Como continuar nuestra ruta.
- Tomando una nueva calle que empieza al final de la calle, o tomando una nueva calle que cruza el final de la calle.

Por ejemplo, el operador inPointWays, devuelve una calle (o calles) en las que está un determinado punto.

# 3. Marco tecnológico

## 3.1 BaseX. Motor de consultas

### 3.1.1 Descripción

BaseX es un sistema de base de datos XML ligero y de alto rendimiento con procesador XPath / XQuery, incluyendo soporte para las últimas actualizaciones recomendadas por W3C.

Es compatible con las instancias XML de gran tamaño y ofrece una interfaz altamente interactiva. BaseX es de código abierto (licencia BSD [RW55]), escrito en Java y muy fácil de usar.

Entre sus características se encuentran:

- BaseX es una base de datos nativa XML escalable, con un alto rendimiento de almacenamiento con texto, atributos, texto completo y rutas y muy ligera.
- Con estructuras de almacenamiento compactas y una implementación muy eficiente de XPath y XQuery.
- Interacciones del interfaz gráfico de usuario para las actualizaciones XML, que facilita el acceso visual a los datos almacenados, panel de consulta para introducir las consultas en XPath y XQuery en tiempo real , con resaltado de sintaxis y feedback de errores.
- Es una aplicación gratuita, escrita en Java y con menos de un megabyte de tamaño.

Su implementación es muy eficiente, con procesador **XPath/XQuery** y soporte de la última actualización de la W3C, alto rendimiento de almacenamiento con texto, atributos, texto completo y rutas, apoyo de las Recomendaciones del W3C XPath/XQuery, con una de las más altas tasas de cumplimiento para todas las especificaciones admitidas.

- Editor XQuery en tiempo real , con resaltado de sintaxis y feedback de errores.
- Visualizaciones altamente interactivas, con soporte para documentos XML de gran extensión.
- **Incluye varios índices**, uno de texto completo (esto es posible gracias a la implementación de XPath 1.0).
- Amplia gama de interfaces: REST/RESTXQ, WebDAV, XQJ, XML:DB.
- Instalación sencilla. Una vez arrancado el servidor ya tenemos a nuestra disposición el control del servidor, con lo cual podemos ejecutar todas las órdenes que nos permite **BaseX**.
- Con arquitectura Cliente/Servidor, con soporte de transacciones seguras ACID, gestión de usuarios y autenticación.

Otras operaciones que permite **BaseX** es cambiar de base de datos, crear colecciones de documentos, borrar documentos, borrar colecciones, obtener información del servidor (listar bases de datos, documentos, usuarios…) y realizar copias de seguridad.

### *3.1.2* Funciones principales de BaseX:

- Tiene soporte para XQuery 1.0, alcanzando un 99.3% en el test de la W3C para XQuery.

- Soporte parcial para texto completo de XQuery.
- Varios índices, incluyendo un índice a texto completo (esto es posible gracias a la implementación de XPath 1.0.
- Interacciones del interfaz gráfico de usuario para las actualizaciones XML.
- Panel de consultar para introducir nuestras XPath y XQuery.
- Vista de tabla para obtener una visión plana y clara de cualquier documento XML convencional en el programa.
- No falta tampoco la ayuda, para resolver cualquier duda que nos pudiera surgir.
- Sintaxis de comandos completamente revisada de arriba a abajo.
- Al estar escrita en Java, BaseX requiere que tengamos instalado Java Runtime Environment [RW56] en el equipo.

### 3.1.3 Ventajas e Incovenientes

- **Ventajas**

- Opción para analizar el sistema de ficheros
- Visualización a pantalla completa con F11
- Posibilidad de crear gráficas cartesianas
- Los paneles pueden distribuirse libremente

- **Inconvenientes**

- Ayuda escasa

### 3.1.4 Utilización

**Se utiliza BaseX** como herramienta q conlleva **dos lenguajes** de consulta de documentos XML:

- Por un lado XPath, es **un lenguaje sencillo** de expresiones q permite acceder a partes de un documento XML
- Por otro XQuery, lenguaje q hace uso de XPath y que **facilita la manipulación de documentos** XML. Permite extraer partes de un documento, generar documentos nuevos a partir de los datos originales.

Las bases de datos relacionales son más adecuadas para almacenar datos. Las bases de datos nativas XML son más adecuadas para almacenar documentos.

Con BaseX se puede crear una base de datos constituida por uno o más documentos XML. No son tablas como en los sistemas gestores de base de datos relacionales, son documentos XML. De tal forma que con diversos lenguajes de consulta como XPath o XQuery, se puede acceder al contenido de estos documentos.

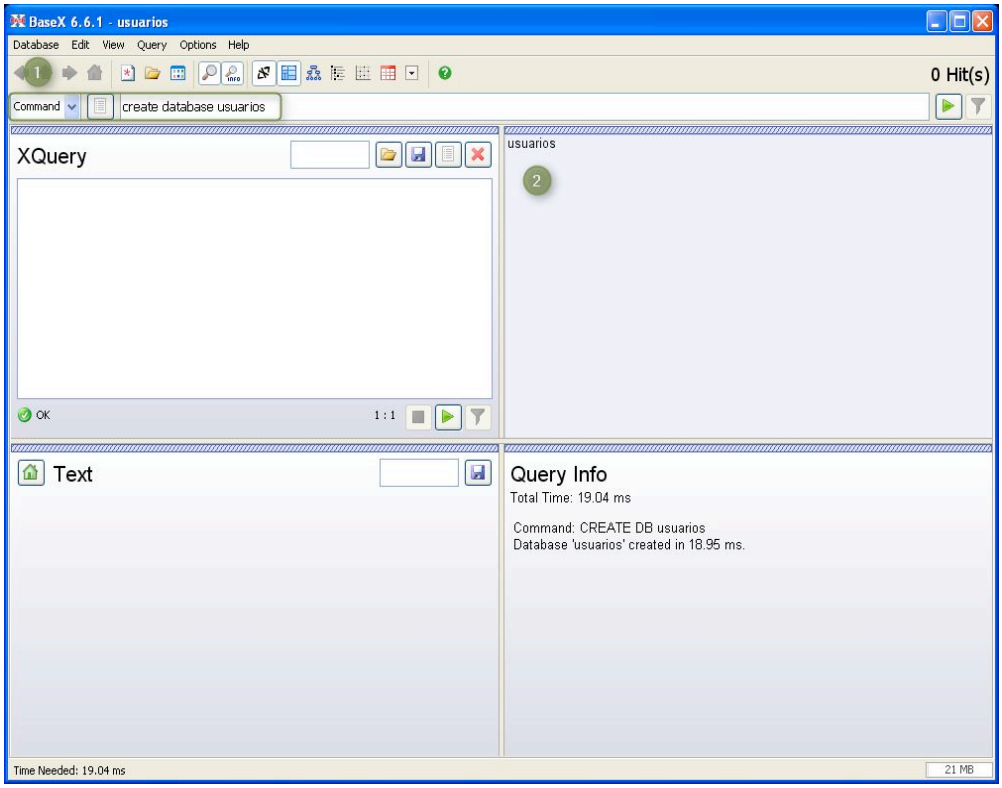
Se puede visualizar el documento XML de diferentes maneras, llamadas vista, que se encuentran sincronizadas entre sí:

- En texto plano
- En forma de mapa
- Como un árbol
- En una serie de carpetas (tipo sistema de ficheros)
- Como una tabla
- Como un diagrama de dispersión

### 3.1.5 Secciones gráficas de la interfaz

- Barra de menús en la parte superior

- Barra de herramientas (botones) inmediatamente debajo
- Líneas de comandos
- Editor de consultas
- Visualizaciones de los datos en árbol, de mapa y de texto,
- Información sobre las consultas ejecutadas (*Query info*)



### 3.1.6 Sentencias ejecutables en el interfaz gráfico de BaseX

**Consultas a la base de datos**

En el interfaz gráfico de BaseX puedo ejecutar tres tipos de sentencias:

- **Opción *Command***: *Órdenes propias de cualquier sistema gestor. Existen comandos propios de un sistema gestor como CREATE DB, OPEN, CREATE INDEX, CREATE USER, ADD, DELETE, REPLACE…*
- ***Opción Search***: *Expresiones XPath, lenguaje de expresiones*
- ***Opción XQuery*** *como lenguaje de consulta*


#### *3.1.6.1 Opción Command* ***CREATE BD***

- Crear Base de Datos y tablas **[Create Database [Nombre]** ]

Lo necesario para comenzar es crear la base de datos XML nativa, para ello se selecciona el menú *Database* y la opción *New*.

Aparecerá una ventana donde se solicita seleccionar un archivo XML o un directorio que contenga agún archivo XML.

- BaseX puede crear bases de datos asociadas a uno o a varios archivos XML

A partir de aquí y según lo que se quiera ejecutar, se darán distintas ordenes al gestor (Command)y se seleccionaran una opción u otra de menú desplegable.

- Importar Datos

- Realizar consultas simples
- Realizar copias de seguridad y exportaciones

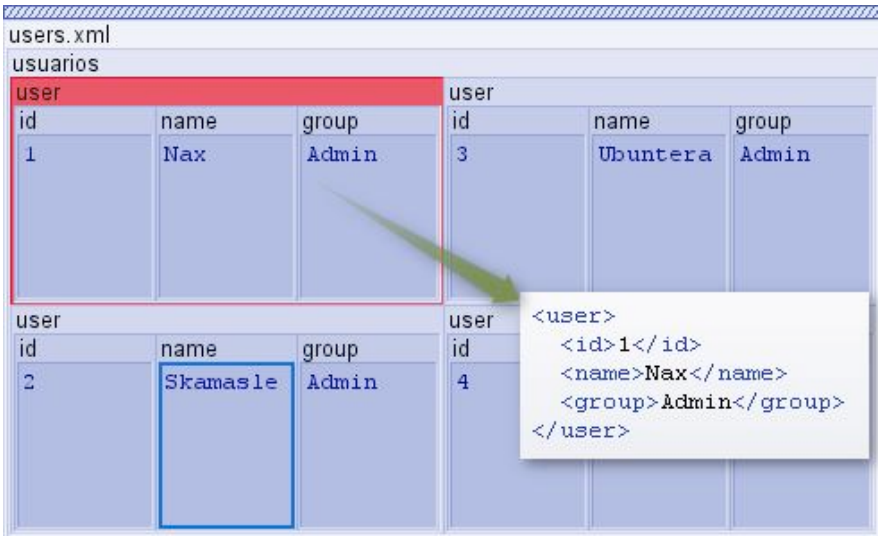
### *3.1.6.2 Comando OPEN [nombre] - Abrir la BD*

Visualizando el documento XML con el que se está trabajando de diferentes maneras, llamadas vistas, que se encuentran sincronizadas entre sí, como:

- ✈ En texto plano o Como un mapa
- ✈ Como un árbol
- ✈ Como una serie de carpetas (tipo sistema de ficheros)
- ✈ Como una tabla
- ✈ Como un diagrama de dispersión

### *3.1.6.3 Agregar registros a la base de datos*

Al importar el archivo nos quedará algo por el estilo:




Si queremos un listado de las bases de datos usamos **SHOW DATABASES.**

- **Backups** Database > Manage, para copias de seguridad

Y para exportar el archivo desde Database > Export XML, eligiendo el directorio a donde exportar y el Encoding que queremos usar.

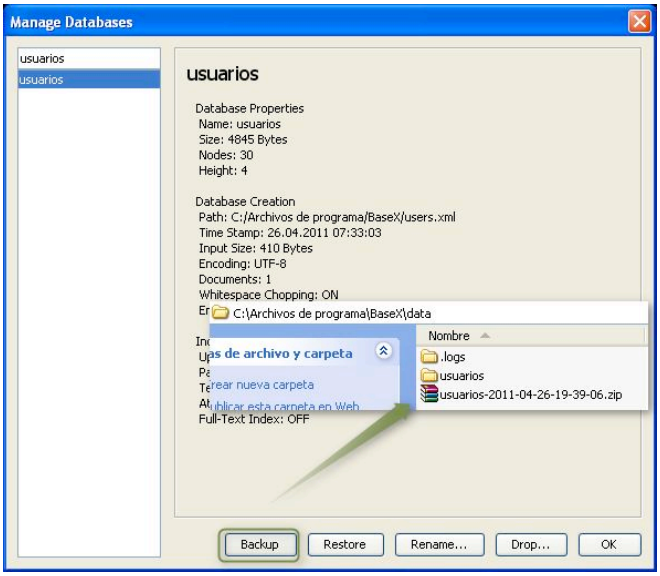
Las bases de datos pueden abrirse desde el menú File, con la posibilidad de especificar qué elementos serán indexados. Una vez cargada en BaseX, la base de datos se mostrará a través de varios paneles relacionados.

El principal es la vista de tabla, desde la cual se puede apreciar la estructura del fichero XML y navegar por él usando el ratón.

- Con un doble clic sobre unos elementos, BaseX lo ampliará con un agradable efecto de zoom.

El texto aparecerá en uno de los paneles, mientras que en los demás puede consultarse la estructura jerárquica y los elementos sueltos.

Para efectuar búsquedas o consultas en lenguaje XQuery, BaseX ofrece una barra superior desde la cual se lanzan operaciones sobre la base de datos cargada.



## 3.2 Java - Lenguaje de alto nivel

### 3.2.1 Definición

- Java es un lenguaje de programación de alto nivel, orientado a objetos.
- Se usa para escribir tanto programas autocontenidos como "applets" (HTML hypertext markup language [RW57]) disponibles en las web.
- Los applets [RW58] permiten a las páginas de red tener un contenido ejecutable accesible con un navegador habilitado con Java.
- Una vez que el compilador e intérprete de Java han sido portados a una nueva plataforma, puede ejecutarse en tal plataforma sin cambios.
- Java es un lenguaje fuertemente tipificado:
  - **El compilador** hace varias verificaciones, tal como comparar los tipos de los argumentos que se pasan a los métodos para asegurar que se ajustan a los tipos de los parámetros.
  - **El intérprete** hace muchas verificaciones durante el tiempo de ejecución, tal como asegurar que el índice de un arreglo no salga de sus límites, o que una referencia a un objeto no sea nula.

No son posibles ni "fugas de memoria" (memory leak, un bloque de memoria que todavía se considera reservado en memoria aun cuando no haya referencias a él) ni "punteros colgantes" (dangling pointers, una referencia a un bloque de memoria que ha sido liberado), ya que

el recolector de basura (garbage collector) de Java recolecta toda la memoria ocupada por objetos inalcanzables.

- No es necesario un procedimiento free()[RW59], susceptible a errores.
- No hay aritmética de apuntadores [RW60]: las únicas operaciones permitidas sobre las referencias a un objeto son asignación de otra referencia a objeto y comparación con otra referencia.
- el programador tiene menor posibilidad de escribir código susceptible a errores que modifique un apuntador.
- Java viene acompañado de una biblioteca de herramientas para la creación de ventanas, conocida como AWT [RW61], que se utiliza por aplicaciones y applets para la implementación de interfaces graficas de usuario.

### 3.2.2 La creación de Java

Java fue diseñado por James Gosling, de Sun Microsystems, en 1990, como software para dispositivos electrónicos de consumo, como calculadoras y microondas.

Inicialmente se llamó Oak (roble en inglés), aunque tuvo que cambiar debido a que dicho nombre ya estaba registrado por otra empresa.

Gosling observó que muchas de las características que ofrecían C o C++ para este tipo de dispositivos aumentaban el gran coste de pruebas y depuración.

Creando un lenguaje de programación para solucionar los fallos que encontraba en C++. Planteándose un nuevo lenguaje de programación lo más sencillo posible, con el objeto de que se pudiese adaptar con facilidad a cualquier entorno de ejecución.

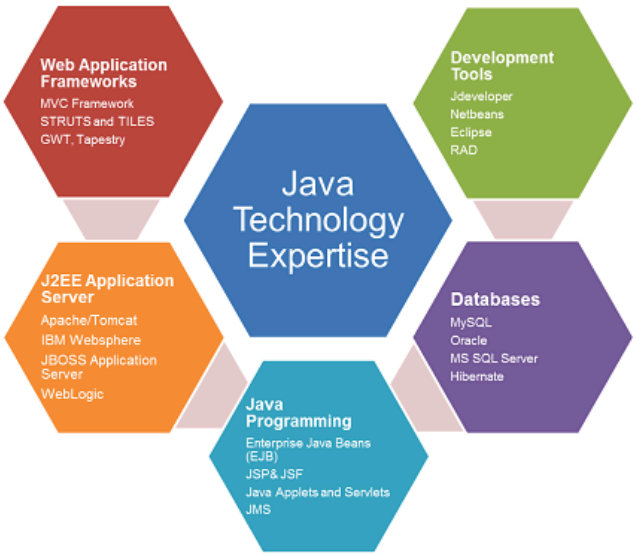
El éxito de Java reside en varias de sus características. Java es un lenguaje sencillo, o todo lo sencillo que puede ser un lenguaje orientado a objetos, eliminando la mayor parte de los problemas de C++ [RW62].

Es un lenguaje independiente de plataforma, por lo que un programa hecho en Java se ejecutará igual en un PC con Windows que en una estación de trabajo basada en Unix [RW63].

También hay que destacar su seguridad, desarrollar programas que accedan ilegalmente a la memoria o realizar troyanos es una tarea difícil en este lenguaje.

Cabe mencionar también su capacidad multihilo, su robustez o lo integrado que tiene el protocolo TCP/IP [RW64], lo que lo hace un lenguaje ideal para Internet.

- Pero es **su sencillez, portabilidad y seguridad** lo que le han hecho un lenguaje de tanta importancia.



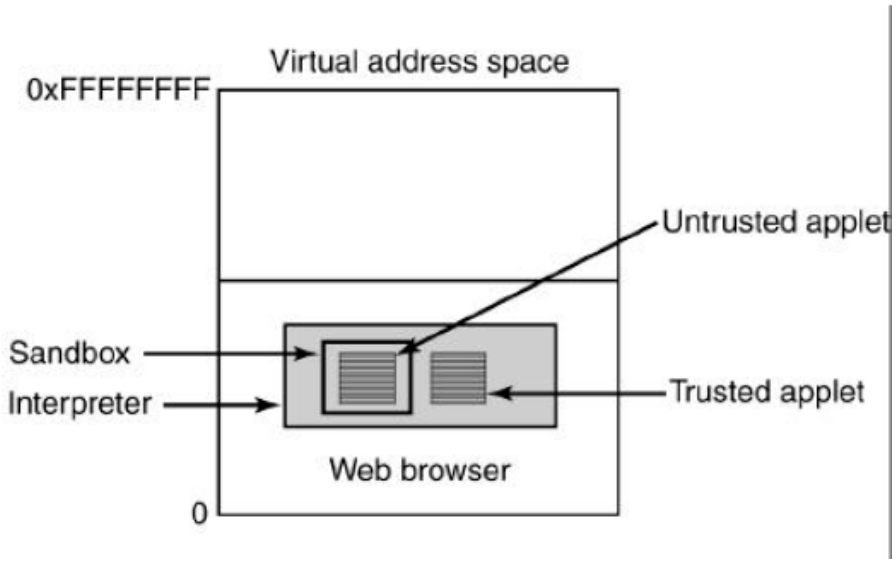
### 3.2.3 Uso típico de Java

(Distribución de aplicaciones a través de Internet)

- **Aplicaciones**: programa que se ejecuta utilizando el sistema operativo del ordenador. Es decir, una aplicación creada con Java más o menos como una creada en C o C++.

- **Applet:** aplicación diseñada para ser transmitida por Internet y ejecutada en un navegador Web compatible con Java. Es similares a las aplicaciones, pero no corren como standalone [RW65] , sino que se apega a un conjunto de convenciones que le permite correr en un navegador (browser [RW66]) compatible con Java (Netscape, Hot Java, etc.).

- Java impone restricciones de seguridad para q los Applets no puedan "dañar" el ordenador en que se ejecutan. Pe. Acceder a ficheros locales, ejecutar otro programa o conectarse a otro ordenador desde el nuestro.



### 3.2.4 Características clave de JAVA

- **Multiplataforma**: Los programas escritos en Java se compilan en un bytecode independiente de la máquina.
- **Simple**: Es un lenguaje relativamente fácil de aprender. No tiene necesidad de apuntadores, crea sus propias referencias y las destruye automáticamente. Realiza la recolección de la basura de manera automática y más eficiente.
- **Orientado a objetos**: Un "objeto es un cuerpo de código ejecutable". Cada objeto tiene su estructura de datos y rutinas o métodos para procesar los datos.
- **Distribuido**: Está pensado para trabajar sobre redes de comunicación. Tiene acceso a recursos de red de una manera relativamente sencilla.
- **Interpretado**: Java es compilado e interpretado. El programa es traducido a un lenguaje intermedio llamado "Java Bytecodes [RW67].
- **Robusto:** Es un lenguaje estable, realiza revisiones más exhaustivas.
- **Seguro**: Pueden forzarse restricciones sobre las operaciones permitidas (los applets no acceden directamente al hardware de la máquina).
- **Arquitectura Neutral**, con amplio conjunto de bibliotecas estándar para trabajar con colecciones y otras estructuras de datos,
- **Portable:** Es independiente de la arquitectura y portable. Se dice, "escribe una vez, ejecuta en cualquier parte" ,
- **Multihilos y Dinámico**

### 3.2.5 Componentes JAVA

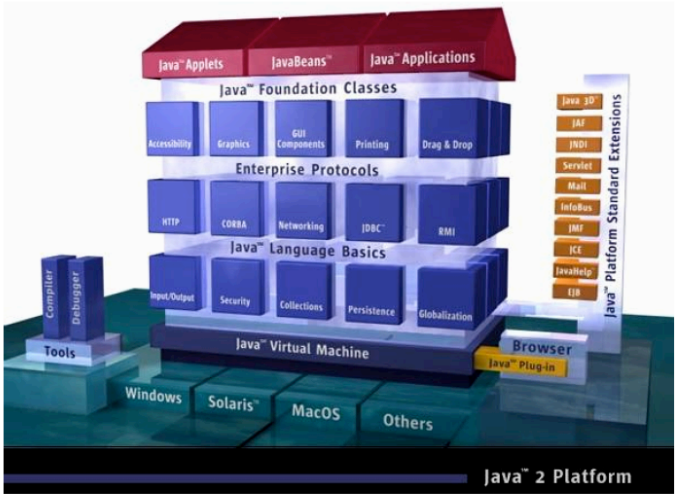
#### *3.2.5.1 La plataforma JAVA*

Una plataforma es el ambiente de hardware y software en el cual un programa corre. La plataforma Java difiere en otras plataformas en que es una plataforma de solo software que corre arriba de otra.

La componen:

- **La máquina virtual de Java (JVM: Java Virtual machine [RW68])** – Imprescindible para poder ejecutar aplicaciones Java.
- El **lenguaje de programación Java**, para escribir aplicaciones.
- **Las bibliotecas estándar de Java (Java Application Programming Interface= Java API** [RW69]**)** – Amplia colección de componentes de software. El API de Java se agrupa en librerías o paquetes de componentes relacionados (clases).

*La plataforma Java*

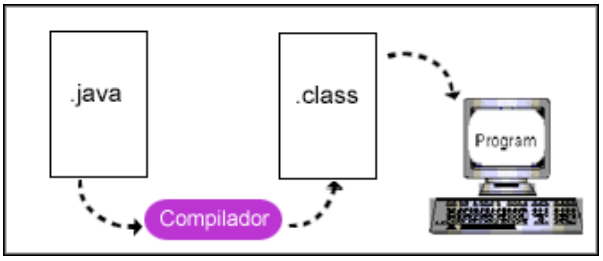


### *3.2.5.2 El Compilador - La máquina virtual JAVA*

Cada programa de Java es compilado e interpretado. Con el compilador, se traduce un programa de Java en un lenguaje intermedio llamado bytecodes, independiente de la plataforma.

- Con un intérprete cada instrucción bytecode es analizada ("parseada" [RW70] ) y se ejecuta o corre .

La compilación sucede solo una vez, mientras que la interpretación ocurre cada vez que el programa es ejecutado.




Los programas en Java son compilados e interpretados

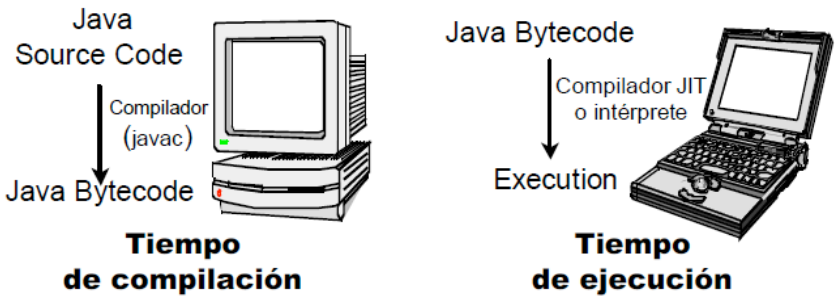
Los bytecodes pueden considerarse como el lenguaje máquina de una máquina virtual, la Máquina Virtual Java (JVM) Los Java bytecodes hacen posible que los programas se escriban una vez, y se ejecuten en donde sea.

Cuando queremos ejecutar una aplicación Java, al cargar el programa en memoria, podemos:

a) Interpretar los bytecodes instrucción por instrucción.

b) Compilar los bytecode para obtener el código máquina necesario para ejecutar la aplicación en el ordenador (complilador JIT [Just In Time] [RW71].

De esta forma, podemos ejecutar un programa escrito en Java sobre distintos sistemas operativos (Windows, Solares, Linux….) sin tener que recompilarlo, como sucedería con programas escritos en lenguajes como C.




### *3.2.5.3 El lenguaje de programación Java*

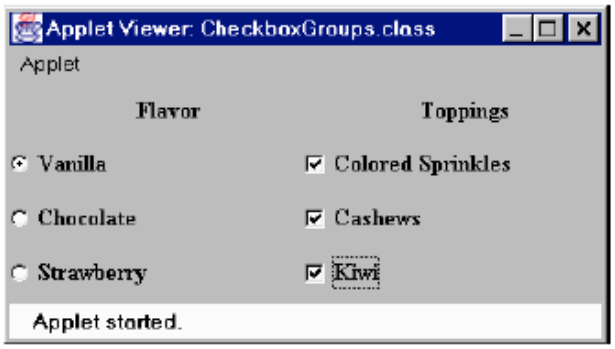
- Java simplifica algunos aspectos a la hora de programar
- Gestión automática de memoria (recolector de basura)
- Comprobación estricta de tipos
- Síntesis simplificada con respecto a C++
- No se manejan punteros explícitamente (todo es puntero en realidad)
- No hay que crear makefiles [RW72] (como en C++)
- No hay que mantener ficheros de cabecera aparte (como en C++)
- No existen macros (#define en C++) al ser propensas a errores

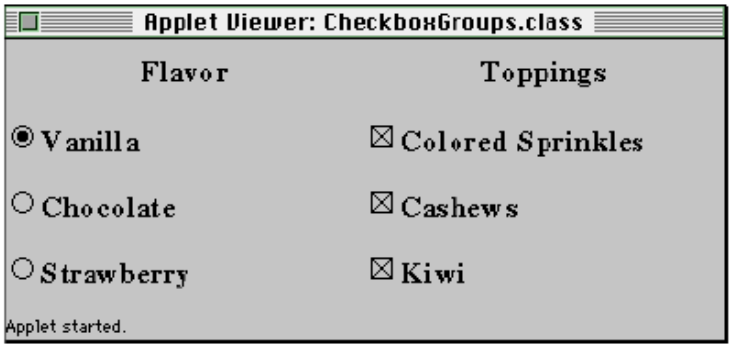
### *3.2.5.4 Las bibliotecas estándar de Java*

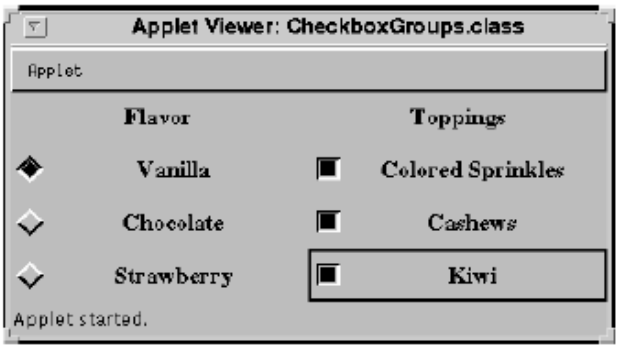
(Java Application Programming Interface= Java API)

Java incluye una biblioteca portable para la creación de interfaces gráficas de usuario (AWT en Java 1.0/1.1 y JFC/Swing en Java 2).

**"Look & feel"** en función del sistema operativo:



## Certificaciones Java.

### JSE [RW73]

- Oracle Certified Associate, Java SE 5/SE 6
- Oracle Certified Professional Java Programmer
- Oracle Certified Master, Java SE 6 Developer

### JEE [RW74]

- Oracle Certified Professional, Java EE 5 Web Component Developer
- Oracle Certified Professional, Java EE 5 Business Component Developer
- Oracle Certified Professional, Java EE 5 Web Services Developer
- Oracle Certified Master, Java EE 5 Enterprise Architect
- Oracle Certified Expert, Java Platform, Enterprise Edition 6 JavaServer Pages and Servlet Developer
- Oracle Certified Expert, Java Platform, Enterprise Edition 6 Enterprise JavaBeans Developer

## 3.3 OSM - OpenStreetMap


**Creado por** Steve Coast

**Fecha de Lanzamiento** 1 de julio de 2004

### 3.3.1 ¿Qué es? (OSM) Mapas digitales

- Un proyecto para crear mapas libres y editables.

Los mapas se crean utilizando información geográfica capturada con dispositivos GPS móviles, ortofotografías ymotras fuentes libres. La cartografía, así como las imágenes creadas como los datos vectoriales almacenados en su base de datos, se distribuyen bajo licencia Creative Commons Attribution-ShareAlike 2.0 [RW75].

Los usuarios suben sus trazas desde el GPS, crean y corrigen los datos mediante herramientas de edición creadas por la comunidad OpenStreetMap [RW76] . Cada día se añaden 25.000 km nuevos de carreteras y caminos con un total de casi 34.000.000 km de viales, así como otros puntos de interés, edificaciones, etc.

El tamaño de la base de datos (llamada *planet.OSM* [RW77] ) se sitúa por encima de los 160 gigabytes (6,1 GB con compresión bzip2), incrementándose diariamente en unos 10 megabytes de datos comprimidos. En enero de 2010 el proyecto **superaba los 200.000 usuarios registrados**, de los cuales cerca de 11.000 realizan alguna edición en la base de datos cada mes.

### 3.3.2 ¿Por qué surge?

En la mayoría de los países la información geográfica pública no es de libre uso, pagando el usuario por la información al adquirirla para su uso. Las licencias de uso restringen su utilización al tener el usuario un derecho limitado de aplicación de la cartografía.

No se puede corregir errores, añadir nuevos datos o emplear esos mapas de determinados modos (integración en aplicaciones informáticas, publicaciones, etc.) sin pagar por ellos.

En los últimos años han surgido iniciativas comerciales como de TomTom [RW78] o de Google, orientadas a animar a los usuarios de sus servicios a completar datos, actualizando y corrigiendo su cartografía y agregando nuevos datos, aunque, en la mayoría de los casos, los usuarios no tienen derecho alguno sobre esta cartografía o datos que están añadiendo o editando, pasando a ser sus contribuciones propiedad de dichas empresas (siguen siendo cartografía propietaria y no libre).

El trabajo de estos servicios comerciales se centra en ciudades principales, lo cual dificulta la incorporación de cartografía de poblaciones pequeñas.

Durante el terremoto de Haití de 2010, voluntarios de OpenStreetMap utilizaron imágenes de satélite disponibles para trazar un mapa de carreteras, edificios y campos de refugiados de Puerto Príncipe en tan sólo dos días. Este mapa está considerado como el "mapa digital de carreteras más completo de Haití". La cartografía ha sido utilizada por diferentes organizaciones que prestan asistencia y socorro.

### 3.3.2 ¿Cómo se va enriqueciendo la base de datos?

La información "de campo" **es realizada por voluntarios**, que consideran la contribución al proyecto, un adictivo hobby. Aprovechan sus desplazamientos a pie, en bicicleta o en automóvil y utilizando un GPS, van capturando las trazas y waypoints [RW79], además de bloc de notas, grabadora de voz o una cámara de fotos digital.

También suelen interrogar a los transeúntes por su conocimiento local sobre datos concretos del lugar que se desconocen (nombres de calles, sentidos de circulación, etc.). Capturan datos geográficos durante una excursión familiar es una forma de descubrir nuevas rutas.

Posteriormente esta información es subida a la base de datos común del proyecto. Algunos contribuidores comprometidos cartografían sistemáticamente su ciudad o núcleo de población en el que residen durante largos periodos hasta ver completada su zona.

También organizan las denominadas mapping parties [RW80], en la que se organizan reuniones de colaboradores para cartografiar y completar zonas determinadas de las que se carece de información y compartir además experiencias (son eventos similares a las LAN parties y a las quedadas de las comunidades virtuales informáticas).

Incluyen fuentes de datos públicas procedentes de instituciones gubernamentales con un tipo de licencia compatible con la de OpenStreetMap permitiendo importar esa información geográfica a la base de datos del proyecto.

Finalmente el proyecto se fundamenta en el gran número de pequeñas ediciones realizadas por la mayoría de los contribuyentes, que corrigen errores o añaden nuevos datos al mapa.

Añadir notas es una funcionalidad de OpenStreetMap que permite a cualquier usuario de la cartografía de OSM informar de errores incorporar notas al mapa para que los contribuidores al proyecto puedan corregirlos. Dispone de una API pública que permite a aplicaciones o sitios web de terceros incorporar esta funcionalidad.

**Estadísticas de uso de OpenStreetMap en febrero de 2011.** Los colores más oscuros corresponden a los países donde los contribuyentes realizan un mayor número de ediciones.

**Capturando con un GPS para cartografiar la ciudad de Estrasburgo**. La bicicleta es uno de los medios de transporte preferidos por los contribuidores de OSM para cartografiar zonas urbanas.

### 3.3.4 Formato de los datos

Con OpenStreetMap se puede producir mapas de carreteras, de senderismo, mapas de vías ciclables, mapas náuticos, mapas de estaciones de esquí, etc.

También se usan en aplicaciones para el cálculo de las rutas óptimas para vehículos y peatones. Gracias a su licencia abierta los datos brutos son de libre acceso para el desarrollo de otras aplicaciones.

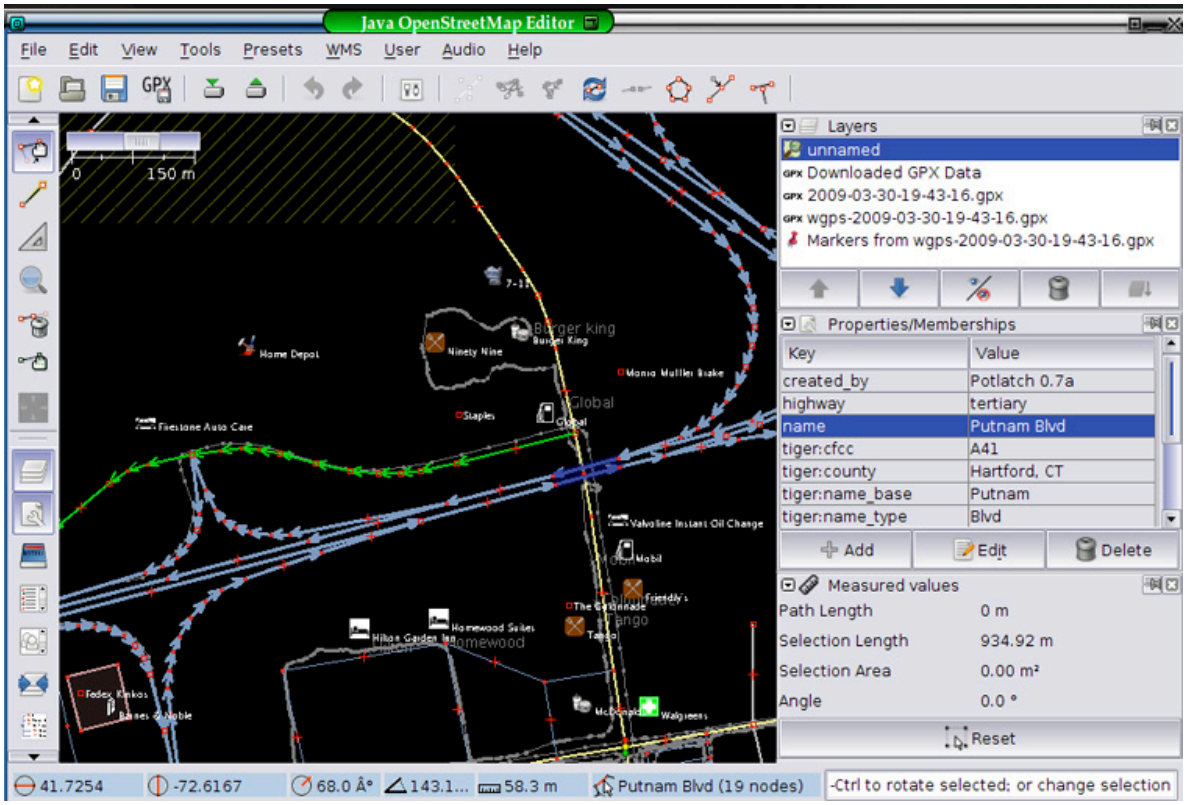
La cartografía de OSM contiene datos en dos dimensiones, por lo que no suele registrar la tercera dimensión, la altura o Z- lo que hace que, por ejemplo, no existan datos sobre líneas de altitud. La importación de datos de elevaciones en la base de datos de OpenStreetMap no está programada. No obstante, existen herramientas para la transformación y representación de los datos de la Misión topográfica Radar Shuttle (SRTM) [RW81] para crear mapas topográficos con isolíneas o sombreados sobre la que superponer los datos de OSM.

A medida que el proyecto ha ido madurando y su base de datos ha mejorado rápidamente en calidad y cobertura, ha ido surgiendo a su alrededor todo un ecosistema de herramientas informáticas y servicios, convirtiéndose en una fuente de datos factible para determinados proyectos complejos que hacen uso de estos datos «de una forma creativa, productiva o inesperada».

OpenStreetMap utiliza una estructura de datos topológica. (conjuntos abiertos y compuestos de subconjuntos), donde los datos se almacenan en el datum WGS84 lat/lon (EPSG:4326) [RW82] de proyección de Mercator.

Los elementos básicos de la cartografía OSM son:

- **Los nodos**. Son puntos que recogen una posición geográfica dada.
- **Las vías** (ways). Son una lista de nodos que representa una polilínea o polígono.
- **Las relaciones**. Son grupos de nodos, caminos y otras relaciones a las que se pueden asignar determinadas propiedades.
- **Las etiquetas**. Se pueden asignar a nodos, caminos o relaciones y constan de una clave (key) y de un valor (ej: highway=trunk).



### 3.3.5 Características actuales y desafíos futuros del proyecto

Los datos en bruto que los contribuidores han capturado con sus dispositivos GPS sirven como guía para dibujar sobre ella las nuevas vías.

Estos datos en bruto suelen cargarse desde el equipo local del usuario o bien solicitando al servidor de OSM que nos descargue aquellas trazas de la zona que vamos a editar y que otros usuarios han subido previamente a OpenStreetMap.

Junto a estos datos en bruto en forma de trazas GPS las herramientas de edición también permiten descargar fotografías aéreas e imágenes de satélite libres sobre las que podemos trazar nuestro mapa.

Una vez tengamos la información geográfica básica sobre la que poder dibujar es el momento de añadir los elementos del mapa que queremos representar mediante nodos, puntos que representan elementos puntuales, vías, líneas que conectan varios nodos que pueden representar elementos lineales o superficies (si la vía empieza y termina en un mismo punto).

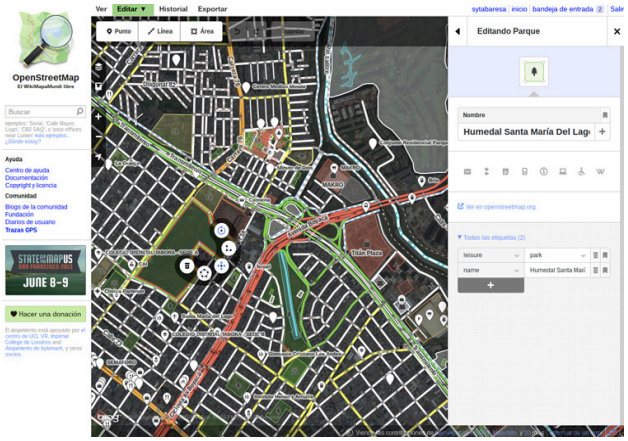
A estos puntos y líneas se les asigna uno o varios atributos que los caracterizan. Por ejemplo, a una línea se le indica la etiqueta y la clave highway:motorway para señalar que es una autopista y name:Autovía del Cantábrico para indicar su nombre.

En principio cualquiera puede asignar libremente cualquier atributo, pero lo más común y recomendado es que se tengan en cuenta la ontología aprobada por la comunidad y documentada en la wiki del proyecto.

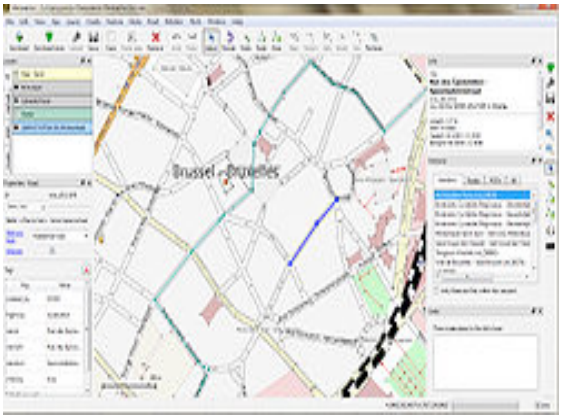
### 3.3.6 Programas para usar OpenStreetMap

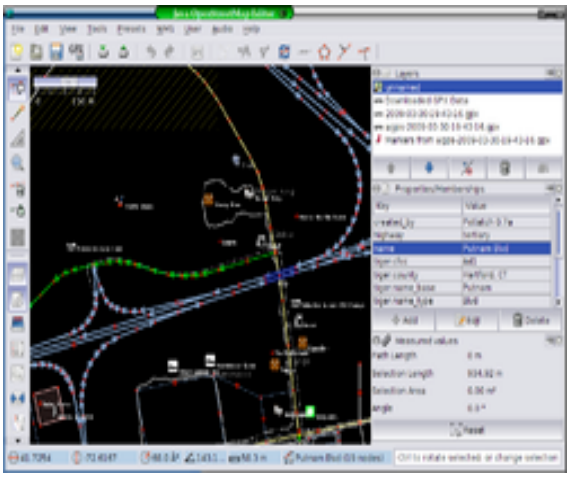
- **Gpsdrive** es un sistema de navegación para coches (bicicletas, barcos, aviones). Por ahora está implementada la representación de la posición en un mapa y un montón de funciones más. GpsDrive [RW83] muestra tu posición suministrada por el receptor GPS con capacidades NMEA, en un mapa ampliable. Los mapas se seleccionan automáticamente dependiendo de la posición. Se puede ajustar la escala preferida, que el programa intenta obtener entre los mapas disponibles.

- **JOSM** (Java OpenStreetMap Editor en su acrónimo en inglés) [RW84] es uno de los principales editores de mapas offline con que cuenta OpenStreetMap. Es una aplicación de escritorio que el usuario se descarga y ejecuta directamente en ordenador. Está basado en tecnología Java y es multiplataforma. Actualmente es el editor más avanzado que posee la comunidad por su versatilidad y características, y en el cual están involucradas en su desarrollo un mayor número de personas.



- **Merkaartor,** Otro editor multiplataforma de mapas que posee un cuidado entorno gráfico haciendo uso de la biblioteca Qt.





- **Mkgmap** es un conversor de mapas en formato OSM, que es el que utiliza OpenStreetMap, a formato IMG, utilizado por los GPS Garmin.

- **GOSMore** es un visor de mapas OSM, como el utilizado por OpenStreetMap. Soporta casi todos los sistemas operativos (Linux, FreeBSD, Mac OS X, Windows, Windows CE, Maemo, etc.)

### 3.3.7 Cálculo de rutas y navegación

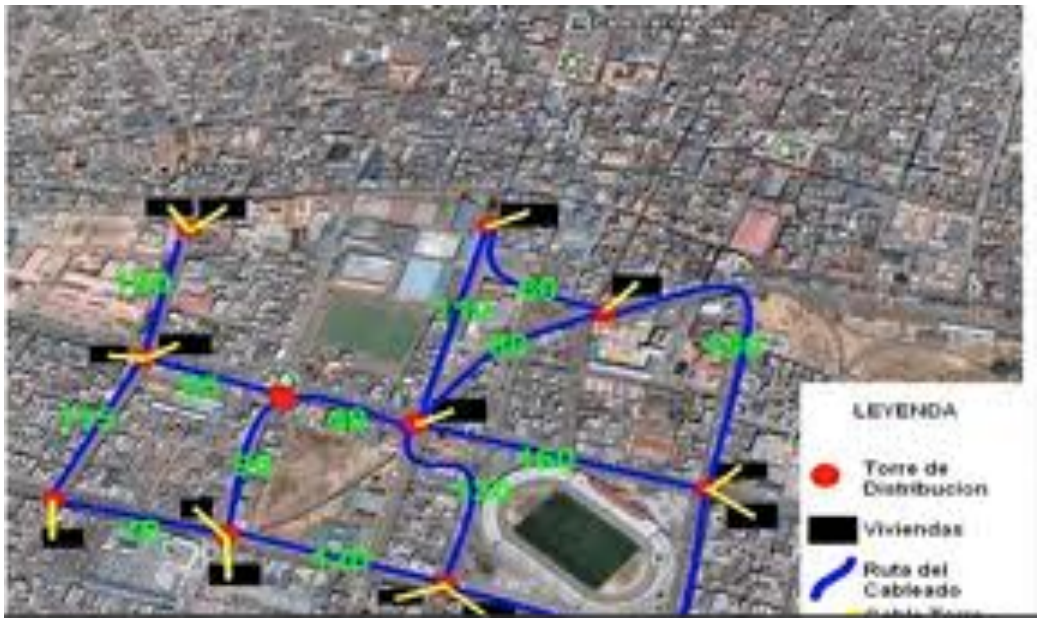
El cálculo de las rutas óptimas utilizando los datos de OpenStreetMap no está totalmente desarrollado, sin embargo el avance en este sentido en los últimos meses ha sido muy importante.

En numerosas regiones los datos existentes hasta la fecha todavía no son suficientemente detallados para que lleguen a ser plenamente fiables, ya que a menudo se carece de información sobre nombres de calles o números de policía, por ejemplo.

Así mismo, pueden existir problemas de consistencia topológica al haber errores comunes de digitalización involuntarios o de etiquetado, como viales no conectados, cruces de calles sin nodo en común, errores de flujo motivados por el sentido de las vías, etc.

El proyecto OpenStreetMap y diversos usuarios facilitan a los colaboradores herramientas para poder detectar y corregir la mayoría de estos problemas.

Diferentes webs ofrecen servicios de enrutamiento basados en datos OSM mediante conocidos algoritmos de búsqueda [RW85] en grafos (A*, Dijkstra, etc.). En la mayoría de los casos estas implementaciones no tratan necesariamente el camino más corto, sino el de menor impedancia en función de las etiquetas OSM tenidas en cuenta.



## 3.3 JOSM

### 3.3.1 JOSM editor de OSM

JOSM (Java OpenStreetMap Editor en su acrónimo en inglés) es uno de los principales editores del mapa con que cuenta OpenStreetMap.

Desarrollado por Immanuel Scholz y actualmente está mantenido por Dirk Stöcker (anteriormente lo hizo Frederik Ramm). Es un editor enriquecido especialmente dirigido a usuarios de OSM experimentados, También incorpora versiones de un modo básico y un modo avanzado, para adaptarlo al tipo de usuario.

JOSM es un programa de descarga, uso libre y gratuito para trabajar sobre un programa y luego cargar la información tratada en los servidores de OSM.

**Se utiliza este editor off-line cuando:**

- Se modifican un gran conjunto de datos.
- Para editar datos sin necesidad de que estos cambien de inmediato en la base de datos de OpenStreetMap (realizar pruebas).
- Cuando has tomado fotos en la recogida de datos y quieres que aparezcan localizadas en el lugar donde fueron tomadas.
- Para editar una zona en la que cree que otras personas están trabajando también editando datos. JOSM permite la fusión de estos cambios y la solución de los conflictos de edición).

Se puede añadir al mapa lo que quieras, sin embargo hay unas reglas básicas que respetar.

En OSM se representará solamente lo que se ajuste a esas reglas básicas, mediante los iconos normalizados OSM tiene creado ya un modelo y unas etiquetas realizadas en base a su experiencia a nivel internacional, recogiendo la mayor parte de los elementos existentes en las zonas urbanas, tales como supermercados, gasolineras, o fuentes.

### 3.3.3 Complementos de JOSM

Hay una larga lista de complementos disponibles mediante el administrador de complementos integrado en JOSM. Algunos significativos son:

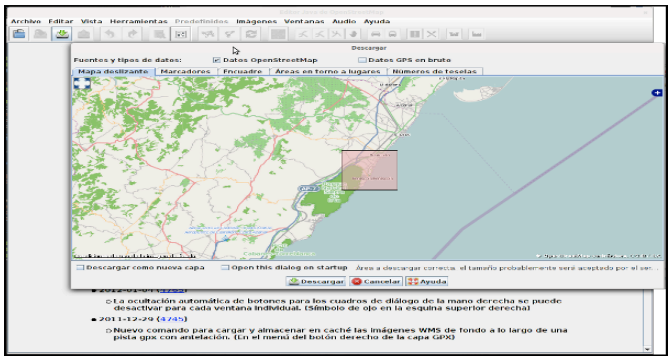
-  AddrInterpolation Agrupa las entradas comunes de "Interpolación de direcciones" en un mismo cuadro de diálogo, así como una opción para generar automáticamente los nodos individuales de los números de portal de una vía
-  alignways Alinea una par de segmentos de vías seleccionados haciendo girar uno de ellos alrededor del designado como pivote
-  buildings_tools Herramientas para dibujar edificios
-  ColumbusCSV Importa archivos CSV propietarios del GPS logger
-  CommandLine Implementa una línea de comandos y permite crear comandos
-  Create_grid_of_ways Crea cuadrícula de viales
- dataimport Permite importar directamente distintos formatos de archivo en JOSM.

-  DirectDownload / DirectUpload : Descarga/ sube trazas GPS desde / a openstreetmap.org
-  download_along Descargar datos OSM cercanos a una vía
-  ElevationProfile Muestra el perfil de elevación y algunos datos estadísticos de una traza GPX
-  ext_tools Utilizar scripts externos en JOSM
-  FastDraw Dibuja rápidamente elementos aburridos de digitalizar sin estar dando todo el rato al botón del ratón.
-  FixAddresses Identifica y corrige las direcciones de calles no válidas de una manera cómoda
-  geochat Hablar con los usuarios más cercanos que estén también editando el mapa y notifica cuando alguien se aproxima al área editada
-  geotools Proporciona partes de la biblioteca GeoTools para otros componentes de JOSM
-  globalsat Descargar puntos GPS desde el registrador de datos GlobalSat DG100 directamente en JOSM
-  gpsblam Analiza un conjunto de puntos GPS para obtener su centro y la dirección de dispersión
-  graphview Visualiza la información de enrutamiento como un grafo de red.
-  HouseNumberTaggingTool Sencilla herramienta para etiquetar números de policía
-  imagery-xml-bounds Genera imágenes XML a partir de los límites de un multipolígono

### 3.3.4 Operaciones que se pueden realizar desde JOSM

***Descargando datos***.

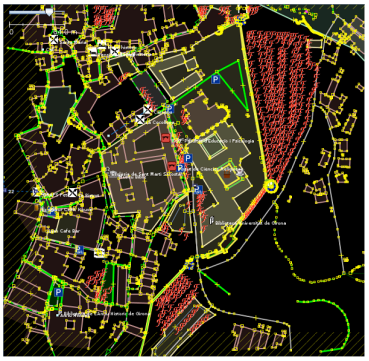
- Seleccionar una zona para descargar los datos, (botón de descarga  )

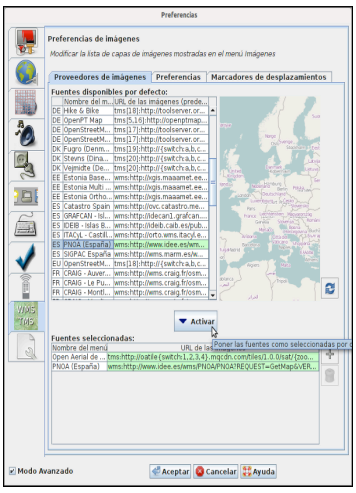


- También se puede utilizar el mapa que se nos muestra para seleccionar la zona, o bien a través de las pestañas dar unas coordenadas que definan un área de trabajo o buscar por nombre.

***Filtrar información***

Dada la cantidad de información en determinadas zonas, es necesario filtrar poder trabajar cómodamente. Para filtrar la información utilizaremos la ventana de filtros a la que se accede pulsando el botón de filtro  .



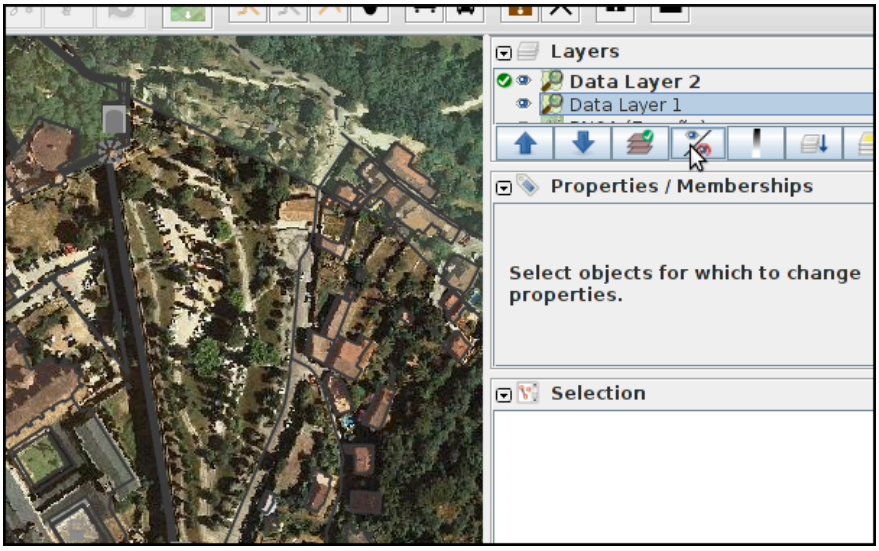


***Añadir imágenes***

Añadir las imágenes de fondo es un proceso en dos pasos, primero hay que definir el origen de datos y después seleccionarlo para que cargue en la zona de visualización. Es una capa que se puede activar o desactivar  , o cambiar la transparencia  .

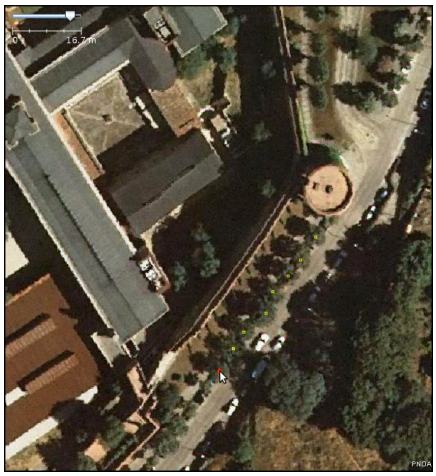
## Digitalizar

Para digitalizar un punto, haremos zoom sobre una zona y con la rueda del ratón, a continuación pulsamos sobre el botón agregar 




## Nodos

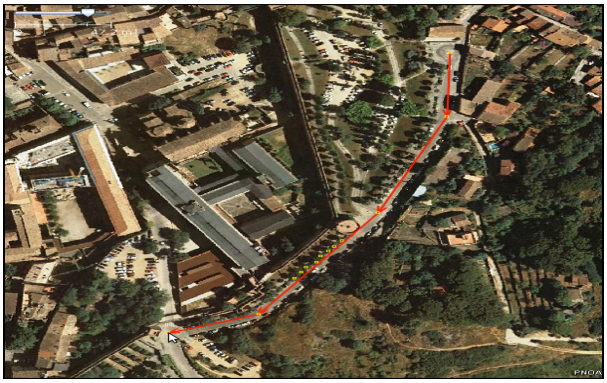
JOSM está pensado para añadir elementos lineales por lo que por defecto espera tener que añadir líneas y puntos



En realidad estamos simplemente poniendo los Nodos, para que OSM los reconozca también podemos añadir Etiquetas

## *Vías*

Para digitalizar una vía, buscaremos un nivel de zoom que nos permita ver la vía en su totalidad o por lo menos una parte muy significativa de ella, mientras vamos marcando nodos de manera consecutiva intentando seguir el eje de esta y respetar la forma siguiéndola sobre la ortofotografía. Es interesante que además pongamos un nodo en cada intersección que tenga la vía, lo que facilitará interconectar las vías entre sí.



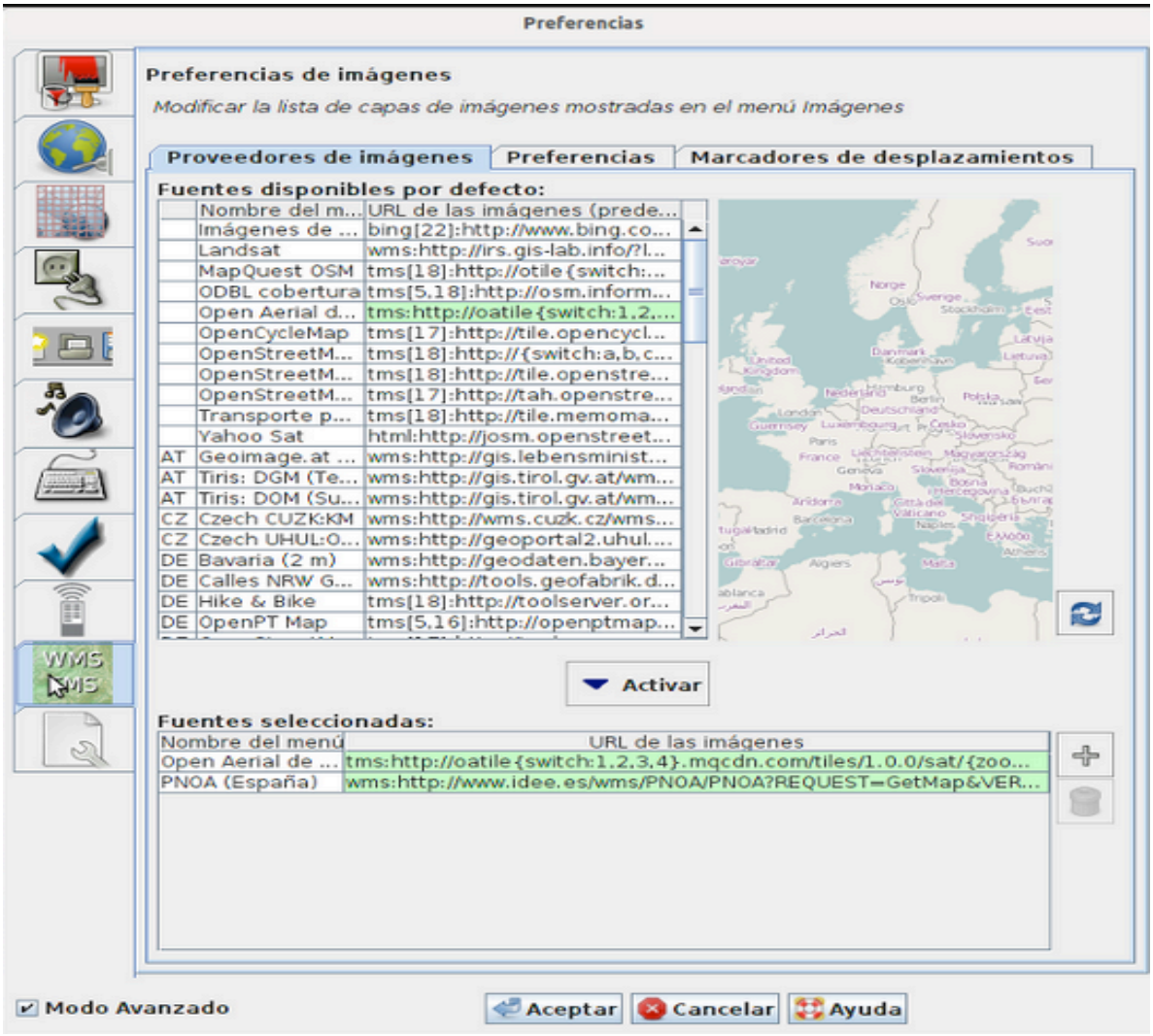
## *Áreas*

Las áreas son vías que empieza y acaba en el mismo punto y tiene una etiqueta que la identifica. Digitalizo un área cerrando la vía al final sobre el primer nodo que se digitaliza.

Los edificios son seguramente el caso más típico de áreas a digitalizar.

Buscando en el listado de elementos comunes en los mapas Map Features de OSM, pero además si pulsamos sobre la palabra concreta nos lleva a la entrada específica de la wiki en la que explican las características a tener en cuenta y generalmente se detallan las claves a las que también suelen estar asociadas las entidades a cartografiar e incluso ejemplos.

La aplicación JOSM tiene, para determinados elementos, una serie de entradas de menú que permiten rellenar de manera cómoda las etiquetas que pueden asignarse a grupos de elementos, para lo que primero hay que seleccionarlos manteniendo pulsada la tecla Mayúsculas mientras se va haciendo click; para posteriormente aplicar la etiqueta.



## 3.4 Java Topology Suite (JTS)

Java Topology Suite (JTS) es una API de Java que proporciona un modelo de objetos espaciales y funciones fundamentales geométricas 2D, utilizando un modelo de precisión explícita y algoritmos geométricos robustos.

JTS ha sido desarrollada por la empresa Vivid Solutions [RW86] y está implementada íntegramente en el lenguaje de programación Java, distribuyéndose bajo licencia LGPL[RW87] .

JTS pretende que sea utilizado en el desarrollo de aplicaciones que admiten la validación, la integración y consulta de bases de datos espaciales.

- Es una especificación de diseño para las clases, métodos y algoritmos implementados en el JTS Topology Suite.
- JTS implementa el OpenGIS(OGC) Simple Features Specification (SFS) [RW88] con la mayor precisión como sea posible.
- JTS cumple con la especificación *Simple Features Specification for SQL* (SFSQL) publicada por el Open Geospatial Consortium y proporciona una implementación completa, consistente y robusta de algoritmos espaciales bidimensionales.

### 3.4.1 API de Java

**JTS es una API** que proporciona:

- Una implementación del modelo de datos espacial (Spatial Data Model) [RW89] definida en el OGC. Además de ofrecer una especificación para SQL (SFSQL).
- Una completa y consistente, la implementación de algoritmos fundamentales 2D espaciales, incluyendo predicados binarios (como el tacto y la superposición) y los métodos de análisis espacial (como la intersección y tampón).
- Un modelo de precisión explícita, con algoritmos que pueden manejar de forma adecuada las situaciones que dan lugar al colapso dimensional.
- Solidas Implementaciones de las operaciones geométricas computacionales claves.

**El diseño API** seguirá convenciones de Java siempre que sea posible. Por ejemplo:

- Las funciones de acceso a utilizar cumplen la convención de Java getX [RW90] y setX [RW91].
- Usarán la convención ISX [RW92].
- Los métodos comenzarán con una letra minúscula .
- Las funciones de JTS proponen un modelo de precisión definida por el usuario. Algoritmos JTS serán sólidos bajo ese modelo de precisión.
- Métodos devolverán resultados topológicamente y geométricamente correctos dentro del modelo de precisión siempre que sea posible.

- La corrección es la más alta prioridad; eficiencia de espacio y el tiempo es importante, pero secundario.
- JTS tiene que ser lo suficientemente rápido como para ser utilizado en un entorno de producción.
- Los algoritmos y códigos utilizados en la JTS deben ser claros y bien estructurados, para facilitar comprensión por parte de otros desarrolladores.

## Spatial Data Model

Todos los métodos de JTS asumen que sus argumentos son objetos geométricos válidos, de acuerdo con la definiciones dadas en el SFS. Geometrías en JTS tienen un interior, un Límite, y un exterior.

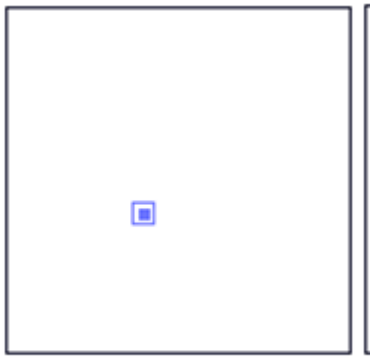
### 3.4.2 Clases de entidad SIMPLES

- Todas las clases de geometría permiten objetos vacíos que se creen y apoyan el método [estaVacia].
- Geometrías vacías serán representados por sus arreglos internos que tienen longitud cero.
- Todas las clases de geometría apoyan el método [equalsExact ()], que devuelve true si dos ssubclases geometricas son equivalentes y tienen secuencia idéntica (s) de coordenadas.
- Dos objetos son "Equivalente" si sus clases son idénticos.
- La única excepción es LinearRing y LineString, que JTS estima equivalente.
- Todas las clases de geometría apoyan el método clone (), que devolverá una copia profunda del objeto.

## Geometría

- *La geometría es no instancia y se implementa como una clase abstracta.*

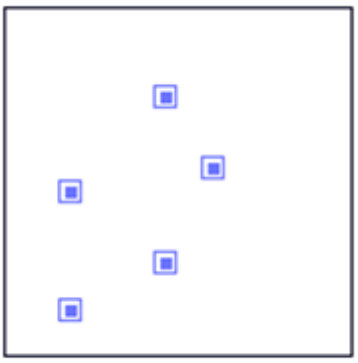
### 3.4.3 Tipos de datos espaciales JTS



Point

#### Punto

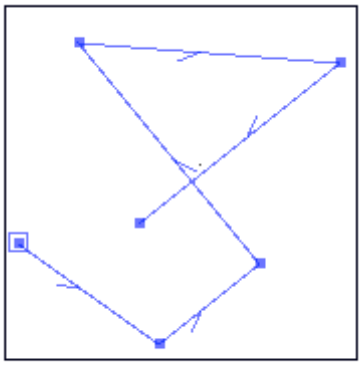
Un punto se implementa como una sola coordenada.



MultiPoint

#### MultiPoint (Multipunto)

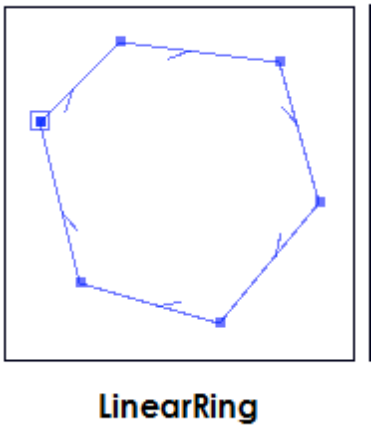
Un MultiPoint hereda la implementación de GeometryCollection, pero sólo contiene puntos.



LineString

#### LineString

Un LineString se implementa como un conjunto de coordenadas.



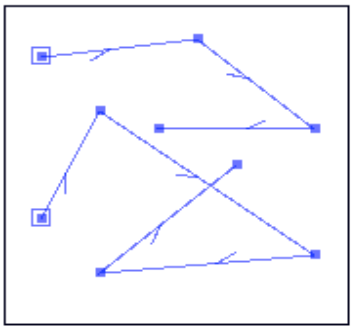
LinearRing

#### LinearRing

Una LinearRing contiene n coordenadas y se implementa con una serie de coordenadas que contiene n + 1 puntos, y coord [0] = coord [n].

### MultiLineString

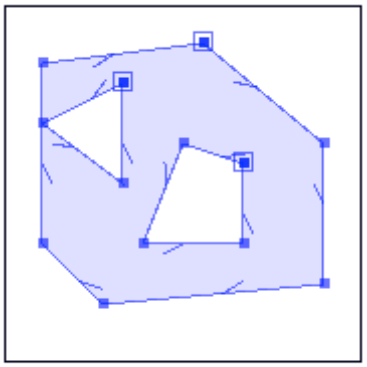
Una MultiLineString hereda la implementación de GeometryCollection, pero contiene sólo Cadenas de líneas.



MultiLineString

### Polígono

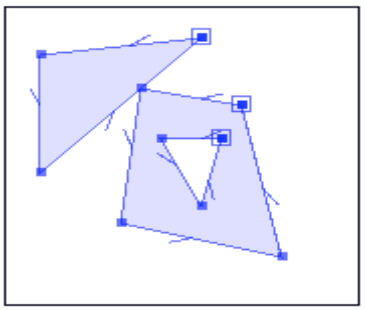
Un polígono es implementado como una única LinearRing para la carcasa exterior, y una matriz de LinearRings para los agujeros. La capa exterior está orientada CW y los agujeros están orientados CCW.



Polygon

### MultiPolygon (Multi polígono)

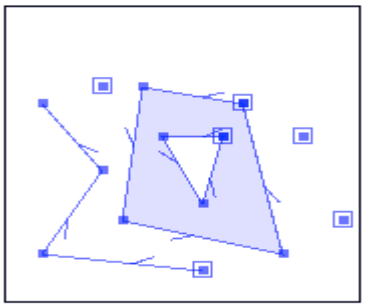
Un MultiPolygon hereda la implementación de GeometryCollection, pero contiene sólo Polígonos.



MultiPolygon

### GeometryCollection (Colección Geometrica)

Una GeometryCollection se implementa como una matriz de objetos geométricos.



GeometryCollection

### 3.4.4 Binary Predicates (Predicados Binarios)

JTS soporta un conjunto completo de los predicados binarios. Los predicados binarios utiliza el método donde toman dos geometrías como argumentos y devuelven un valor booleano que indica si las geometrías han llamado la relación espacial.

Es importante tener en cuenta que los predicados binarios son operaciones topológicas. Incluso para los predicados aparentemente sencillos como por ejemplo (=) iguales, es fácil encontrar casos en los que una comparación punto a punto no produce el mismo resultado que en una comparación topológica.

Las relaciones soportadas son:

- **Equals (Igual)** La relación igual, se aplica a todas las combinaciones de geometrías. Dos geometrías son topológicamente iguales si y sólo si sus interiores forman intersección y ninguna parte del interior o perímetro de una intersección geométrica con el exterior de la otra.
- **Equals ()** es una relación topológica, y no implica que las geometrías tengan los mismos puntos o incluso que sean de la misma clase. (Esta norma más restrictiva de la igualdad se implementa en el método).
- **equalsExact ()**

| ***Argument Dimensions*** | ***Relate Pattern*** |
|---|---|
| all | T*F**FFF* |

- **Disjoint (Distintos)**

| ***Argument Dimensions*** | ***Relate Pattern*** |
|---|---|
| all | FF*FF**** |

- **Intersects (Cruzan)**

A.intersects(B) = ! A.disjoint(B)

- **Touches (Tocan)**

| ***Argument Dimensions*** | ***Relate Pattern*** |
|---|---|
| P/L, P/A, L/L, L/A, A/A | FT*******<br>or F**T*****<br>or F***T**** |
| P/P | undefined |

- **Crosses (discurren)**

| ***Argument Dimensions*** | ***Relate Pattern*** |
|---|---|
| P/L, P/A, L/A | T*T****** |
| L/L | 0******** |
| P/P, A/A | undefined |

- **Within (Dentro)**

| ***Argument Dimensions*** | ***Relate Pattern*** |
|---|---|
| all | T*F**F*** |

- **Contains (Contiene)**

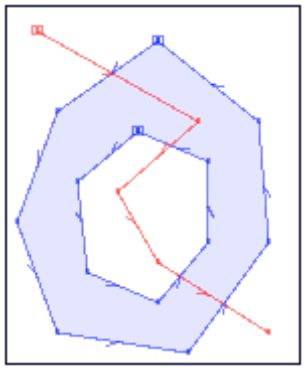
A.contains(B) = A.within(B)

- **Overlaps (Superppone)**

| ***Argument Dimensions*** | ***Relate Pattern*** |
|---|---|
| P/P, A/A | T*T***T** |
| L/L | 1*T***T** |
| P/L, P/A, L/A | undefined |

El algoritmo utilizado para el cálculo de predicados binarios en JTS es robusto, y no está sujeto a los problemas de colapso dimensionales.

**Ejemplo:**



**Dos comparaciones geométricas**

Binary Predicates

| | AB | BA |
|---|---|---|
| Equals | F | F |
| Disjoint | F | F |
| Intersects | T | T |
| Touches | F | F |
| Crosses | T | T |
| Within | F | F |
| Contains | F | F |
| Overlaps | F | F |

**Predicado Binario**

Intersection Matrix

| | | B | | |
|---|---|---|---|---|
| | | Int | Bdy | Ext |
| A | Int | 1 | F | 2 |
| | Bdy | 0 | F | 1 |
| | Ext | 1 | 0 | 2 |

**DE-9IM retorno relativo**

## 3.5 Desarrollo de plugin en JOSM

## Complementos de JOSM

Son características adicionales de JOSM que ayudan a mejorar las habilidades para dibujar mapas. JOSM te permite instalar numerosos complementos, que son herramientas extra que permiten realizar tareas específicas en el programa, como el complemento que te permite usar SDS (*Separate Data Store* o Almacén de Datos Independiente) o el complemento para usar *Walking Papers*. Hay muchos complementos disponibles para JOSM.

Para instalar un complemento desde Edición –> Preferencias → «Complementos» → Marcar la casilla del complemento → clic en el botón Aceptar . Cuando se instala un nuevo complemento es necesario reiniciar JOSM.

### 3.5.1 Otros complementos parar JOSM

| | | |
|---|---|---|
| | DirectDownload | Descarga tus trazas GPX desde openstreetmap.org |
| | DirectUpload | Este componente sube directamente trazas GPS desde la capa activa en JOSM a openstreetmap.org |
| | download_along | Descargar datos OSM cercanos a una vía |
| | editgpx | Permite al usuario que la marca de tiempo sea anónima y borrar las partes más grandes del GPX que suelen corresponder a trazas muy rápidas |
| | buildings_tools | Herramientas para dibujar edificios |
| | ColumbusCSV | Importa archivos CSV propietarios del GPS logger Columbus/Visiontac V-900 a una capa GPX |
| | FastDraw | Dibuja rápidamente elementos aburridos de |

| | | |
|---|---|---|
| | | digitalizar sin estar dando todo el rato al botón del ratón |
| | geochat | Hable con los usuarios más cercanos que estén también editando el mapa. Será notificado cuando alguien se aproxime a su área de edición |
| | geotools | Proporciona partes de la biblioteca GeoTools para otros componentes de JOSM. No se pretende que sea instalado directamente por los usuarios, sino más bien que sirva como una dependencia para otros componentes. |
| | livegps | Soporte GPS para dispositivos de entrada (punto en movimiento) a través de una conexión con un servidor DSGP. |
| | mirrored_download | Simplifica la descarga desde distintas APIs de sólo lectura. |
| | pdfimport | Importa un archivo PDF y los convierte a vías. |
| | photo_geotagging | Agrega información sobre la posición del GPS en la cabecera del archivo de imagen. Acceda a esta característica pulsando el botón derecho del ratón sobre la capa de imagen. |
| | print | Añade la posibilidad de impresión del mapa a JOSM |
| | scripting | Runs scripts in JOSM |
| | sds | Carga datos desde SDS |
| | SeaMapEditor | Editar características de OpenSeaMap |
| | | |
| | surveyor | Permite añadir marcadores/nodos a las posiciones GPS actual. |
| | tag2link | Inicia el navegador mostrando un recurso Web |

| | | |
|---|---|---|
| | | acerca del objeto seleccionado recogido en las etiquetas, tales como la Wikipedia |
|  | turnlanes | Proporciona una sencilla interfaz gráfica de usuario para agregar, editar y suprimir carriles de giro. |
|  | waydownloader | Descargar fácilmente a lo largo de una larga serie de vías interconectadas |
|  | wikipedia | Simplifica la vinculación de objetos OSM con artículos de Wikipedia |
|  | wms-turbo-challenge2 | Conduce un coche de carreras desde el punto A al B sobre una imagen aérea. Deja a los cactus atrás... |

Vamos a ver un ejemplo del plugin "Directupload" para añadir trazas y puntos de referencia GPS al servidor de OSM es útil por muchas razones, ya que las trazas GPS son la forma más útil de recopilar y georreferenciar objetos en OSM.

Los dispositivos GPS tienen mayor precisión que las imágenes por satélite así que son herramientas útiles para comprobar si una imagen aérea está desplazada. Usar muchas trazas GPS (a mayor número de trazas mayor capacidad para determinar la precisión de la geolocalización) que permiten determinar si la imagen de fondo está desalineada.

Subir trazas al servidor permite compartir la información. Permite a la gente que no tiene acceso al terreno (ya sea porque no viven en esa zona o porque no tienen acceso a dispositivos GPS) ayudar en la digitalización. Hay dos formas más de subir trazas:

- Mediante el complemento de JOSM.
- En la página principal de OSM.

Los puntos de referencia GPS no pueden subirse directamente a la base de datos de OSM. Sin embargo, pueden ser convertidas en trazas y después ser subidas temporalmente, de forma que pueden ser mostrados como objetos de fondo en Potlatch.

Una vez abierto un archivo GPX en JOSM, selecciona «Herramientas» y dentro de este menú, «Subir trazas». Describe el archivo GPX, escribe algunas etiquetas y marca la visibilidad.

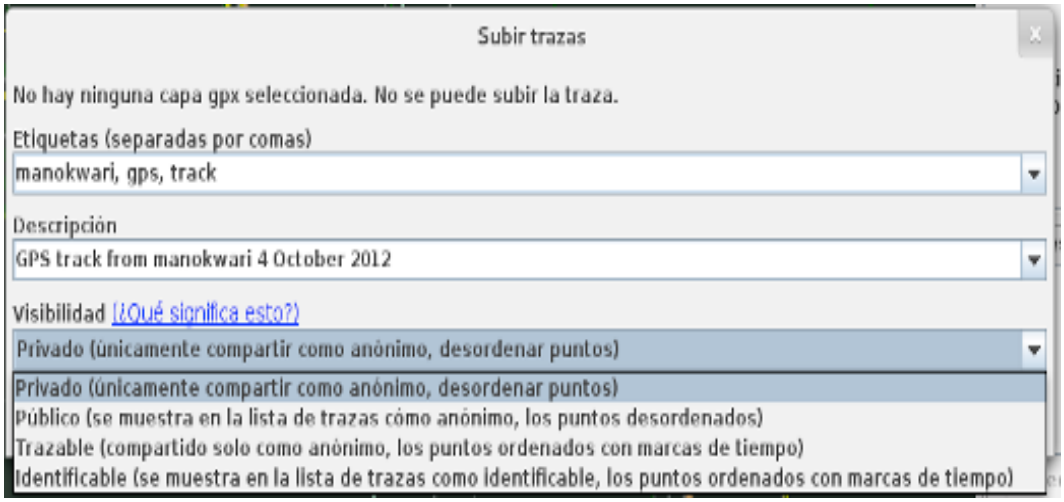
Para la visibilidad, elegir entre privado, trazable, público o identificable.

1. **Identificable**: la traza se muestra públicamente en Tus trazas**GPS y en la lista de trazas GPS públicas. Otros usuarios pueden descargar la traza sin procesar y relacionarla con el nombre de usuario. Las marcas de tiempo también estarán disponibles mediante la API pública de GPS.

2. **Público**: tus trazas se mostrarán públicamente en Tus trazas GPS y en la lista de trazas GPS públicas. Los demás usuarios todavía serán capaces de descargar las trazas sin procesar de la lista pública y ver las marcas de tiempo que contiene. Sin embargo, los datos mostrados en la API no harán referencia a la página de tus trazas, ni dispondrán de marcas de tiempo, aunque los puntos estarán ordenados cronológicamente.

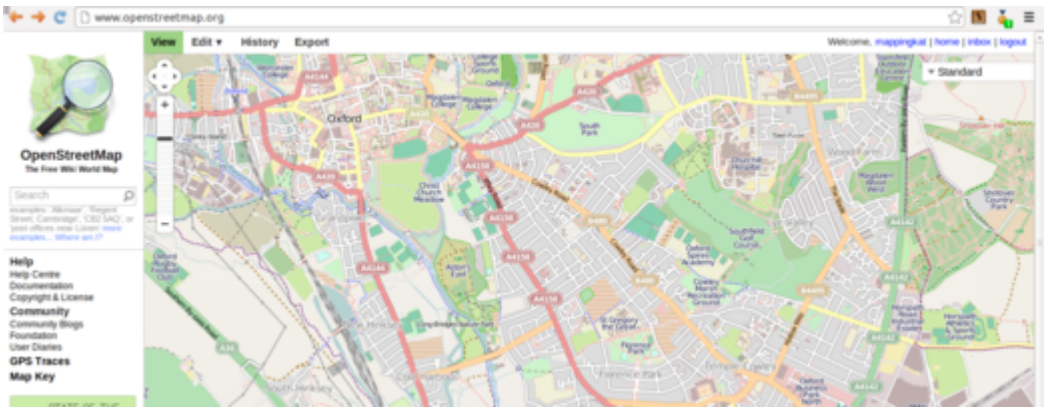
3. **Trazable**: las trazas **no** se mostrarán en ningún listado público, pero los puntos seguirán estando disponibles mediante la API pública GPS **con marcas de tiempo**. Los demás usuarios podrán descargar los puntos pero estos no estarán asociados a ti.

4. **Privado**: Las trazas **no** se mostrarán en ningún listado público. Los puntos estarán disponibles en la API GPS pública, ordenados cronológicamente pero **sin marcas de tiempo**.



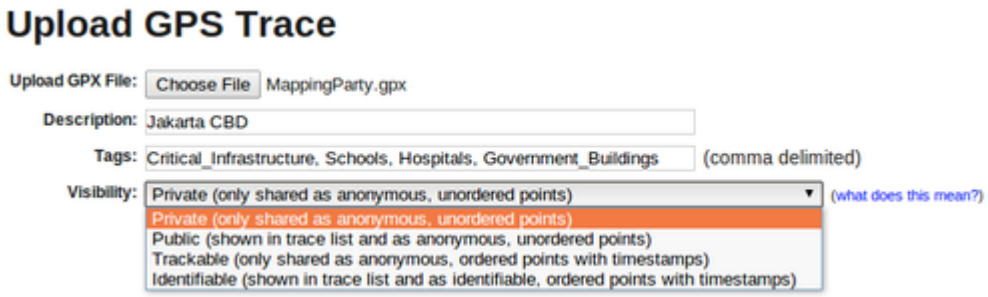

### 3.5.2 Subida De Trazas GPS Online

1. Desde la web http://www.openstreetmap.org/ hacer login.
2. Seleccionar «GPS Traces» en la columna situada a la izquierda del mapa.




3. Seleccionar upload a trace. Aquí también podrás **See your traces** (ver tus trazas) para revisar tus trazas GPS anteriores.
4. Selecciona tu archivo en «Choose File». Etiquetar en el campo *Description* (descripción),proporciona algunas *Tags* (etiquetas) y elige la visibilidad que tendrá. Si tienes muchos archivos .gpx puedes comprimirlos en un archivo ZIP y subirlo. Será tratada

como un único fichero GPX mayor y solo se creará una entrada en la lista de trazas.




5. Seleccionar *Upload* (subir). El archivo se subirá al servidor OSM, donde se pasará a la lista de archivos que esperan a ser insertados en la base de datos.

### 3.5.3 Desarrollo de un plugin

A continuación se detallan los pasos básicos a seguir en el desarrollo de un plugin:

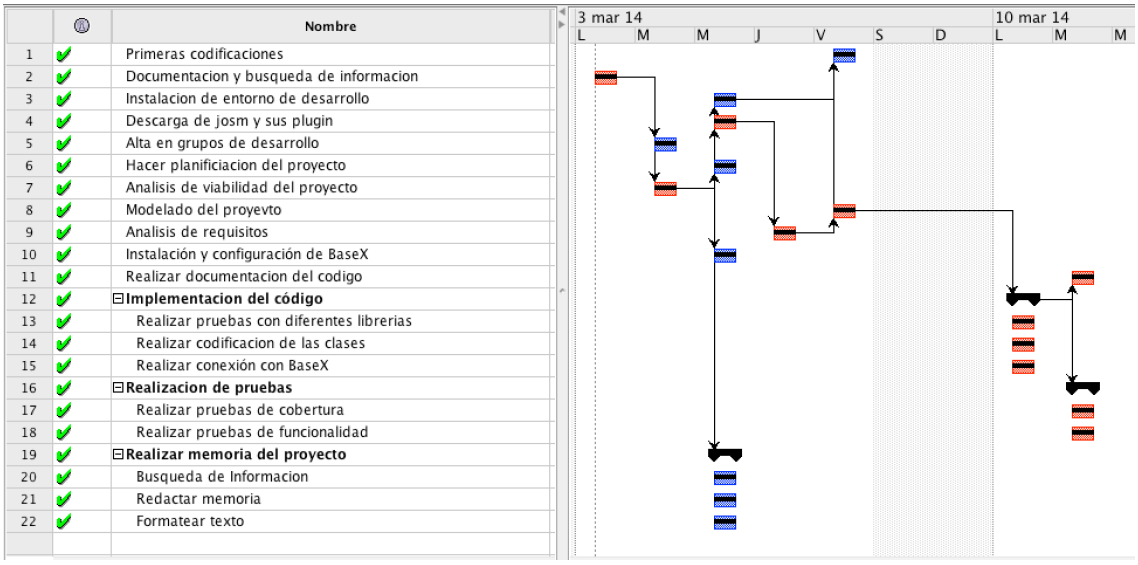
- Se creará un proyecto de plugin con un nombre p.e org.example.demo y la información del plugin se que va a introducir.
- Tras esto, se nos indicará si deseamos crear el plugin siguiendo una plantilla, escogeremos "Plugin with a view".
- A continuación se nos pedirán algunos datos para generar el código que creará el asistente.
- Examinar la descripción del plugin.
- Descargar.
- Crear un fichero en text/info.txt en el proyecto. Este fichero deberá añadirse como parte de la exportación en la sección Build Configuration.
- Exportar como "Deployable plugins and Fragments" e indicar que se genere un fichero Ant con nombre build-demo.xml.

- Copiar el plugin en la carpeta plugins de eclipse y comprobar que funciona.

# 4. Propuesta de desarrollo: JOSM-URBAN

## 4.1 Planificación del proyecto.

Para gestionar la planificación del proyecto se va a realizar un diagrama de Gantt que permita saber la estimación de tiempo y recursos.



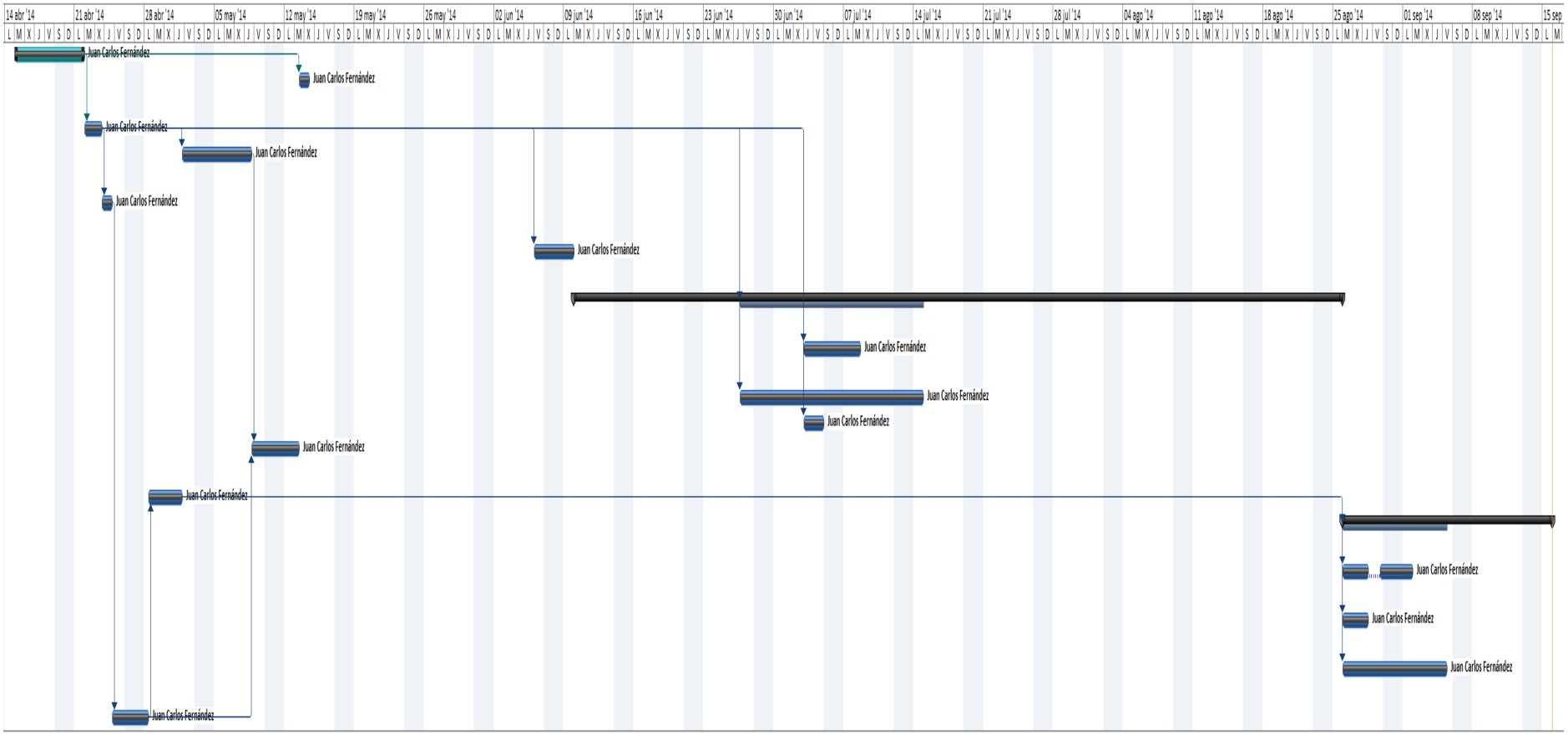
| | Nombre |
|---|---|
| 1 | Primeras codificaciones |
| 2 | Documentacion y busqueda de informacion |
| 3 | Instalacion de entorno de desarrollo |
| 4 | Descarga de josm y sus plugin |
| 5 | Alta en grupos de desarrollo |
| 6 | Hacer planificiacion del proyecto |
| 7 | Analisis de viabilidad del proyecto |
| 8 | Modelado del proyevto |
| 9 | Analisis de requisitos |
| 10 | Instalación y configuración de BaseX |
| 11 | Realizar documentacion del codigo |
| 12 | **Implementacion del código** |
| 13 | Realizar pruebas con diferentes librerias |
| 14 | Realizar codificacion de las clases |
| 15 | Realizar conexión con BaseX |
| 16 | **Realizacion de pruebas** |
| 17 | Realizar pruebas de cobertura |
| 18 | Realizar pruebas de funcionalidad |
| 19 | **Realizar memoria del proyecto** |
| 20 | Busqueda de Informacion |
| 21 | Redactar memoria |
| 22 | Formatear texto |



En esta imagen del diagrama de Gantt únicamente se muestran las tareas y las dependencias. Los tiempos estimados serán puestos en la siguiente imagen de diagrama de Gantt.

Para saber el esfuerzo que supone el proyecto a nivel de líneas de código se va a utilizar el método de estimación de puntos de función, ya que permite la comparación con otros proyectos.

## *Diagrama de Gantt*



**Estimación por puntos de función.**

Para realizar una estimación por puntos de función es necesario seguir una serie de pasos:

**1-Determinar tipo de cuenta.**

En este caso el tipo de cuenta es el de una aplicación en desarrollo.

**2-Identificar el alcance de la medición y los limites de la aplicación.**

En este caso consideraremos archivos internos cuando guardemos dentro de nuestro sistema de archivos, así como cualquier otra operación que realicemos nosotros y consideraremos agentes externos las consultas a JOSM o el envío de una capa para que se muestre en JOSM.

**3-Contar las funciones de datos.**

Para realizar el conteo de datos es necesario dividirlos en ILF (Internal logical) File EIF (External Interface File), y luego clasificarlo según su complejidad midiendo la cantidad de DET (Data Element Type) y RET (Record Element Type) que tienen, una vez calculado es necesario clasificarlos según la complejidad siguiendo la tabla de los puntos de función.

*ILF(Archivos Lógicos Internos)*

| Nombre Clase | Nº Registros Lógicos | Atributos | Clasificación |
|---|---|---|---|
| ControladorAnadirConsultas | 1 | 5 | Baja |
| ControladorConexion | 1 | 3 | Baja |
| ControladorConsultas | 1 | 5 | Baja |
| ControladorInfo | 1 | 6 | Baja |
| ControladorOpcionesConsultas | 1 | 5 | Baja |
| ControladorPreview | 1 | 7 | Baja |

| | | | |
|---|---|---|---|
| ControladorPrincipal | 1 | 13 | Baja |
| ControladorTutorial | 1 | 7 | Baja |
| ModeloConexion | 1 | 6 | Baja |
| ModeloConsultas | 1 | 5 | Baja |
| ModeloInfo | 1 | 3 | Baja |
| ModeloPreview | 1 | 3 | Baja |
| ModeloPrincipal | 1 | 3 | Baja |
| ModeloTutorial | 1 | 3 | Baja |
| VistaAnadirConsultas | 1 | 11 | Baja |
| VistaCentro | 1 | 6 | Baja |
| VistaConexion | 1 | 17 | Baja |
| VistaNorte | 1 | 6 | Baja |
| VistaOpcionesConsulta | 1 | 7 | Baja |
| VistaTutorial | 1 | 5 | Baja |

EIL(Archivos Logicos Externos)

| Nombre Clase | Nº Registros Lógicos | Atributos | Clasificación |
|---|---|---|---|
| ControladorConsultas | 1 | 4 | Baja |
| ControladorPrincipal | 1 | 2 | Baja |

**4-Contar las funciones Transaccionales.**

Para contar las transacciones es necesario saber la cantidad de operaciones EI (External Input), EO (External Output),EQ (External Query).

EI(Entradas)

| Nombre Clase | Nº operaciones | Atributos | Clasificación |
|---|---|---|---|
| ControladorAnadirConsultas | 2 | 3 | Baja |
| ControladorConexion | 1 | 5 | Baja |

| | | | |
|---|---|---|---|
| ControladorConsultas | 3 | 3 | Media |
| ControladorOpcionesConsultas | 1 | 1 | Baja |
| ControladorPreview | 1 | 1 | Baja |
| ModeloConexion | 1 | 5 | Baja |
| ModeloConsultas | 2 | 4 | Baja |

EO(Salidas)

| Nombre Clase | Nº operaciones | Atributos | Clasificación |
|---|---|---|---|
| ControladorConsultas | 1 | 1 | Baja |
| ModeloConexion | 1 | 5 | Baja |
| ModeloConsultas | 3 | 4 | Medio |
| ModeloPreview | 1 | 1 | Baja |
| VistaAnadirConsultas | 1 | 1 | Baja |

EQ(Consultas)

| Nombre Clase | Nº operaciones | Atributos | Clasificación |
|---|---|---|---|
| ControladorAnadirConsultas | 1 | 1 | Baja |
| ControladorConexion | 1 | 5 | Baja |
| ControladorConsultas | 2 | 6 | Medio |
| ControladorInfo | 1 | 3 | Baja |
| ControladorOpcionesConsultas | 1 | 1 | Baja |
| ControladorPrincipal | 4 | 3 | Medio |
| ControladorTutorial | 3 | 6 | Medio |
| ModeloConexion | 1 | 5 | Baja |
| ModeloConsultas | 3 | 4 | Medio |

| | | | |
|---|---|---|---|
| VistaCentro | 1 | 1 | Baja |
| VistaConexion | 1 | 1 | Baja |
| VistaSur | 1 | 1 | Baja |

**5- Determinar los puntos de función.**

Calculo de puntos de función totales

| | BAJA | MEDIA | ALTA | TOTAL |
|---|---|---|---|---|
| ENTRADAS | 7*3 | 1*4 | 0*6 | 25 |
| SALIDAS | 4*4 | 1*5 | 0*7 | 21 |
| CONSULTAS | 8*3 | 4*4 | 0*6 | 40 |
| FICH.LOGICOS | 20*7 | 0*10 | 0*15 | 140 |
| FICH.INTERFACES | 2*5 | 0*7 | 0*10 | 10 |

Dado que cada punto de función en Java corresponde a 55 líneas de código, y que en total tenemos 236 puntos de función lo que nos da aproximadamente de 12980 líneas de código.

## 4.2 Análisis de requisitos

JOSM-Urban es un plugin para el programa JOSM. El software permite la realización consultas XQuery sobre mapas urbanos, y tiene los siguientes requisitos funcionales y no funcionales.

### Requisitos funcionales:

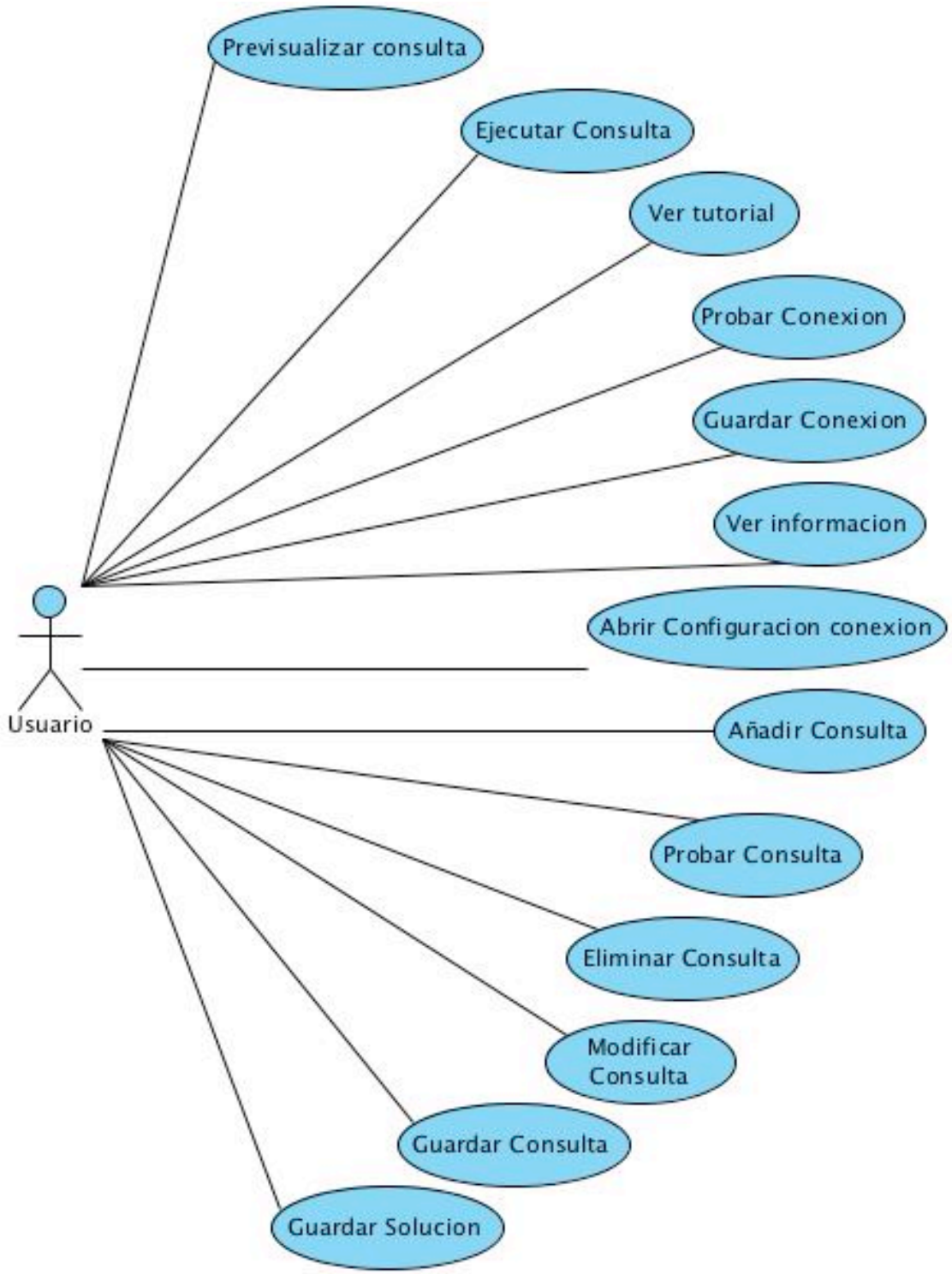
- Dado que el sistema se basa en la arquitectura cliente-servidor, tiene que **ser capaz de comprobar la conexión**.
- El sistema guarda la información de la conexión.
- La información de la conexión podrá ser modificada por el usuario mediante una ventana.
- Dentro del sistema tiene que existir un tutorial explicativo sobre el funcionamiento.
- Muestra la información explicativa acerca de la creación y funcionalidades.
- Permite la **previsualización de consultas**.
- Permite la **ejecución rápida de consultas** con la capa activa.
- Guarda las diferentes consultas realizadas.
- Visualiza en forma de lista las diferentes consultas.
- Permite **visualizar** las opciones de las consultas en una ventana a parte.
- Permite **eliminar** consultas.
- Permite **modificar el texto** de las consultas.
- Permite **añadir consultas** con un nombre.
- Al añadir consultas se puede probar la ejecución.
- Al realizar una ejecución se guarda automáticamente la solución en un archivo temporal.

### Requisitos no funcionales:

- El sistema se tiene que **programar en lenguaje Java**.
- Se requiere de un sistema que haga de servidor de BaseX.
- La interfaz gráfica se tiene que realizar utilizando Java Swing.
- La persistencia de las consultas y la configuración del servidor se realiza mediante archivos de texto.
- Se tiene que realizar la codificación siguiendo el patrón arquitectónico modelo vista controlador.
- Al diseñar la interfaz se tienen que seguir las directrices de usabilidad.
- El software tendrá un juego de pruebas las cuales serán de cobertura y funcionalidad.
- Se tiene que asegurar ejecución normal del programa evitando excepciones.

## 4.3 Modelado y diseño I

### 4.3.1 Diagramas de casos de uso



### 4.3.2 Diagramas de clases

Para una mejor presentación de los diagramas de clases se han dividido en grupos.

- **Diagrama de clases ControladorPrincipal, VistaPrincipal y ModeloPrincipal**

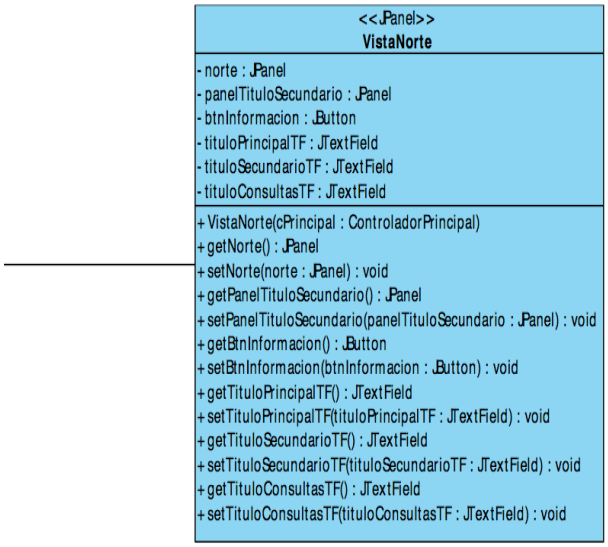
En la **Clase VistaPrincipal** se define el elemento gráfico principal, en este elemento se empotraran los demás componentes gráficos.

La **Clase ControladorPrincipal** se encarga de realizar los métodos imprescindibles del plugin, así como la inicialización del resto de componentes gráficos, modelos y controladores. Esta clase es la clase raíz o clase de partida del plugin.

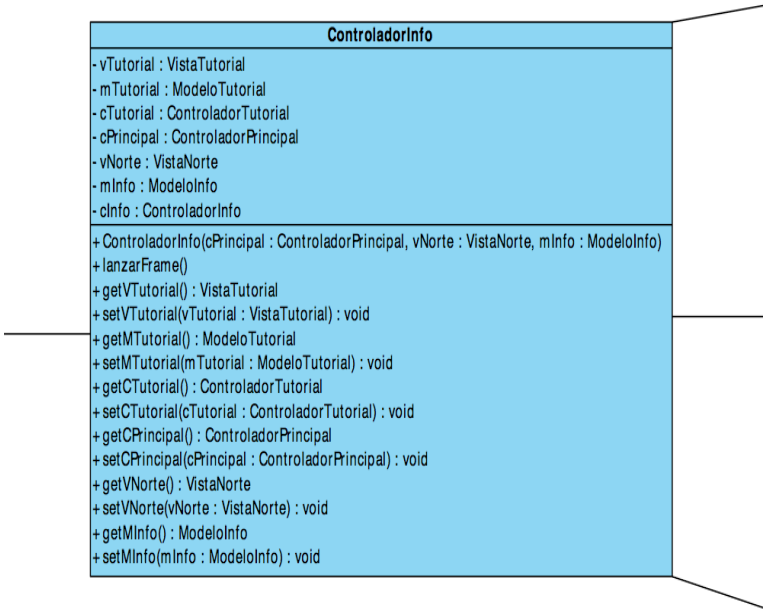
Mientras que en la **Clase Modelo principal** se asegura que exista y este correctamente creada la estructura de archivos del plugin, creando las diferentes carpetas y subcarpetas donde ira toda la configuración y soluciones.

## ☞ Diagrama de las clase VistaNorte, ControladorInfo y ModeloInfo

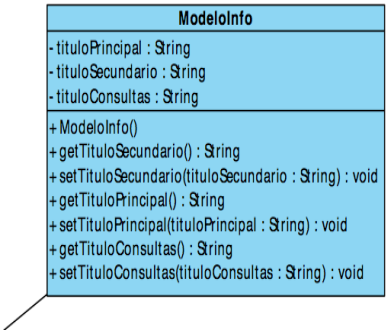
La **Clase VistaNorte** es la componente gráfica de la parte superior, esta parte contiene varios títulos del plugin, así como un botón que permite acceder a la información completa, cambio de la configuración de la conexión y acceso al tutorial.




La **Clase ControladorInfo** permite lanzar el siguiente frame con la información completa, el tutorial y la configuración de la conexión.

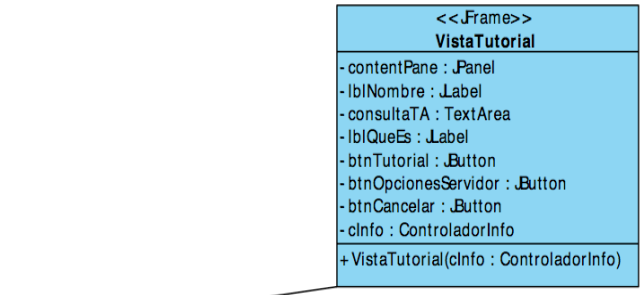



La **Clase ModeloInfo** se encarga de guardar la información que se mostrará en la clase VistaNorte.

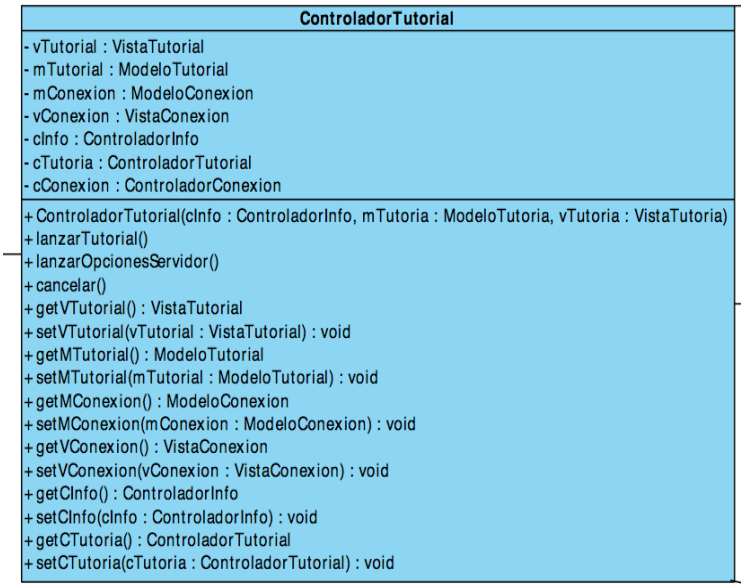


- **Diagrama de las clase VistaTutorial, ControladorTutorial y ModeloTutorial.**

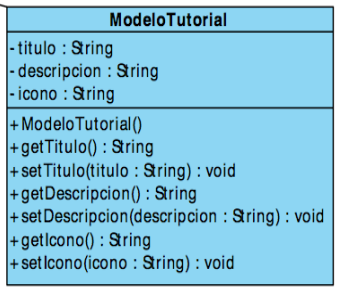
En la **Clase VistaTutorial** se muestra los componentes gráficos de la parte de Información y tutorial, además permite acceder a la configuración de la conexión.




En la **Clase ControladorTutorial** se gestionan las funcionalidades o procesos de la **VistaTutorial**, se encarga de lanzar el siguiente frame con la configuración de la conexión y de lanzar el tutorial.

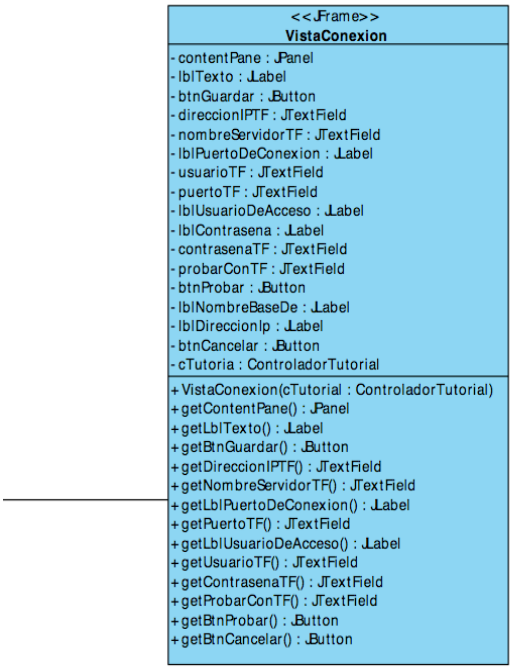



La **Clase ModeloTutorial** se encarga de gestionar la información que se muestra en la clase **VistaTutorial**.

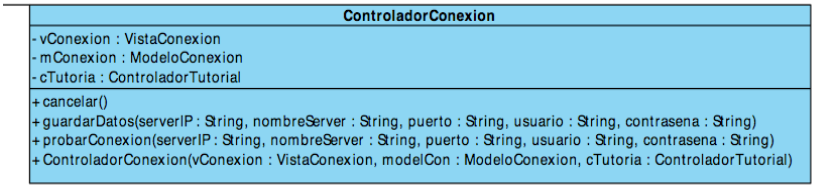


☞ **Diagrama de las clases VistaConexion, ControladorConexion y ModeloConexion.**

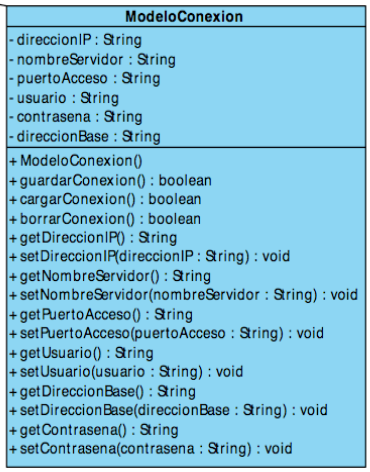
La **Clase VistaConexion** se encarga de mostrar toda la interfaz necesaria para probar la configuración de la conexión con el servidor y modificarla.




La **Clase ControladorConexion** se encarga de gestionar el procesamiento de la información, lanzar la prueba y cerrar la ventana de configuración.

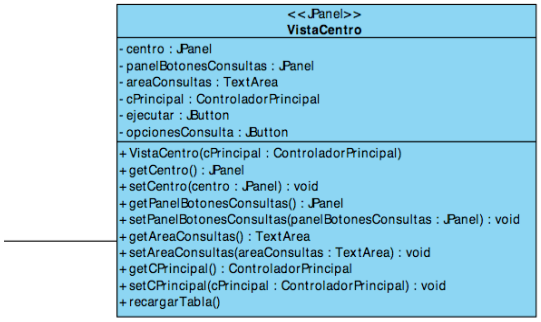



La **Clase ModeloConexion** se encarga de realizar la persistencia de la información que se muestra en la vista de forma que es capaz de recuperar y guardar los datos para no perderlos.

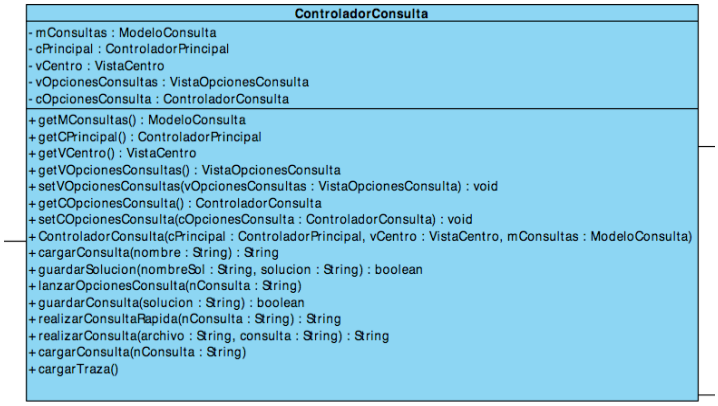


☞ **Diagrama de las clases VistaCentro, ControladorConsultas y ModeloConsultas.**

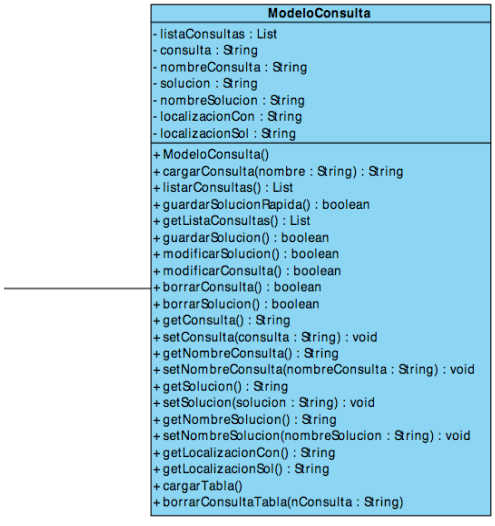
En la **Clase VistaCentro** se muestra la parte central de la interfaz gráfica principal, básicamente los botones de opciones de consulta y ejecutar y la tabla con las diferentes consultas.




En la **Clase ControladorPrincipal** se encarga de gestionar la funcionalidad la tabla, así como su carga y procesos, el lanzado de el frame de las opciones y la ejecución de la consulta.

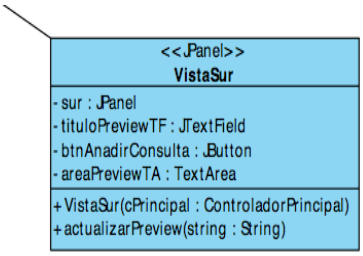



En la **Clase ModeloConsulta** se encarga de guardar los elementos que se encuentran en la vista, se encarga la persistencia de las consultas, realizar listas de consultas y persistencia de las soluciones.

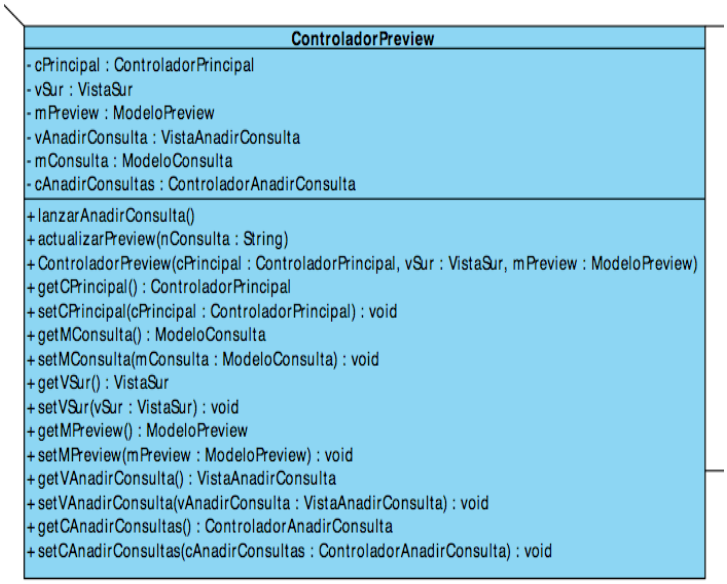


## ☞ Diagrama de las clases VistaSur, ControladorPreview, y ModeloPreview

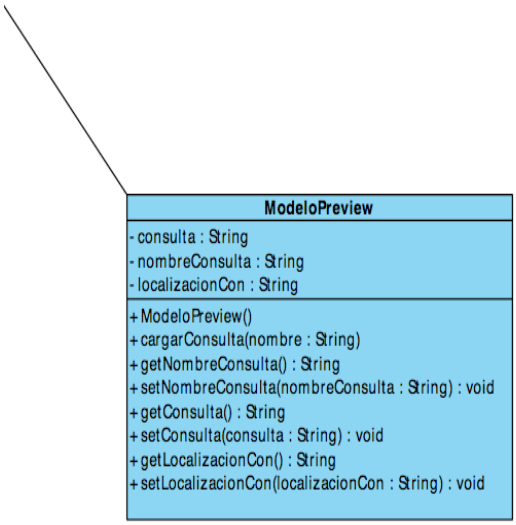
En la **Clase VistaSur** se encuentra la inferior de la interfaz gráfica principal. Esta parte se encarga de previsualizar el contenido.




La **Clase ControladorPreview** se encarga de realizar las gestiones y el procesamiento para lanzar el método de añadir consulta y actualizar la previsualización.

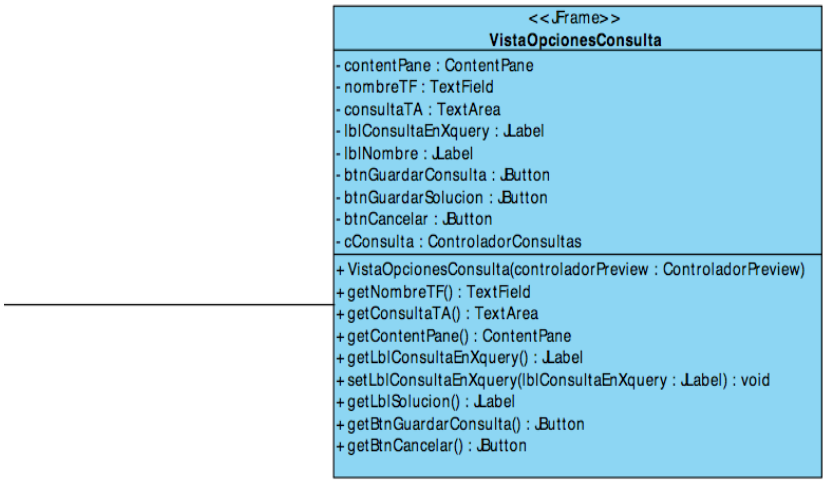


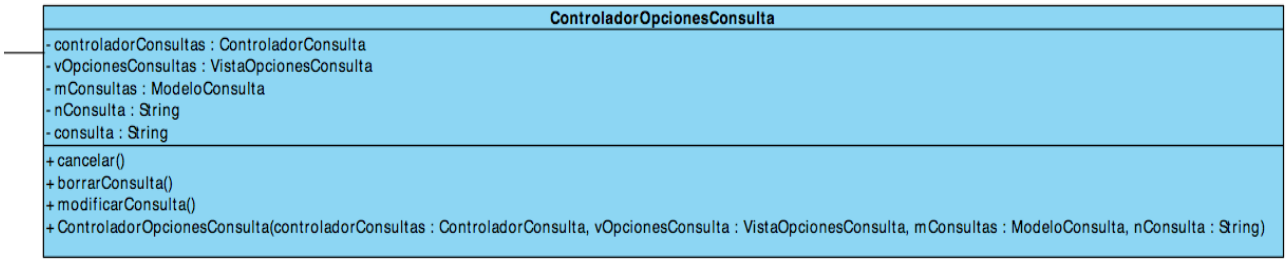

La **Clase ModeloPreview** es la encargada de la persistencia de los datos de la previsualización, y del cargado de datos para previsualizarlos.



### ☞ Diagrama de las clases VistaOpcionesConsulta y ControladorOpcionesConsulta

La **Clase VistaOpcionesConsulta** es la encargada de mostrar la interfaz gráfica que permite la modificar y eliminar una consulta.

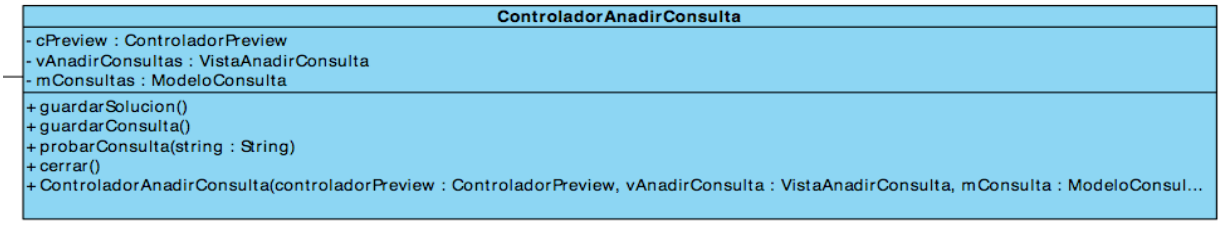


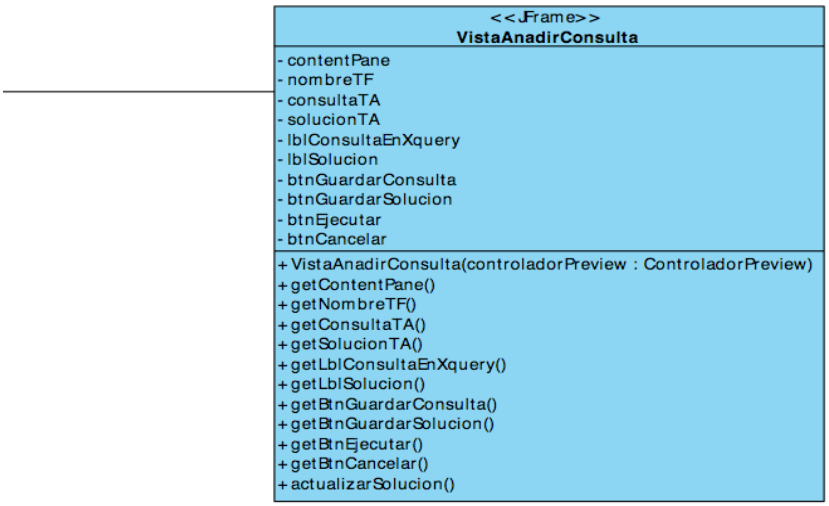




Y la **Clase ControladorOpcionesConsulta** es la encargada de gestionar la funcionalidad de la clase **VistaOpcionesConsulta**, además se encarga gestionar la funcionalidad de modificar y eliminar la consulta.

☞ **Diagrama de las clases ControladorAnadirConsulta y la clase VistaAnadirConsulta.**

La **Clase ControladorAnadir Consulta** es la encargada de procesar el guardado de la solución y la consulta, además de que permite probar la consulta.



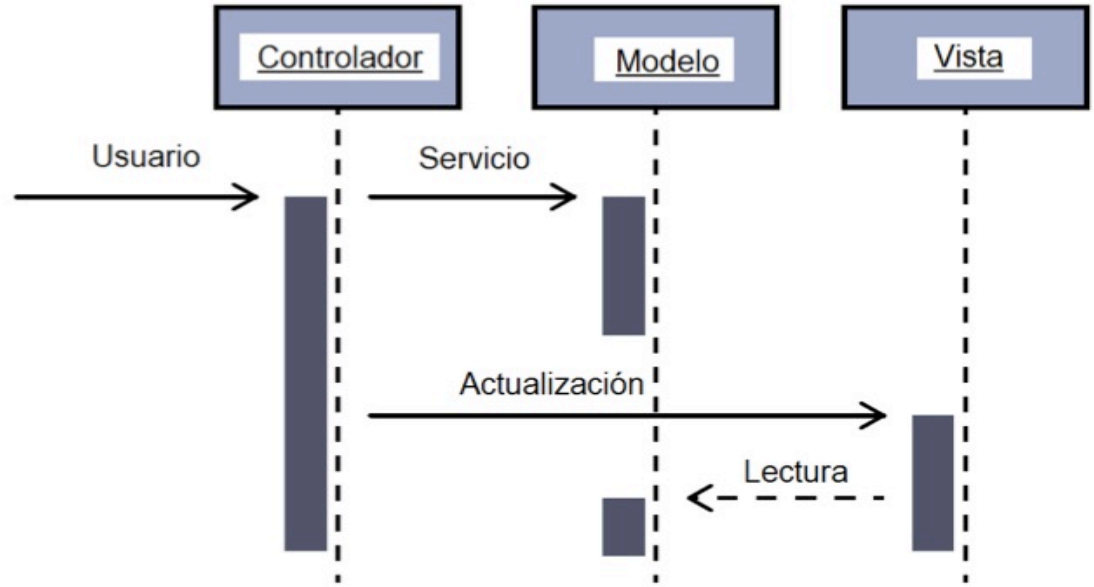




La **Clase VistaAnadirConsulta** muesta una interfaz gráfica que permite probar y guardar la solución/ consulta.

### 4.3.3 Diagramas de secuencias

Los diagramas de secuencias se encargan de modelar interacción entre objetos en un sistema. Dado que se ha seguido el patrón ModeloVistaControlador Pasivo, el cual se explica ampliamente punto 4.4.1.1, la mayoría de las interacciones vienen dadas por este patrón.

El diseño básico del mismo es el siguiente:




En este punto pondremos una serie de diagramas de secuencia de los casos de uso mas importantes.

Diagrama de secuencias de **Ejecutar Consulta**

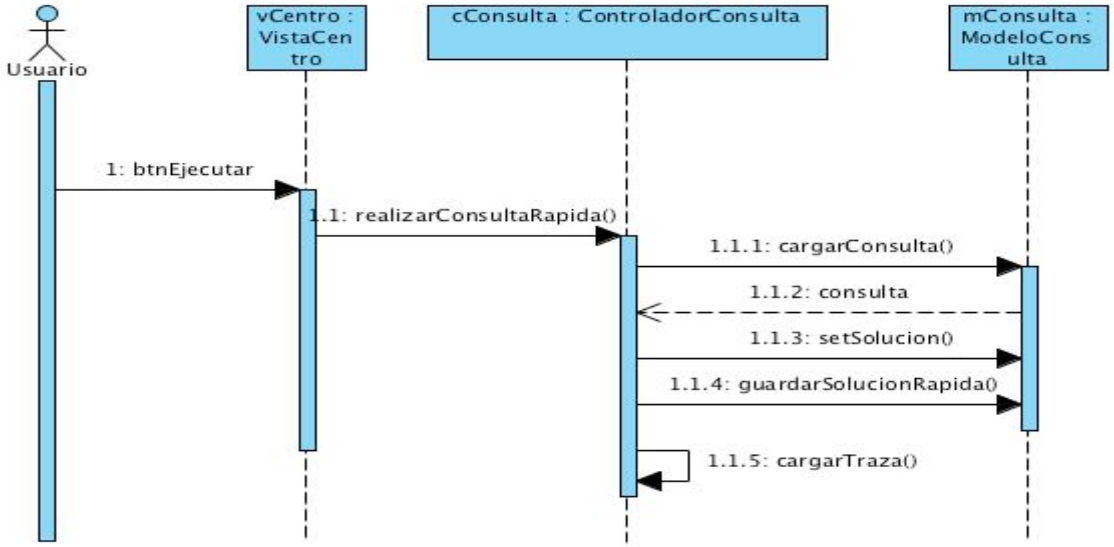



Diagrama de secuencias de **Probar Consulta**

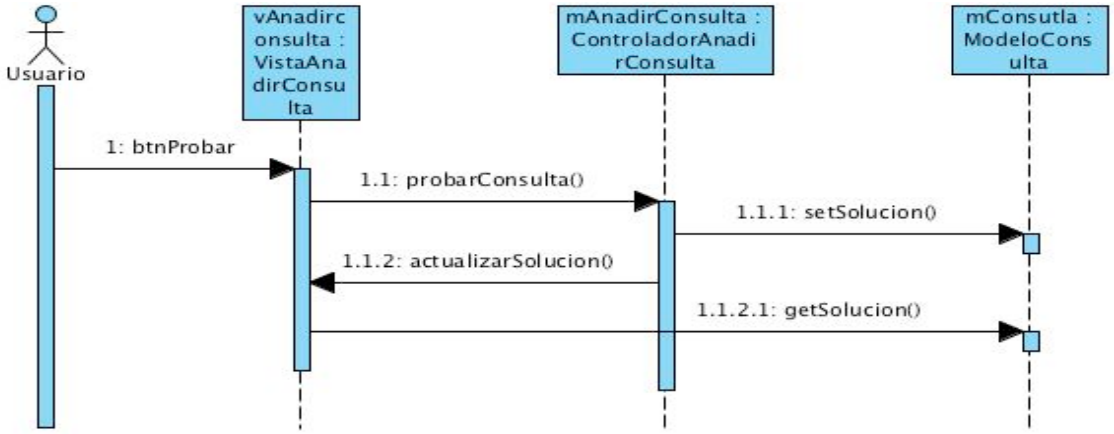



Diagrama de secuencias de **Eliminar Consulta**

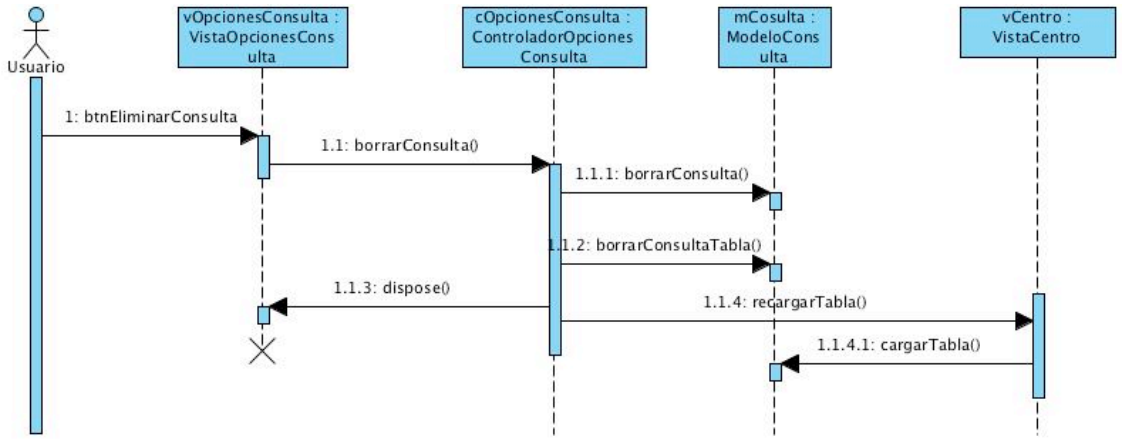


Diagrama de secuencias de **Previsualizar Consulta**

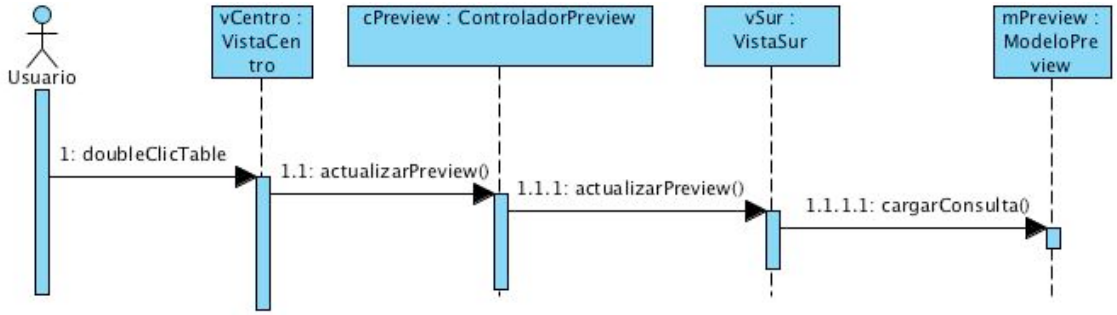
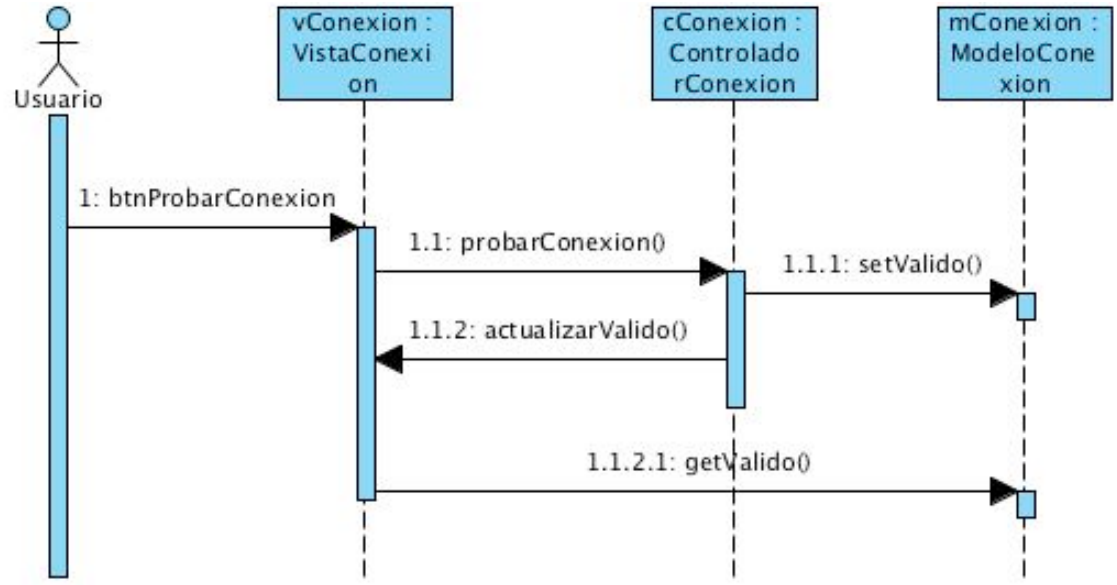



Diagrama de secuencias **Probar Conexión**



## 4.4 Implementación

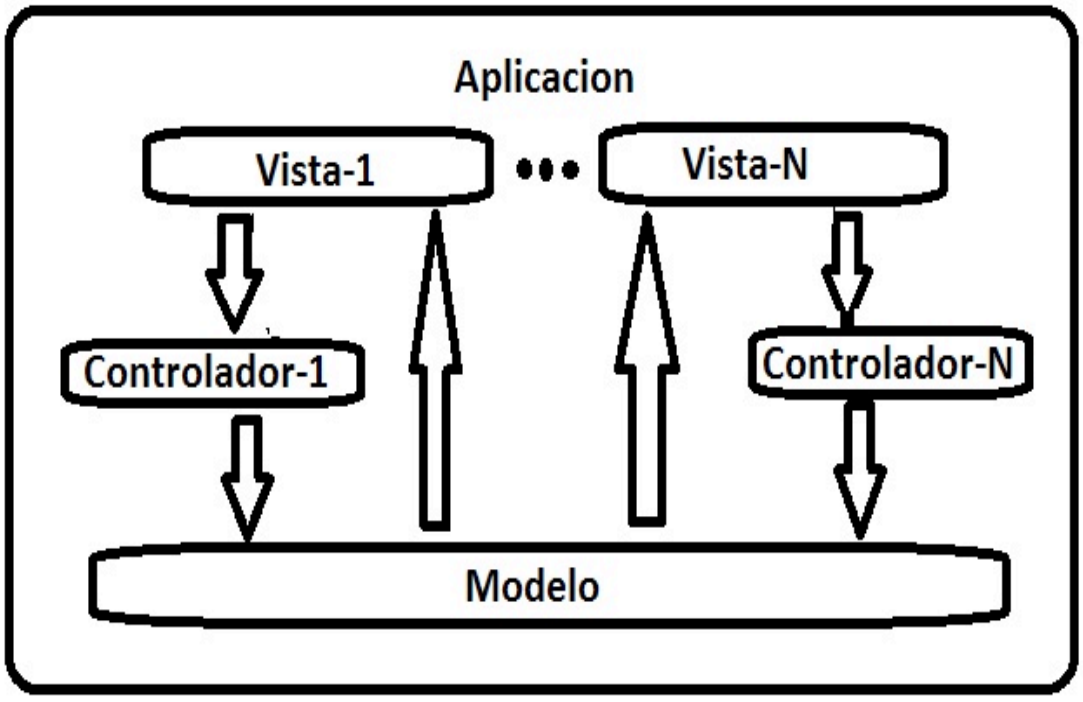
### 4.4.1 Fundamento Teórico

#### *4.4.1.1 Patrón arquitectónico Modelo Vista Controlador*

El **Modelo-Vista-Controlador (MVC) [RB04]** es un patrón de arquitectura de software que **separa** los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos:

- Modelo
- Vista y
- Controlador

El MVC es usado generalmente en aplicaciones web y la separación que se realiza en los distintos componentes (Modelo, Vista y Controlador) facilita su desarrollo, modificación y prueba.



#### *4.4.1.1.1 Vista*

Es la responsable de visualizar los datos y captar la interacción del usuario con la aplicación.

La vista obtiene sus datos del modelo, incluyendo las notificaciones de las actualizaciones realizadas en los datos y que necesitan ser refrescadas.

Una vista está asociada a un modelo, pudiendo existir varias vistas asociadas al mismo modelo.

Cuando el usuario interactúa con la vista a través de los dispositivos de entrada, entonces la vista es la responsable de captar el efecto de dicha interacción y enviarla al controlador para su procesamiento.

#### *4.4.1.1.2 Controlador*

Es el responsable de tomar las entradas del usuario y remitirlas al modelo para su procesamiento.

El controlador es un objeto que dirige el flujo del control de la aplicación debido a mensajes externos, como datos introducidos por el usuario u opciones del menú seleccionadas por él.

A partir de estos mensajes, el controlador se encarga de modificar el modelo o de abrir y cerrar vistas. El controlador tiene acceso al modelo y a las vistas, pero las vistas y el modelo no conocen de la existencia del controlador.

El principal beneficio es que el controlador proporciona el aislamiento del código (componente externo) que puede ser testeado automáticamente.

#### *4.4.1.1.3 Modelo*

El modelo es la representación en memoria de los datos que se han obtenido desde el almacenamiento persistente. El modelo es un conjunto de clases que representan la información del mundo real que el sistema debe procesar.

El modelo también es responsable de notificar a la vista los cambios en el estado.

La abstracción del modelo en esta forma, permite compartir fácilmente el estado de la sesión entre varias vistas. **El modelo desconoce la existencia de las vistas y del controlador.**

#### *4.4.1.1.4 MVC diseñado por capas empresariales.*

Las aplicaciones del ámbito empresarial generalmente están separadas en 3 capas lógicas. Esta lógica permite que las aplicaciones sean escalables, permitiendo su implementación en distintos servidores.

- **Capa de presentación**

La capa de presentación es responsable de visualizar datos, proporcionar información al usuario y captar la interacción con el usuario la cual se pasa a la capa de lógica de negocio para el procesamiento.

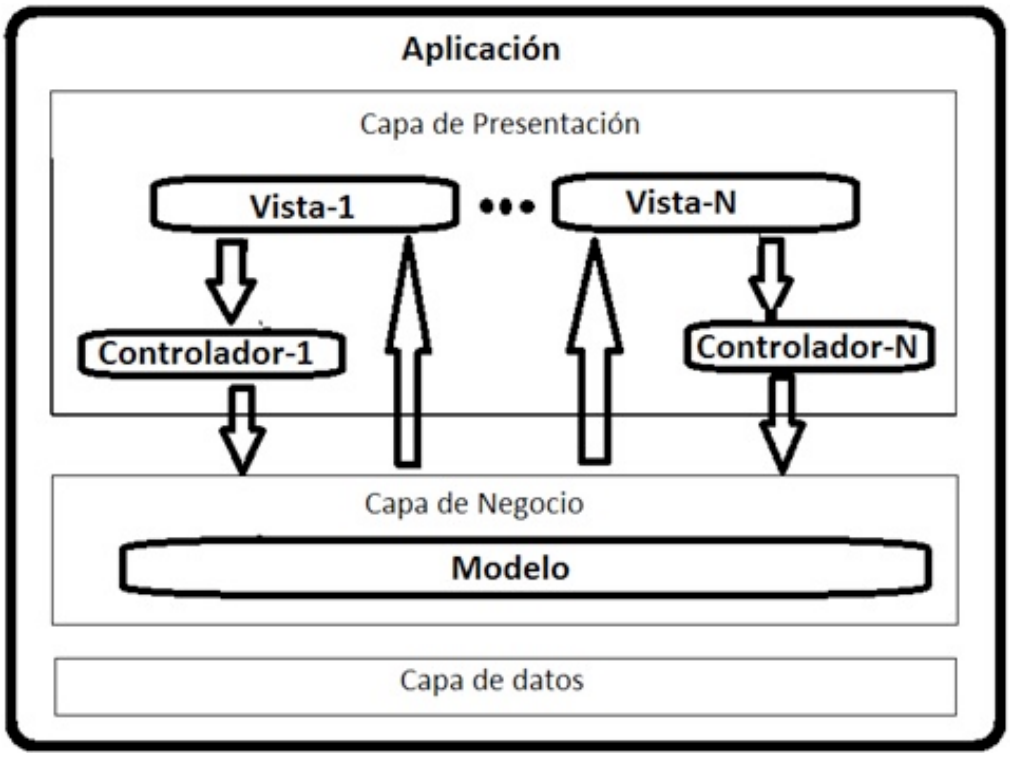
- **Capa de datos**

La capa de datos es responsable de las operaciones de entrada/ salida de los datos en el sistema de almacenamiento de datos (base de datos). Al tener una capa de datos podemos permitir cambios en el almacenamiento de datos sin tener que modificar las capas superiores.

### Capa de negocio

En la capa de negocio es donde reside la funcionalidad básica del sistema. La lógica que sigue esta capa es la lógica de negocio y esta capa tiene como función procesar los datos que se descarguen de la capa de acceso.

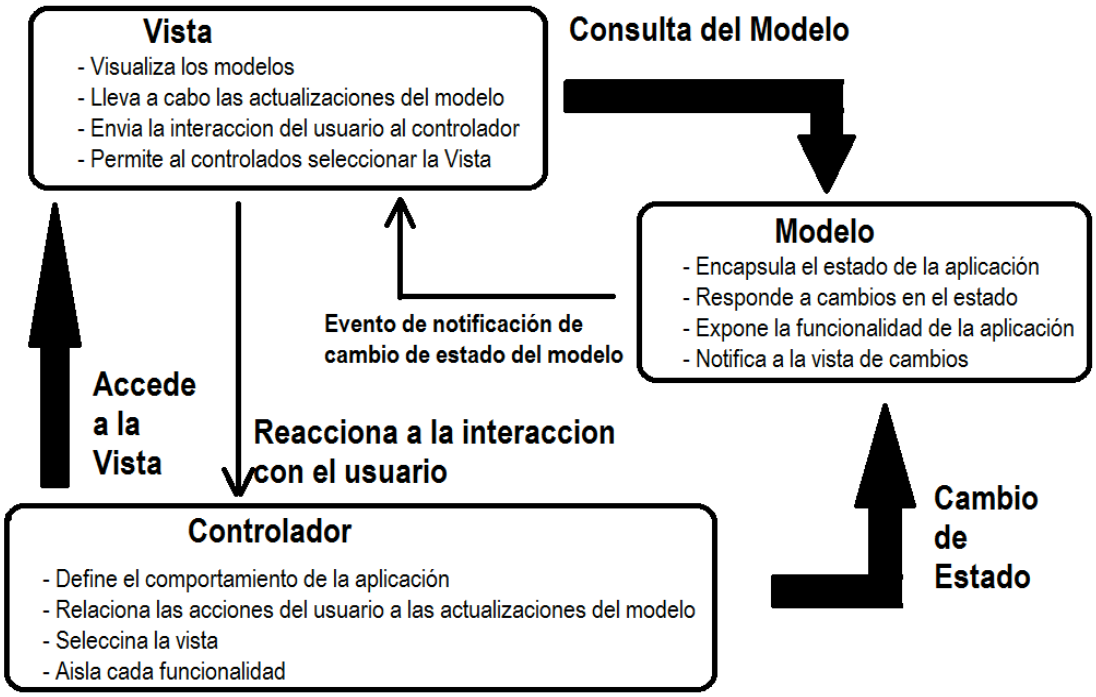
Al tener una capa de lógica de negocio en una aplicación, podemos escalar la aplicación de forma que permite que se puedan guardar por separado las diferentes capas, esto proporciona la flexibilidad necesaria para soportar múltiples tipos de interfaces de usuario y múltiples tipos de almacenes de datos.




Si utilizamos el patrón de diseño MVC en un diseño con capas, la vista y el controlador formaran parte de la capa de presentación y el modelo de la capa de la lógica de negocio.

#### 4.4.1.1.5 Ventajas, problemas y soluciones

- Los datos del modelo se pueden representar en distintos formatos (HTML, XML, CVS …) -> Para solucionarlo se puede separar la funcionalidad interna de la aplicación de la presentación y de la lógica de control que usa dicha funcionalidad, de forma que el formato de los datos no afecte a la funcionalidad.
- Los mismos datos del modelo se pueden actualizar a través de diferentes interacciones desde el modelo -> Se soluciona permitiendo que las vistas involucradas compartan el mismo modelo.
- El hecho de que una aplicación soporte múltiples tipos de vistas e interacciones no debería tener impacto sobre su funcionalidad básica -> Hacer que la aplicación pueda soportar múltiples clientes de una manera más fácil de implementar, testear y mantener.



**Algunos aspectos a destacar de la arquitectura MVC son:**

Ni el Modelo, ni la Vista, ni el Controlador son una sola clase, estos se pueden componer de diferentes clases (ya sea como proyectos diferentes o como conjuntos funcionales de clases dentro de un mismo proyecto).

El Controlador contiene al Modelo y a la Vista (relación de composición) La Vista envía una notificación al Controlador de los cambios que ocurren para que tome las acciones correspondientes. La notificación es transparente, es decir, la Vista no conoce al Controlador .

El Modelo lo actualiza el Controlador y puede notificar a la Vista sobre los cambios producidos para que ésta se actualice. El Modelo no conoce ni al Controlador ni a la Vista.

Dado que el MVC es un patrón de arquitectura [RW32], este define un esquema organizativo del sistema y no es muy estricto en la forma de implementar esta organización, por lo que se pueden encontrar múltiples variaciones e interpretaciones del mismo. El punto más importante es separar los tres elementos (Modelo, Vista y Controlador).

**Ventajas**

- Se puede tener diferentes vistas para un mismo modelo.
- Permite construir nuevas vistas sin necesidad de modificar el modelo subyacente.
- Proporciona un mecanismo de configuración para componentes complejos mucho más tratable que el puramente basado en eventos.
- Para que la vista se entere de los cambios producidos en el modelo, se puede utilizar el patrón Observer, de forma que la vista se declare como oyente y refleje los cambios del modelo.

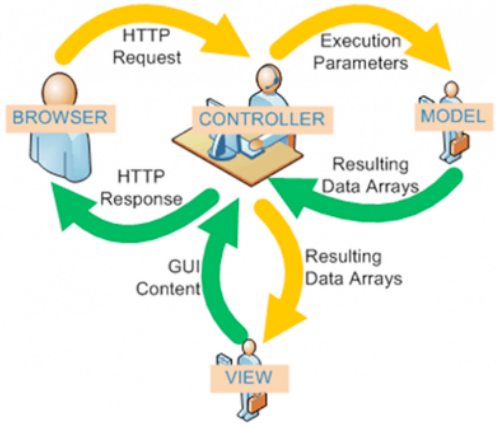
## Requisitos de la reusabilidad

El Modelo no debe de poder ver a ninguna clase de los otros grupos, por lo que se podría cambiar de Vista y Controlador sin tocar el modelo.

El Controlador debe ver las clases del Modelo, pero no de la Vista, por lo que el cambio de Vista no afecta al Controlador.

En algunas implementaciones de la arquitectura [RW31] el controlador puede ver a la Vista por si alguna acción del Controlador afecta a la Vista pero no al Modelo (por ejemplo, un mensaje de error).

La Vista no debe poder ver las clases del Modelo de forma que el cambio del Modelo no afecte a la Vista.



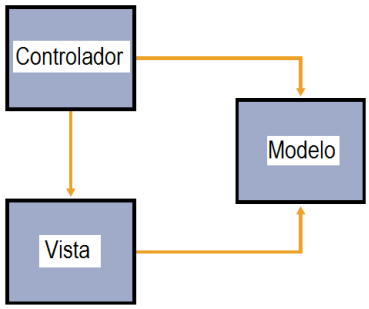

En algunas implementaciones la Vista ve al Modelo para consultarle información, pero nunca para realizar cambios en él.

Dentro del Modelo vista controlador existen diferentes formas de implementarlo, normalmente se suelen elegir diseños de implementación que se adapten a la aplicación y al sistema, pero principalmente hay dos vertientes de implementación:

- MVC (pasivo)
- MVC (activo)

### *4.4.1.1.6 MVC (pasivo)*

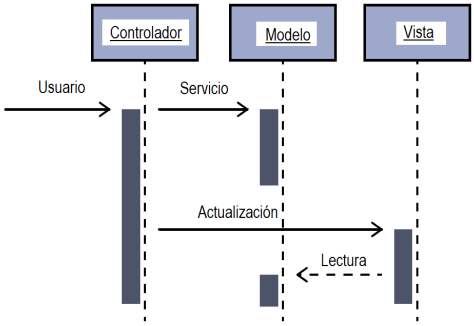
**Características más importantes del MVC pasivo.**




El **Modelo** no sabe de la existencia de la Vista ni del Controlador, por lo que se puede verificar en otro contexto separado.

La **Vista** y el **Controlador**, si pueden ver al Modelo, ya que la Vista tiene que poder acceder al modelo para visualizar los datos que han cambiado, y el Controlador tiene que reflejar las acciones del usuario sobre los datos del Modelo.

El **Controlador** es el único que manipula los datos del Modelo, y cuando lo hace le informa a la Vista de que el Modelo lo ha cambiado y tiene que acceder a él para actualizar su correcta visualización.




**Ventajas e Inconvenientes del MVC (pasivo)**

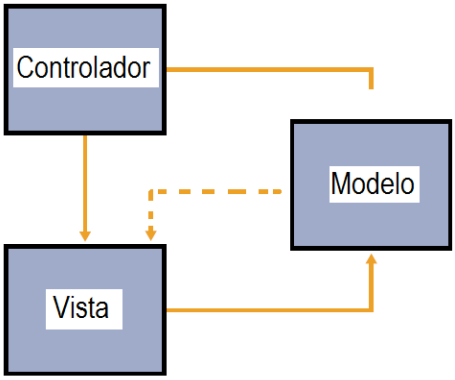
- Es conveniente usarlo para proyectos Web, dado que permite tener el **mismo modelo bajo diferentes vistas**, además de la posibilidad de modificar una vista concreta sin afectar al modelo.

- Esta arquitectura es adecuada para un **desarrollo dirigido por test** (TDD), ya permite que se pueda probar con casos negativos.

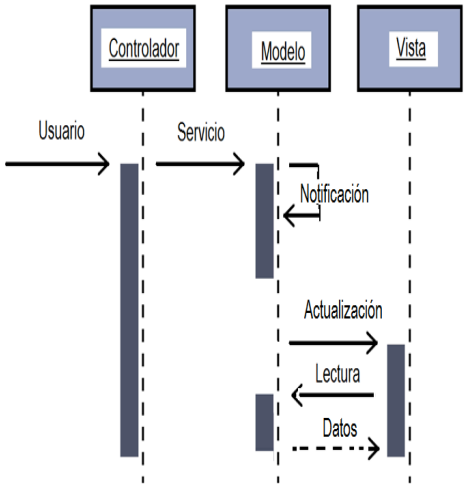
- Dado que la arquitectura está orientada a eventos, el controlador consume gran cantidad de recursos al reaccionar a los eventos.

#### *4.4.1.1.7 MVC (activo)*

El MVC activo se caracteriza por el cambio de estado sin la intervención del Controlador, por lo que solo el modelo es capaz de detectar los cambios y deberá notificar a la vista para que actualice.




Esto introduce un problema, el modelo debería de conocer la vista, para notificar el cambio, por lo que para poder solucionar este problema se introduce el patrón Observer.




Dicho patrón consiste en un mecanismo de alerta para los objetos sin introducir una dependencia entre ellos.

### Ventajas del MVC (Activo)

- Dado que la vista se comunica con el modelo mediante el patrón Observer existe un menor acoplamiento.
- Existe una mayor cohesión dado que Cada elemento del patrón está especializado en su tarea .
- Al cambiar el controlador se puede cambiar el modo en que una vista responde al usuario sin cambiar su representación visual.
- Las vistas pueden especializarse en mostrar en diferentes aspectos del modelo.
- Mayor facilidad para el desarrollo múltiples dispositivos y canales.
- El MVC permite una mayor facilidad de mantenimiento y mayor escalabilidad.

#### *4.4.1.2 Arquitectura Cliente – Servidor*

La arquitectura cliente servidor [RW33] consiste en que un programa cliente que realiza peticiones a otro programa servidor el cual le da una respuesta. Esta idea se puede aplicar a programas que se ejecutan sobre un único equipo aunque es más útil en un sistema operativo multiusuario distribuido a través de una red.

La interacción cliente-servidor es el soporte de la mayor parte de la comunicación por redes.

##### *4.4.1.2.1 Fundamento*

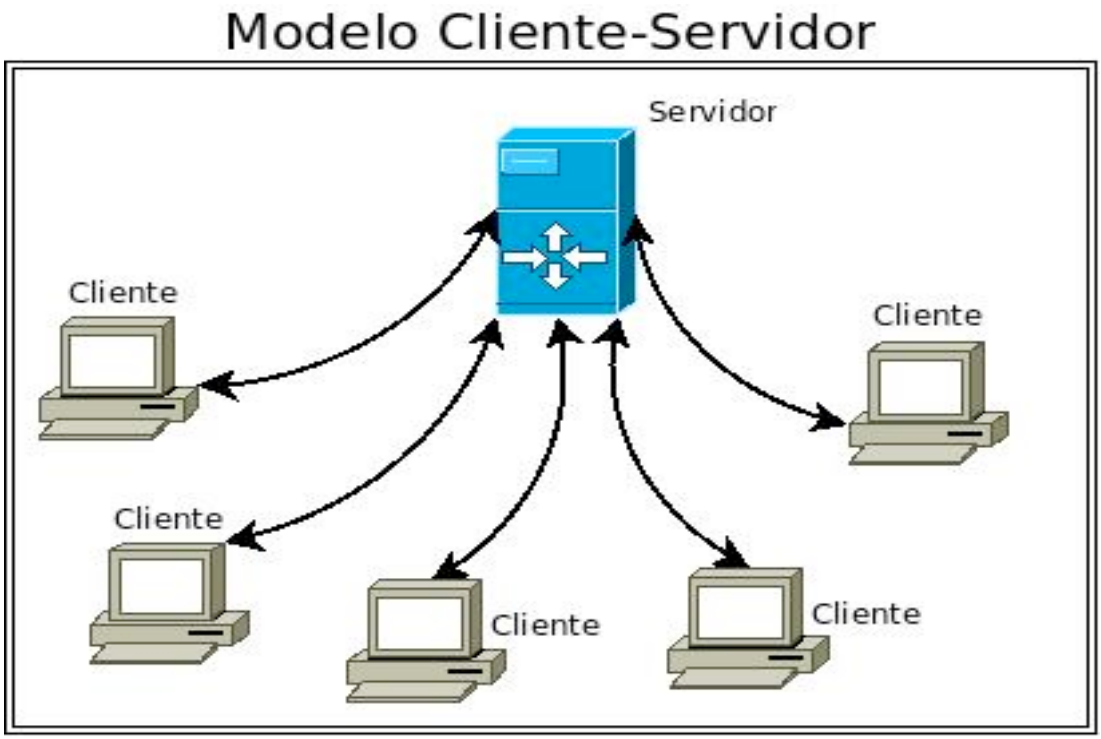
La arquitectura cliente-servidor [RW34], funciona de la siguiente manera. El servidor debe gestionar junto con su Sistema Operativo un puerto (casi siempre bien conocido) donde esperar las solicitudes. El servidor espera pasivamente las peticiones en un puerto conocido que haya sido reservado para el servicio que ofrece.

El cliente necesita dos puertos para la comunicación, solicita, a su sistema operativo, un puerto que no esté en uso desde el cual enviar su solicitud y esperar respuesta. Y luego utiliza otro puerto cualquiera dentro del rango de puertos disponibles y no reservados por el sistema o aplicaciones.

Desde el punto de vista del cliente la interacción necesita reservar solo uno de los dos puertos, los cuales están asignados con identificador único de puerto para cada servicio.

Los servidores por lo general son más difíciles de construir que los clientes, dado que deben de manejar diferentes peticiones de múltiples usuarios, manteniendo la integridad del sistema y evitando problemas de consistencia en el acceso a datos. El cliente y el servidor pueden interactuar en la misma máquina.

#### *4.4.1.2.2 Partes que componen el sistema*




**Cliente:** Es el programa ejecutable que participa activamente en el establecimiento de las conexiones. Envía una petición al servidor y se queda esperando por una respuesta. Su tiempo de vida es finito una vez que son servidas sus solicitudes, termina el trabajo.

**Servidor:** Es un programa que ofrece un servicio que se puede obtener en una red. Es el encargado de aceptar las peticiones, realizar el servicio y devolver el resultado al solicitante.

Usualmente se encuentra como aplicaciones, y puede ejecutarse en cualquier sistema donde exista TCP/IP junto con otros programas.

El servidor comienza su ejecución antes de comenzar la interacción con el cliente. Su tiempo de vida o de interacción es "interminable". Los servidores pueden ejecutar diferentes tipos de tareas.

Los servidores sencillos procesan una petición a la vez, por lo que no revisan si ha llegado otra petición antes de enviar la respuesta de la anterior.

Los servidores complejos trabajan con peticiones concurrentes aún la petición tarde mucho tiempo en ser resuelta.

La forma de construir dichos programa servidor es más compleja ya que tienen necesitan altos niveles de protección y autorización. Los servidores complejos tienen que preprocesar las solicitudes que envía el cliente y comprobar si tiene autorización para realizar dicha operación.

Los servidores por lo general tienen dos partes:

I. Programa o proceso que es responsable de aceptar nuevas peticiones: **Maestro o Padre.**
II. Programas o procesos que deben manejar las peticiones individuales: **Esclavos o Hijos**.

#### *4.4.1.2.3 Tareas*

**Tareas del programa maestro**

- Abrir un puerto local conocido al cual pueda acceder los clientes.
- Esperar las peticiones de los clientes.
- Iniciar un programa esclavo que atienda la petición en el puerto local, el esclavo cuando termina de manejar una petición no se queda esperando por otras.
- Volver a la espera de peticiones mientras los esclavos, en forma concurrente, se ocupan de las anteriores peticiones.

**Características de la arquitectura Cliente-Servidor**

- La arquitectura combina un cliente que interactúa con el usuario, y un servidor que interactúa con los recursos a compartir. El proceso del cliente proporciona la interfaz entre el usuario y el resto del sistema. El proceso del servidor actúa como un motor de software que maneja recursos compartidos tales como bases de datos, impresoras, Módem, etc.
- Las tareas del cliente y del servidor tienen diferentes necesidades en cuanto a recursos.
- Se establece una relación entre procesos distintos, dichos procesos son los procesos clientes y los procesos servidores, los cuales pueden ser ejecutados en la misma máquina o en máquinas diferentes distribuidas a lo largo de la red.
- La relación establecida puede de distinta carnalidad, aunque la más normal es la de muchos a uno, en la que un servidor puede dar servicio a muchos clientes, regulando su acceso a los recursos compartidos.

- Los clientes responden a procesos en los que han realizado peticiones. Esto tiene un carácter pasivo, ya que esperan peticiones de los clientes.
- Solamente existe relación entre clientes y servidores a través del intercambio de mensajes entre ambos. El mensaje es el mecanismo para la petición y entrega de solicitudes de servicios.
- La plataforma de hardware y el sistema operativo del cliente y del servidor no son siempre los mismos. Permitiendo que conectar clientes y servidores independientemente de sus plataformas.
- El concepto de escalabilidad tanto horizontal como vertical es aplicable a cualquier sistema Cliente-Servidor. La escalabilidad horizontal permite agregar más clientes sin afectar significativamente el rendimiento. La escalabilidad vertical permite mejorar las características del servidor o agregar múltiples servidores.

#### *4.4.1.2.4 Ventajas y desventajas de esta arquitectura*

En este punto mostraremos las ventajas e inconvenientes [RW38] de esta arquitectura .

**Ventajas del esquema Cliente-Servidor**

Permite dar soluciones de bajo coste uniendo varios servidores para dar servicios a varios clientes. Este esquema favorece la reducción de costos y la flexibilidad.

- Facilita la integración entre sistemas diferentes y compartir información.
- La estructura inherentemente modular facilita además la integración de nuevas tecnologías y el crecimiento de la

infraestructura computacional, favoreciendo así la escalabilidad de las soluciones.

- Facilita el procesado de la información ya que normalmente el cómputo se realiza desde el servidor.

**Desventajas**

- El mantenimiento de los sistemas es más difícil pues implica la interacción de diferentes partes de hardware y de software.
- Cuenta con muy escasas herramientas para la administración y ajuste de los sistemas.
- Es importante que los clientes y los servidores utilicen el mismo mecanismo, lo cual implica que se deben tener mecanismos generales que existan en diferentes plataformas.
- Es necesario tener estrategias para el manejo de errores y para mantener la consistencia de los datos, dada la concurrencia de usuarios.
- Una arquitectura cliente-servidor es susceptible de saturarse, ya sea por falta de cómputo del servidor, como por el colapso de la red.

### *4.4.1.3 Guía de diseño*

#### *4.4.1.3.1 Definición*

*Es la disciplina relacionada con el diseño, evaluación e implementación de sistemas informáticos interactivos para el uso de seres humanos, y con el estudio de los fenómenos más importantes con los que está relacionado.*

La disciplina de Interacción Persona-Ordenador (IPO) [RW39] se conoce en la comunidad internacional como *Human-Computer Interaction* (HCI).

**La interfaz de usuario**

La interfaz es una superficie de contacto entre dos entidades. En la interacción persona-ordenador estas entidades son la persona y el ordenador.

*La interfaz es el sitio donde los bits y las personas se encuentran*.

La interfaz refleja las cualidades físicas de las dos partes de la interacción. Esta idea en el diseño puede concretarse en dos conceptos:

- **Visibilidad:** para poder realizar una acción sobre un objeto ha de ser visible.
- **Comprensión intuitiva**, o propiedad de ser evidente la parte del objeto sobre la que hemos de realizar la acción y como hacerlo.

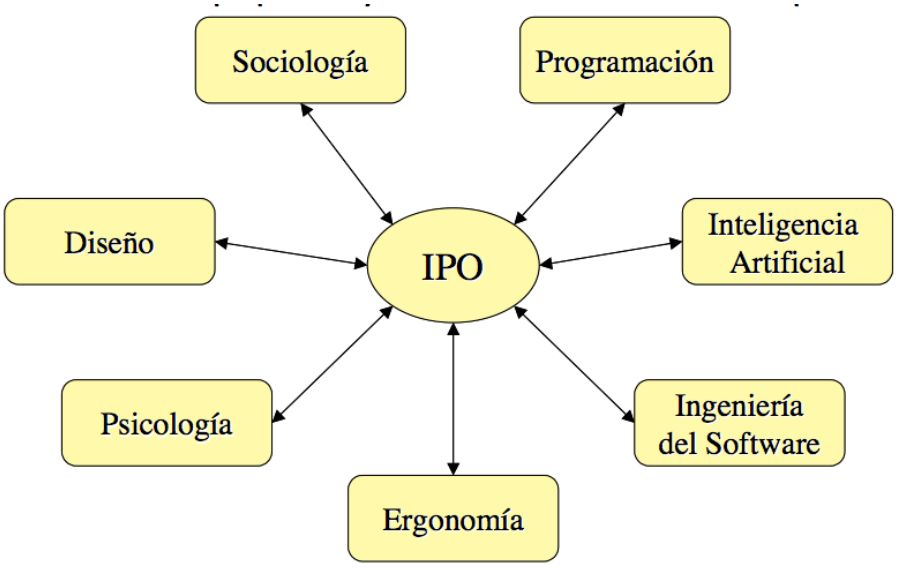
**Este principio se conoce como *affordance* .**

Esta interfaz deberá ser visible y de comprensión intuitiva .

## Diseño de interfaz

Para poder diseñar interfaces, además del aspecto informático, hace falta tener en cuenta otras disciplinas.

Necesitamos trabajar los aspectos psicológicos del usuario, la ergonomía del equipamiento, los aspectos sociales, temas de diseño…




Principales disciplinas que contribuyen en el diseño de la interfaz.

## Principales disciplinas en el diseño de la interfaz

- **Psicología**

La psicología es la ciencia que estudia el comportamiento y los estados de la conciencia de la persona humana, considerada individualmente o bien al mismo tiempo como miembro de un grupo social.

La psicología contribuye a la IPO mediante conocimientos y teorías acerca de como los sujetos se comportan, procesan la información y actúan en grupos y organizaciones, proporcionando una manera de comprobar que la interfaz es efectiva.

- **Diseño**

Actividad encaminada a conseguir la producción en serie de objetos útiles y bellos. Tal como se entiende actualmente, pretende actuar sobre el entorno físico del hombre con tal de mejorarlo en su conjunto.

- **Sociología**

Es la ciencia que estudia las costumbres y las tradiciones de los pueblos. La observación detallada, entrevistas sutiles, documentación sistemática, pueden responder a aquellas cuestiones sobre organizaciones y mercados que otros métodos no pueden.

- **Ergonomía o factores humanos**

Su propósito es definir y diseñar herramientas y artefactos para diferentes tipos de ambiente: trabajo, descanso y doméstico.

Su objetivo es maximizar la seguridad, la eficiencia y la fiabilidad para simplificar las tareas e incrementar la sensación de confort y satisfacción.

Las radiaciones de las pantallas, por ejemplo han sido un tema de trabajo importante en los últimos años.

Un ordenador por sí solo no es capaz de hacer nada, sino que solo puede hacer lo que se la ha indicado, o mejor dicho, programado que haga.

La programación es la herramienta que nos permite ***"decirle al ordenador lo que debe hacer".***

Un programa:

Debe funcionar --- no debe tener dificultades .

Debe estar bien documentado ---- debe ser eficiente.

## Tipos de programación

Básicamente podemos clasificar los tipos de programación en cinco grandes grupos:

- **ORIENTADA a OBJETOS.** La programación Orientada a Objetos (POO) está basada en los conceptos de Clases y Objetos.
- **IMPERATIVA (Entrada, procesamiento y salidas de Datos):** Es llamada así porque está basada en comandos que actualizan variables que están en almacenamiento. Están *orientados a la arquitectura del computador*.
- **FUNCIONAL:** Los datos son funciones y los resultados pueden ser un valor o una función. *La Programación Funcional se caracteriza por el uso de Expresiones y Funciones mientras que la Imperativa por el uso de variables, comandos y procedimientos.*
- **DECLARATIVA:** La programación declarativa está basada en la noción de Relación debido a que la relación es el concepto más general de una Aplicación. Los lenguajes declarativos están basados en el modo de pensar de los humanos en lugar del ordenador.

Principales ventajas que aportan:

- Elegancia, claridad, sencillez, potencia y concisión.
- Semánticas claras, simples y matemáticamente bien fundadas.
- Cercanos al nivel de abstracción de las especificaciones formales/informales de los problemas a resolver.
- Aplicaciones variadas y de gran interés.

- **Inteligencia artificial** trata de diseñar programas de ordenador inteligentes que simulen diferentes aspectos del comportamiento humano inteligente.

- **Ingeniería de software** Es importante tenerla en cuenta en el desarrollo de un sistema interactivo. Esta disciplina estudia técnicas de diseño y desarrollo del software.

- **Usabilidad** Para que un sistema interactivo cumpla sus objetivos tiene que ser *usable* y, además, *accesible* a la mayor parte de la población humana.

**¿Por qué es importante la usabilidad?**

Principios de diseño en ingeniería basados en la usabilidad, tiene como resultado:

- una reducción de los costes de producción.

- una reducción de los costes de mantenimiento y apoyo.

- una reducción de los costes de uso .

- una mejora en la calidad del producto .

*El problema radica en el desarrollo del producto, en el énfasis de la tecnología, en vez del usuario, la persona para la cual está hecho el dispositivo.*

Que las interfaces de usuario estén mal diseñadas es un factor que frena el uso de las funcionalidades. Por tanto, es muy importante diseñar **interfaces de usuario usables**.

Podemos entender la **usabilidad como aquella característica que *hace que el software sea fácil de utilizar y fácil de aprende*r**.

### 4.4.2 Entorno de desarrollo

El entorno de desarrollo de desarrollo en el que se va a realizar el plugin es Eclipse, se van a utilizar dos tecnologías que incorpora eclipse para el control de código y para la compilación, la ejecución y pruebas, dichas tecnologías son Apache Ant [RW93] y Subversive SVN [RW94].

Subversive SVN es una herramienta de control de versiones open source basada en un repositorio cuyo funcionamiento se asemeja enormemente al de un sistema de ficheros. Es software libre bajo una licencia de tipo Apache/BSD.

Utiliza el concepto de revisión para guardar los cambios producidos en el repositorio. Entre dos revisiones sólo guarda el conjunto de modificaciones (delta), optimizando así al máximo el uso de espacio en disco. SVN permite al usuario crear, copiar y borrar carpetas con la misma flexibilidad con la que lo haría si estuviese en su disco duro local, lo cual permite que varias personas desde distintos puntos del mundo puedan modificar y administrar el mismo código, sin temer que la calidad del software se vea afectada.
Apache Ant es una herramienta usada en programación para la realización de tareas mecánicas y repetitivas, normalmente durante la fase de compilación y construcción (build). Es, por tanto, un software para procesos de automatización de compilación, desarrollado en lenguaje Java y requiere la plataforma Java, así que es más apropiado para la construcción de proyectos Java.

Apache Ant, tiene la ventaja de no depender de las órdenes del shell de cada sistema operativo, sino que se basa en archivos de configuración XML y clases Java para la realización de las distintas tareas, siendo idónea como solución multi-plataforma.

Ant utiliza XML para describir el proceso de generación y sus dependencias. Por defecto, el archivo XML se denomina build.xml.
Ant es un proyecto de la Apache Software Foundation. Es software open source, y se lanza bajo la licencia Apache Software.

El proceso de creación del plugin requiere una preparación previa del entorno. Nos hará falta instalar complementos a Eclipse en el caso de que no se encuentren instalados, además del código fuente de JOSM y de los sistemas añadidos. Todo esto permitirá la correcta ejecución de las pruebas e integración de nuestro plugin con JOSM.

Para facilitar la comprensión del proceso, se va a dividir en dos partes bien diferenciadas, la primera parte va a consistir en instalar Eclipse con los complementos y el código fuente de JOSM, la segunda parte consiste en la instalación del entorno de desarrollo de los plugin y la configuración de las variables para permitir la ejecución del código del plugin dentro del programa JOSM.

Los pasos la instalación de eclipse y código fuente de JOSM son los siguientes:

**1- Instalación de Eclipse.**

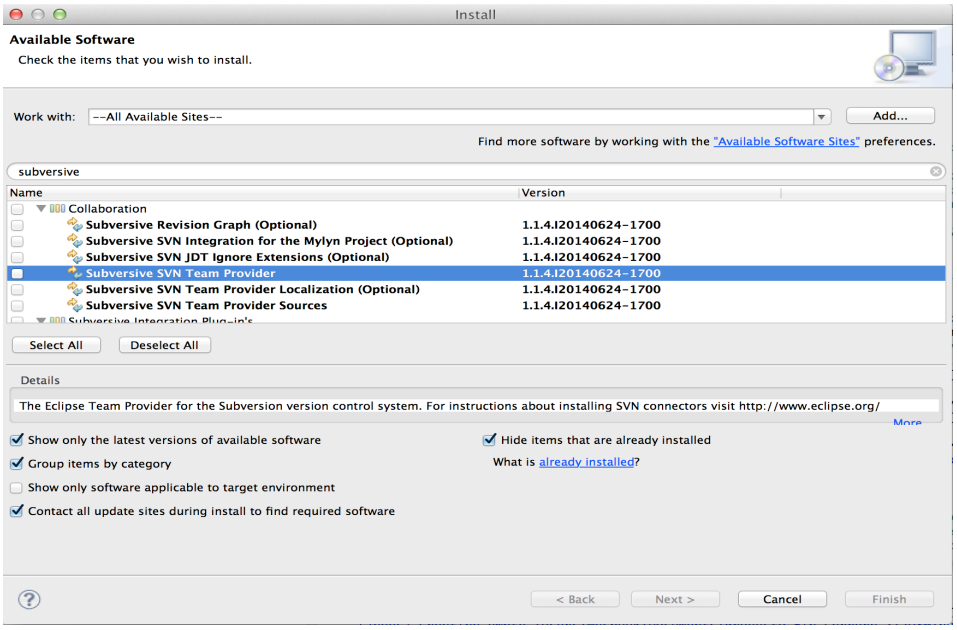
Se puede elegir cualquier distribución de eclipse instalándola de forma normal, según nuestro sistema operativo cambia la forma de instalación de eclipse.

**2- Instalación del complemento SVN.**

Se requiere la instalación del complemento SVN para descargar del repositorio el JOSM y los plugin. Para ello es necesario seguir los siguientes pasos:

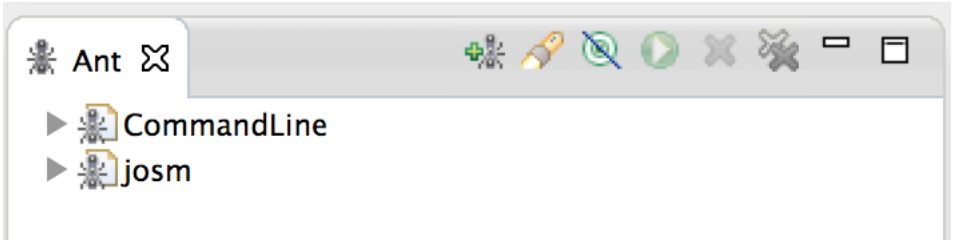
- En el menú superior ir a la pestaña de Help.

- Seleccionar la opción de Install new Software.
- Seleccionar en la pestaña de Work with la opción de All Availables sites.
- En el buscador poner SVN, y entre las opciones que aparecen instalar Subversive SVN Team Provider.
- También será necesario instalar los Subversive SVN conectors, y dado que queremos que el programa tenga la máxima compatibilidad, sería recomendable instalar todos los conectores.



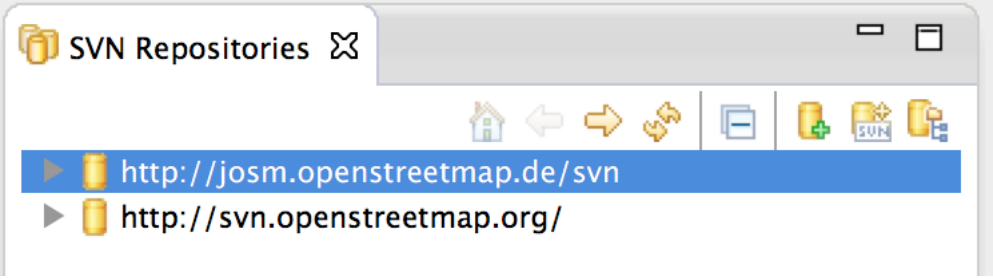

### 3- Vistas de ANT.

Una vez instalado el SVN, ir al menú superior, la pestaña de Window y seleccionar Show View, entre las opciones seleccionar Ant, esta ventana nos será útil en un futuro.



### 4-Perspectivas de SVN

Ir al menú superior a la pestaña de window y seleccionar Open Perspective, y seleccionar la opción de SVN Repository, en el caso de que no aparezca se encontrara dentro de la pestaña de other.




### 5- Agregar Repositorio.

Una vez que tengamos abierta la perspectiva SVN Repositry, buscar el botón de añadir repositorio -> , y añadir el repositorio de JOSM, el cual se encuentra publicado en la página https://JOSM.openstreetmap.de/wiki/Download
En esta página se encuentra la información de la localización del source, actualmente, el repositorio se encuentra alojado en la siguiente localización:
https://JOSM.openstreetmap.de/svn/trunk

Pegamos esta dirección y pulsamos el botón de finish, no hacen falta más parámetros de configuración.
En este punto se nos tiene que añadir el repositorio en la pestaña de repositories.

### 6- Descargar de repositorio.

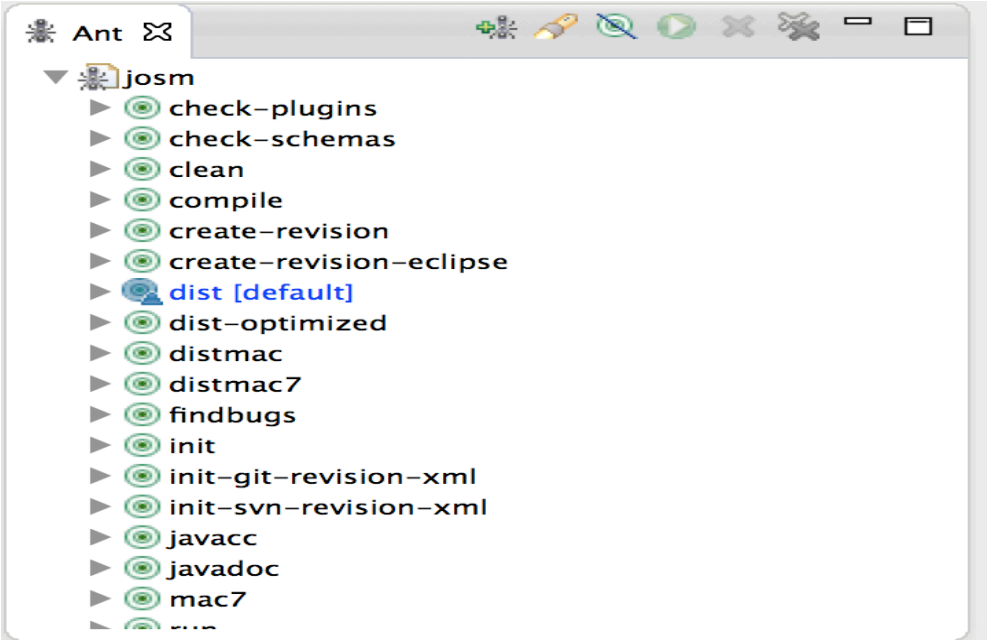
Para descargar el repositorio pulsamos en la pestaña de repostiories, y encima del repositorio JOSM con el botón derecho, y seleccionar la opción de CheckOut.
En este punto se nos tiene que descargar completamente JOSM a nuestro workspace.

### 7- Cargar build.xml de JOSM mediante Ant

Nos vamos a la perspectiva Java, la cual se encontrará en la esquina superior derecha, una vez allí tendremos la vista que hemos conseguido anteriormente Ant,

En la vista Ant, una vez allí pulsamos  el cual nos permite buscar y añadir un archivo XML de Ant.

Buscamos el dentro del proyecto JOSM, el archivo buid.xml y lo abrimos.



## 8- Compilado de JOSM.

En este momento en la vista Ant tendremos que tener diferentes opciones, tales como dist y compile.
Ejecutamos la opción de compile la cual nos tendría que devolver por consola BUILD SUCCESSFUL de forma que sabemos que JOSM lo compila perfectamente.

Luego utilizamos la opción Dist, está la deberemos de ejecutar en función de nuestro sistema operativo y nuestra versión de Java, ya que existen diferentes tipos de Dist, Distmac (Para sistemas operativos Macintosh) y Distmac7 (Para sistemas operativos Macintosh y con Java 7).

Una vez ejecutado, nos generará en la carpeta de salida la cual se especifica en la salida de la consola, nuestra propia compilación de JOSM-custom.jar, la cual nos será útil para enlazarla luego con el sistema de plugin.

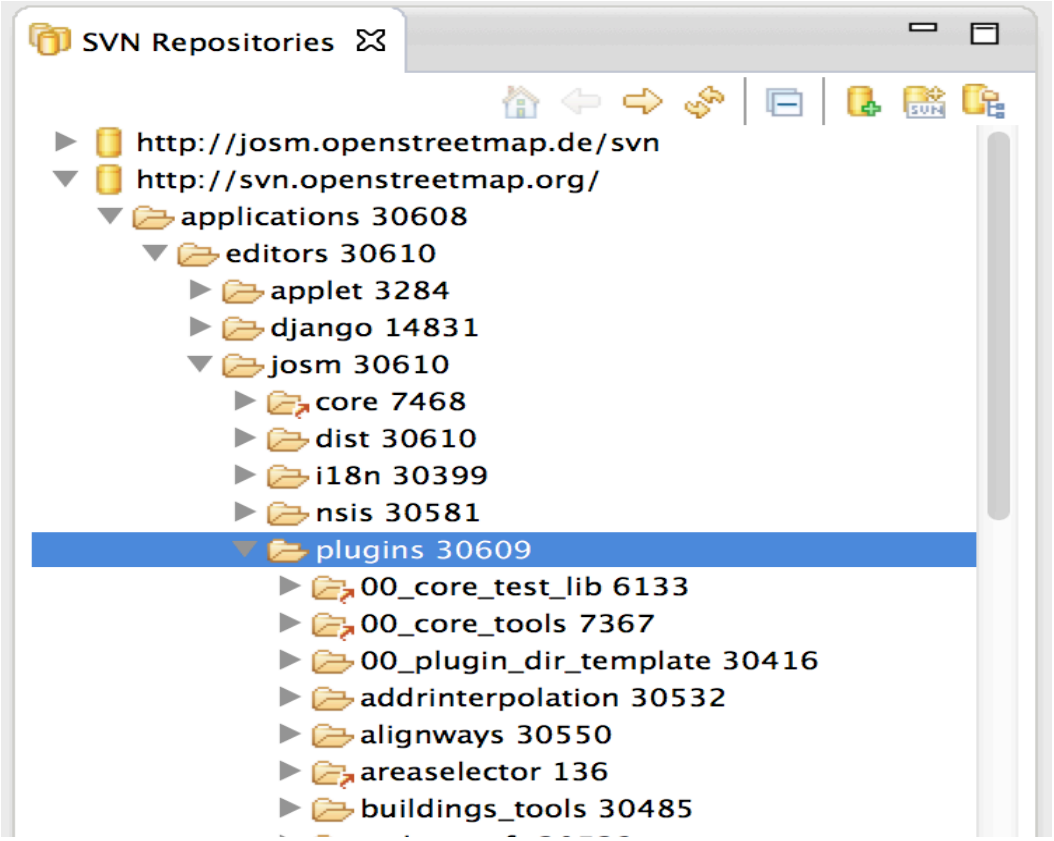
Una vez completada la primera parte y teniendo instalado JOSM y compilado, se puede pasar añadir el entorno de desarrollo de los plugin, los pasos son los siguientes:

### 1- Añadir repositorio

Seleccionamos la perspectiva de SVN Repositories, y añadimos un nuevo repositorio SVN al igual que hemos hecho anteriormente.
El repositorio que se necesita añadir es el siguiente:
http://svn.openstreetmap.org/

### 2- Realizar Check Out

Se realiza un Check Out completo de la carpeta Editors, para ello en la perspectiva SVN buscamos la carpeta Editors y pulsando con el botón derecho sobre la carpeta, se elige la opción de checkOut.




### 3- Cargado del plugin.

Seleccionamos dentro de la perspectiva de SVN y vamos navegando por el repositorio que acabamos de añadir, applications Editors -> JOSM -> plugins.

Una vez allí, en función de lo que nos interese, podemos elegir entre cargar un plugin completo de los que tiene JOSM para tomarlo como base y a partir de ahí generar el nuestro, o bien partir del plugin predefinido que te ofrece JOSM.

Realizamos un CheckOut del proyecto correspondiente.

build-common.xml

build-common.xml 〉 No Selection

```
<?xml version="1.0" encoding="utf-8"?>
<!--
** Template for the build targets common to all plugins
** ================================================
**
** To override a property, add it to the plugin build.xml _before_
** this template has been imported.
** To override a target, add it _after_ this template has been imported.
**
** Paths are relative to the build.xml that imports this template.
**
-->
<project name="plugin_common" basedir=".">

    <property name="josm"                    location="../../core/dist/josm-
        custom.jar"/>
    <property name="plugin.build.dir"        value="build"/>
    <property name="plugin.src.dir"          value="src"/>
    <property name="plugin.lib.dir"          value="lib"/>
    <!-- this is the directory where the plugin jar is copied to -->
    <property name="plugin.dist.dir"         value="../../dist"/>
    <property name="ant.build.javac.target" value="1.6"/>
    <property name="ant.build.javac.source" value="1.6"/>
    <property name="plugin.jar"              value="${plugin.dist.dir}/$
        {ant.project.name}.jar"/>

    <!--
    ***********************************************************
    ** init - initializes the build
    ***********************************************************
    -->
    <target name="init">
        <mkdir dir="${plugin.build.dir}"/>
    </target>
    <!--
    ***********************************************************
    ** compile - compiles the source tree
    ***********************************************************
    -->
    <target name="compile" depends="init">
        <echo message="compiling sources for ${plugin.jar} ..."/>
        <javac srcdir="src" debug="true" destdir="${plugin.build.dir}"
            includeantruntime="false" encoding="UTF-8">
            <compilerarg value="-Xlint:deprecation"/>
```

## 4- Configuración de las variables de entorno.

Este es uno de los puntos más complicados y críticos del sistema dado que apenas hay documentación en internet sobre esta configuración.

Vamos a realizar la unión entre el JOSM-custom.jar que realizamos anteriormente y nuestro plugin, de forma que directamente desde eclipse podamos probar nuestro plugin con JOSM.

Para ello con el explorador de carpetas del sistema, nos posicionamos en nuestro el inicio del workspace. A partir de ahí nos vamos a Editors-> JOSM -> plugins.

En esta carpeta buscamos el archivo build-common.xml y modificamos la localización del JOSM, y ponemos el que hemos generado JOSM-custom.jar, la dirección completa en mi caso sería la siguiente, locación="../../corre/dist/JOSM-custom.jar" .


## 5- Utilización de Ant

Al igual que hicimos en caso anterior, vamos a utilizar el Ant, pero esta vez usaremos opciones diferentes.

Si todos los pasos anteriores han ido bien, desde la perspectiva Java y la vista de ant podremos ejecutar opciones de Dist, para crear un .jar de nuestro plugin, y para una de las principales para comprobar

la funcionalidad es RunJOSM, que incrusta nuestro plugin dentro de JOSM.

### 4.4.3 Servidor de BaseX

BaseX Server es un servidor de alto rendimiento y de bajo mantenimiento de bases de datos XML, está especialmente diseñado para el almacenamiento y consulta de XML en la red.

BaseX Server no depende otra biblioteca y el impacto en el uso de la memoria es reducido, por lo que es una buena alternativa para sistemas integrados y soluciones web ligeros.

BaseX servidor agrega toda la funcionalidad del sistema de base de datos XML BaseX y su procesador XQuery eficiente en una arquitectura cliente / servidor.

Las principales características de BaseX Server:

- **Comunicación cliente-servidor** El BaseX Server ofrece un almacenamiento central para millones de documentos XML y archivos binarios. Se puede acceder de forma remota con clientes en varios lenguajes de programación.
- **Administración de transacciones** En un entorno multiusuario, el gestor de bloqueos ayuda a las operaciones de lectura simultáneas y transacciones de escritura ACID, de forma que se realicen de forma segura.
- **Gestión de usuarios** Permisos globales y locales se pueden asignar a los usuarios que se conectan al servidor mediante autenticación segura.
- **Registro de transacciones** Se guarda un registro cronológico de todas las operaciones realizadas en el servidor, tanto de lectura como de escritura.

- **REST Server**. BaseX proporciona una interfaz REST para acceso a datos. El servidor REST se puede utilizar para acceder a bases de datos XML locales y remotos. La salida de los resultados de la consulta en el formato JSON permite a los desarrolladores de AJAX para aprovechar el potente lenguaje XQuery y la interoperabilidad de XML.

- **Servidor WebDAV** Para el acceso ad-hoc a las bases de datos, BaseX ofrece un servicio WebDAV, que permite a los usuarios almacenar rápidamente, modificar y organizar los recursos con un simple administrador de archivos compatible con WebDAV.

### Utilización del servidor BaseX

El propio programa de BaseX permite su utilización como BaseX Server. Aprovechando las características de BaseX Server, se va a establecer una arquitectura cliente-servidor entre el programa BaseX Server y el plugin. Siendo el plugin el cliente y BaseX el servidor.

La necesidad de implementar el plugin con esta arquitectura se justifica con diferentes razones;

- La utilización de una arquitectura Cliente-Servidor permite que la mayor parte de la carga de cómputo se realice de forma remota, evitando la carga excesiva del cliente.
- Este tipo de arquitectura permite que sea independe del sistema operativo en el que se ejecute la aplicación. Siendo este uno de los grandes problemas que se han tenido que solucionar.

El problema viene motivado por las librerías del programa BaseX [RW95], las cuales cambian según el sistema operativo, por lo que la librería GeoModule [RW96] de XPath necesaria para ejecución de las consultas

únicamente estaba disponible bajo el sistema operativo Windows, dada esta problemática es necesario crear una estructura que funcione con independencia del sistema operativo.

La arquitectura cliente-servidor implementada en el sistema permite controlar el flujo de los datos de la aplicación. Se puede utilizar un servidor local o remoto de BaseX, de forma que podemos controlar el acceso a nuestras consultas.

## Guía de instalación

Para poder realizar una conexión a un servidor de BaseX y utilizarlo es necesario realizar una serie de pasos:

### 1- Instalación de BaseX

La instalación depende del sistema operativo, aunque se recomienda que se realice bajo una plataforma Windows.

### 2- Configuración del Sistema Operativo

Es necesario una pequeña configuración del sistema operativo, dicha configuración es básica para cualquier tipo de servidor.
Necesitamos comprobar que el antivirus no bloquea la conexión de BaseX a internet.

En el caso de que necesitemos conectarlo a la red es probable que necesitemos abrir los puertos del router.
También puede darse el caso de que este en nuestro propio equipo instalado BaseX, en ese caso la IP de conexión será localhost o 127.0.0.1.

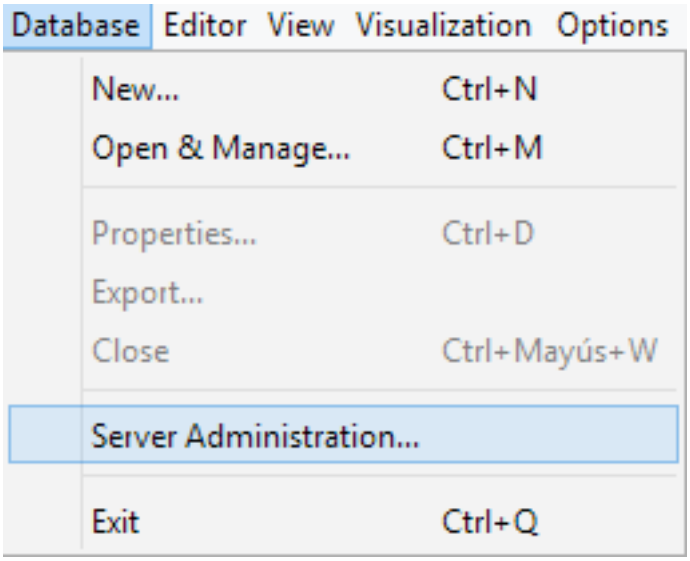
En el caso de que se realice la conexión mediante máquina virtual o en una lan, necesitamos la IP del equipo, para configurar el plugin.

### 3- Configuración de BaseX

Para la configuración de BaseX es necesario configurar BaseX

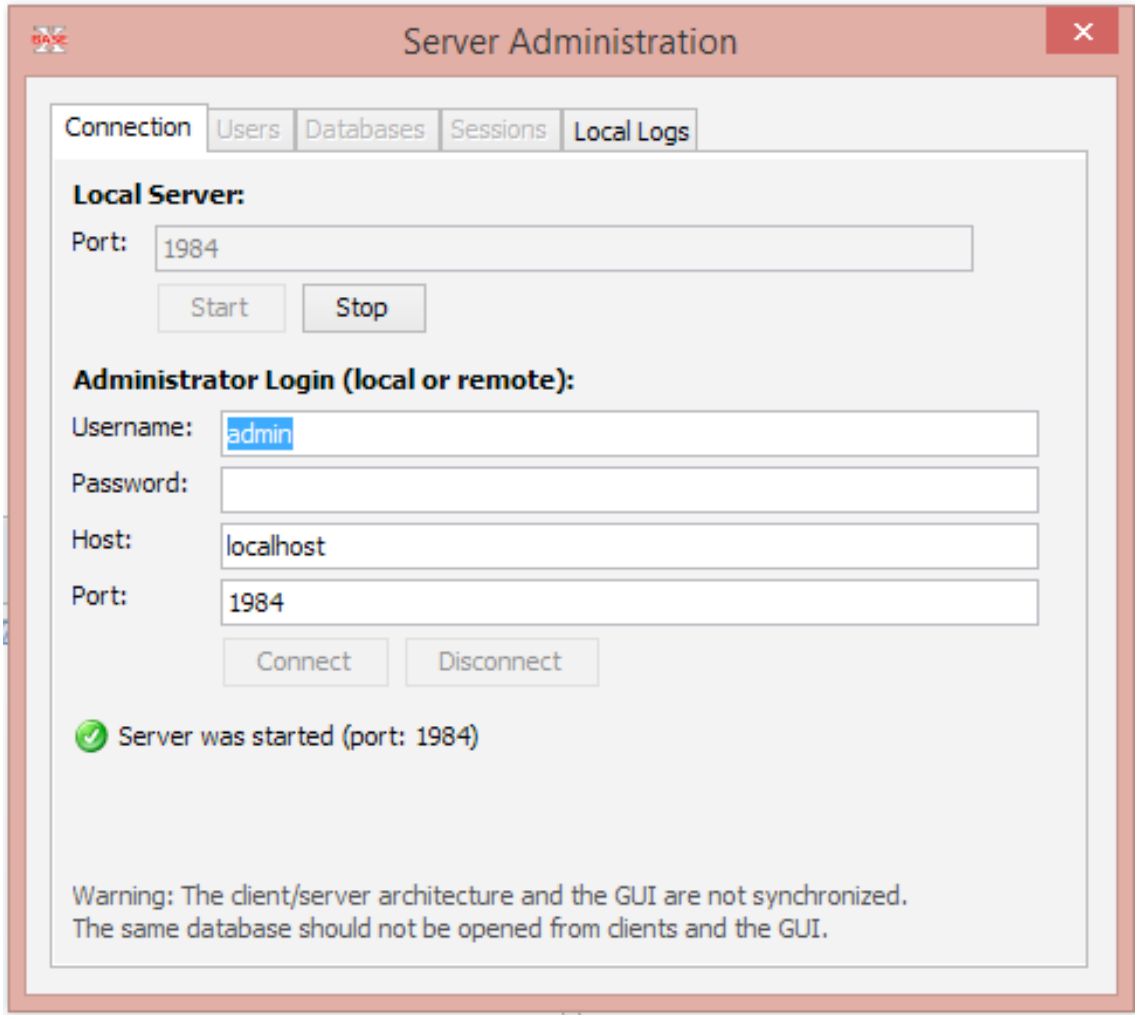
| | |
|---|---|
| Dirección IPv4: | 172.16.7.128 |
| Servidores DNS IPv4: | 172.16.7.2 |
| Fabricante: | Intel Corporation |
| Descripción: | Conexión de red Gigabit Intel(R) 82574L |

como servidor, para ello nos vamos al menú de base de datos y




Seleccionamos la opción de Server Administration.

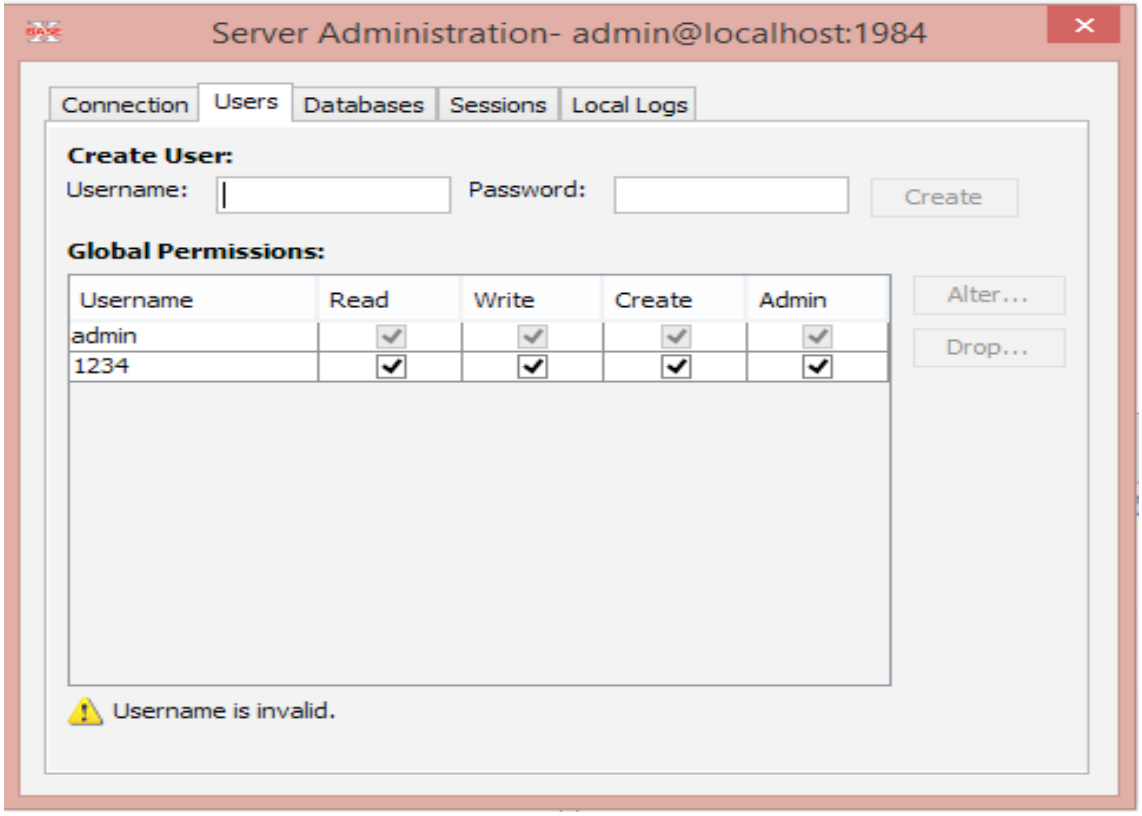
Una vez ahí para poner al servidor en marcha presionamos el botón de Start.




Con el servidor ya funcionando podemos acceder para configurar el resto del servidor como un usuario registrado, para ello seleccionamos la ip del host, y por defecto el usuario es "admin" y la contraseña igual.

## 4- Administración de servidores

Una vez logueados, podemos pasar a realizar acciones de mantenimiento, creación de usuarios, dar privilegios y gestionar bases de datos




Los comandos usados comúnmente para la gestión del sistema son:

-Ejecutar

RUN [file]

- Abrir bases de datos

OPEN [name]

- Añadir elementos a la base de datos

STORE (TO [path]) [input]

- Crear Base de datos

CREATE DB [name] ([input])

- Mostar el estado del gestor

INFO

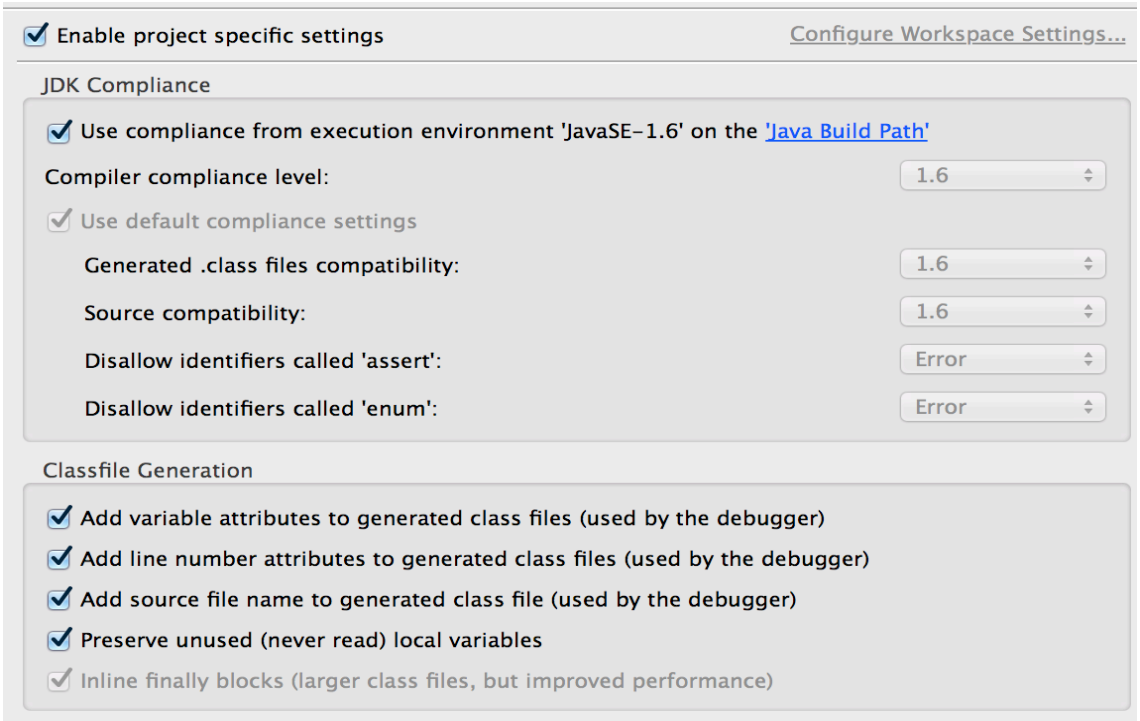
### 4.4.4 Clases Java y Codificación

#### *4.4.4.1 Primeros pasos de creación de JOSM-Urban*

Para crear un nuevo proyecto-directorio es necesario ubicarlo en la siguiente ruta JOSM / plugins / JOSM-Urban. Se puede crear a partir de una copia del directorio de plantillas 00_plugin_dir_template, también es posible partir de plugin ya desarrollado. Este directorio incluye el diseño básico, un archivo de licencia y una plantilla para el build.xml .

Para comenzar abrimos el script de ANT ( build.xml ) en el directorio de plugins y configuramos las propiedades. No vamos a entrar en profundidad en estos pasos ya que los hemos explicado detenidamente en el punto anterior "4.4.2 Entorno de desarrollo".

Hay que tener en cuenta, que JOSM se compila en archivos Java 6, por lo que si se utiliza Java7, hay que especificar "target 1.6" como parámetro en la configuración del proyecto o no se podrá utilizar con el JOSM oficial.



**Nombrar Plugin**

Cada plugin tiene un nombre único para identificarlo. El nombre es un identificador corto como URBANJOSM.

- No está permitido utilizar caracteres de espacio en blanco en el nombre.
- No utilizar caracteres especiales en el nombre del plugin, se permiten únicamente caracteres alfanuméricos. Hay que tener en cuenta que el nombre se usa para crear nombres de archivos y directorios en el equipo donde están instalados los plugins.

  Si se incluyen caracteres especiales como /,., \, Etc estas operaciones pueden fallar.

Es necesario seguir el convenio estándar de paquetes Java, por lo que usar un paquete de Java para la clase principal y todas las demás clases que necesita, es decir, requiere su propio paquete.

En el nombrado se siguen las convenciones de nombres establecidos utilizadas en el mundo Java. Seguir las convenciones permite a JOSM identificar y deshabilitar temporalmente un plugin que ha lanzado una excepción.

### *4.4.4.2 Elementos clave de la compilación.*

JOSM-Urban sigue la misma estructura de cualquier proyecto, tiene una clase principal llamada ControladorPrincipal, esta clase es de la que nacen todas las demás.

Para que la clase ControladorPrincipal pueda ser usada como raíz del plugin tiene que heredar de la clase Plugin de la API de JOSM. Al heredar de esta clase el sistema entiende que es el nodo raíz de plugin.

Al heredar es obliga a tener un constructor con un elemento PluginInformation como entrada.

Este método constructor será el usado por JOSM para crear inicializar el plugin. El archivo que se le pasara por parámetro será siempre un archivo POJO (Plain Old Java Object).

**Construir un JAR**

Para construir el URBANJOSM.jar es necesario utilizar ANT, una vez seleccionado, ejecutamos el target(objetivo), dist y nos creara en la carpeta de salida un arhivo.jar. Este punto se encuentra desarrollado en la sección 4.4.6.

## Ejecutar el proyecto

Para ejecutar nuestro plugin y probar la funcionalidad existen dos métodos.

a) Realizar el empaquetado utilizando ANT, una vez generado el archivo.jar añadirlo manualmente a la capeta de plugins de JOSM y al arrancar el JOSM el plugin estará correctamente cargado.

b) Desde el propio ANT ejecutar el target RunJOSM, el cual se encargará de realizar automáticamente el empaquetado, y añadirlo a nuestro JOSM, para poder utilizarlo es necesario tener bien puesta la ruta del .jar de JOSM. Este punto está más desarrollado en la sección 4.4.6.

## La interfaz de un complementos

La clase principal no tiene porqué ser derivada de ninguna clase común, aunque es muy recomendable hacerlo. La clase de la que es recomendable que herede es Plugin de la API de JOSM. La razón es que permite que JOSM ejecute algunos métodos a lo largo del ciclo de vida de la aplicación. Algunos de estos métodos es necesario sobrescribirlos para JOSM pueda ejecutarlos.

Los métodos más representativos que se encuentran al heredar de Plugin son:

```
public void mapFrameInitialized (MapFrame oldFrame, MapFrame newFrame)
```

JOSM puede gestionar como máximo un MapFrame . Al iniciar la aplicacion, el MapFrame es nulo. Este MapFrame solo se crea cuando se añade la primera capa, y a su vez al eliminar la última capa se restablece a un valor.

Al tener este método nos permite jugar con el cambio entre capas y recoger la capa nueva y la vieja e intercambiarlas. Es un método muy útil dado que informa del cambio entre capas.

```
public PreferenceSetting getPreferenceSetting ()
```

Este método es muy útil a la hora de gestionar las preferencias. En el caso de que nuestro plugin necesite de preferencias especiales, podremos modificar las preferencias actuales y aplicar las que sean necesarias. En caso de no implementarlo o volverlo nulo damos por hecho que no es necesario utilizar las preferencias.

```
public void addDownloadSelection (List <DownloadSelection> list)
```

El método sirve para gestionar manualmente la descarga de datos de JOSM. Sobrescribir este método nos permite tener control sobre el contenido descargado, añadiéndolo o no a la interfaz activa entre otras cosas. También permite modificar la interfaz de descarga añadiendo un elemento gráfico como un JPanel.

### Detalles de implementación

Para poder recibir notificaciones sobre los cambios producidos por una capa activa y que se gestionen automáticamente mediante eventos (P.E cada vez que se seleccione una calle que muestre un dialogo) está la lista de eventos LayerChangeListener.
Si creamos una LayerListener y la agregamos la lista de eventos, JOSM nos notificara acerca de los cambios en una capa.
Existen más peculiaridades en cuanto a la respuesta de eventos, pero este es el más significativo para nuestro caso.

### Acceso al sistema de archivos local

A los plugins se les permiten actualmente a leer y escribir en el sistema de archivos local. Es importante escribir la ruta especifica en las preferencias y para acceder a la clase donde se ubica el plugin se puede utilizar el comando `getPluginDir ()` el cual nos permite obtener el nombre del directorio del plugin para acceder a él.

### *4.4.4.3 Clases y codificación*

De todas las funciones implementas se van a mostrar las funciones mas significativas.
Dichas funciones son las siguientes:

**1-Funcion probarConsulta(String consulta)**

Esta función es la encargada nos permite realizar una prueba cuando se esta añadiendo una consulta. De forma que permite una previsualización de la capa solución en formato de texto para realizar comprobaciones acerca de la exactitud de la consulta.

```
public void probarConsulta(String consulta) {
//cargamos los parametros de conexion guardados en el modelo
           ModeloConexion mConexion=new ModeloConexion();

           if(consulta!=null){
                //guardamos la capa activa
                File file= new
File(mConsulta.getLocalizacionSol()+File.separator+"CapaActivaSo
lucion.OSM");
                org.openstreetmap.JOSM.gui.layer.Layer activa=
Main.map.mapView.getActiveLayer();
                activa.setAssociatedFile(file);
                Main.main.menu.save.doSave(activa);
                if (Main.map.mapView.getActiveLayer()==null) {
                     System.out.println("fallo guardado");

                }
                //Iniciamos la conexion con BaseX
                XQDataSource ds=null;
                ds = new BaseXXQDataSource();
                try {

     ds.setProperty(mConexion.getNombreServidor(),mConexion.getD
ireccionIP());

                     ds.setProperty("port",
mConexion.getPuertoAcceso());
                } catch (XQException e1) {
                     e1.printStackTrace();
```

```
                }
                XQConnection xqc=null;
                XQExpression xqe=null;
                try{
                      xqc =
ds.getConnection(mConexion.getUsuario(),
mConexion.getContrasena());
                }
                catch(Exception time){
                }
                //generamos la expresion que dara lugar al
comando de crear una nueva base de datos
                try {
                      xqe = xqc.createExpression();
                } catch (XQException e1) {
                      e1.printStackTrace();
                }
                try{
                      xqe.executeCommand("CREATE DB JOSMDB");
                }
                //muy importante cerrar las conexiones despues
de su uso si no dan errores
                catch(Exception e){
                      System.out.println("he cerrado en creadte
db");
                }
                //definimos las propiedades de la base de datos
                try {
                      ds.setProperty("databaseName", "JOSMDB");
                } catch (XQException e1) {
                      // TODO Auto-generated catch block
                      e1.printStackTrace();
                }
                //preparamos la conexion de envio de archivos
                XQConnection2 xqc2=null;
                //seleccinamos el archivo que acabamos de
generar
                File myTestDbDataFile = new
File(mConsulta.getLocalizacionSol()+File.separator+"CapaActivaSo
lucion.OSM");

                try{
                      xqc2 = (XQConnection2)
ds.getConnection(mConexion.getUsuario(),
mConexion.getContrasena());
                }
                catch(Exception e){
```

```
                    try {
                        xqe.close();

                        xqc.close();
                    }
                    catch (XQException e1) {
                        e1.printStackTrace();
                    }
                }

                try {
                    xqe.executeCommand("SET WRITEBACK true");
                } catch (XQException e1) {
                    e1.printStackTrace();
                }
                XQItem xqItem = null;
                try {
                    try {
                        xqItem =
xqc2.createItemFromDocument(new
FileInputStream(myTestDbDataFile), null, null);
                    } catch (XQException e) {
                        e.printStackTrace();
                    }
                } catch (FileNotFoundException e) {
                    System.out.println("fallo");
                    e.printStackTrace();
                }
                //INSERTA EL ITEM DENTRO DEL BASEX
                try {

    xqc2.insertItem(myTestDbDataFile.getName(), xqItem, null);
                } catch (XQException e1) {
                    e1.printStackTrace();
                }
// una vez cargado el archivo pasamos a realizar la consulta
pertinente
                XQPreparedExpression xqpe = null;
                try {
                    xqpe = xqc.prepareExpression(consulta);

                } catch (XQException e1) {
                    e1.printStackTrace();
                }
                //obtenemos los resultados.

                XQResultSequence rs = null;
```

```
                try {
                      rs = xqpe.executeQuery();
                } catch (XQException e) {
                      System.out.println("razones excepcion");
                      System.out.println("razones
excepcion"+e.getMessage());

      System.out.println(e.getLocalizedMessage());
                      e.printStackTrace();
                }

                String solucion="";
                try {
            //tengo el resultado de la consulta guardado en rs
                      while(rs.next())

      System.out.println(rs.getItemAsString(null));

      solucion=solucion+rs.getItemAsString(null);
                      rs.close();
                } catch (XQException e) {
                      e.printStackTrace();
                }

                try {
                      xqe.close();

                      xqpe.close();
                      xqc.close();
                      xqc2.close();
                }
                catch (XQException e) {
                      e.printStackTrace();
                }
                if(solucion.equals("")){
                      mConsulta.setSolucion(solucion);
                }
//Una vez cargado el resultado se modifica la solución y se
actualiza el TextArea
                vAnadirConsulta.actualizarSolucion();


          }
```

**2-Funcion realizarConsulta (String nombreConsulta)**

Esta función se encarga de capturar la traza activa de Josm, ejecutar una consulta, guardar la solución en un archivo ya determinado por le programa y cargar en programa otra vez dicha capa.

Dada similitud de esta clase con la anterior y la similitud en cuanto a código, no se va a mostrar de nuevo. La funcionalidad del código es básicamente la misma pero modificando ligeramente el flujo de datos, es decir, en vez de mostrarlo en un cuadro de texto como hacíamos anteriormente, se guarda y se carga en el programa la traza.

Hay otra variante de esta función realizarConsulta(String consulta, String archivo) en la cual se realizan las mismas operaciones, pero con la diferencia que entran por parámetro la consulta, por lo que no es necesario cargarla y el nombre del archivo con el que se quiere guardar la solución.

**3-Funcion cargarConsulta(String nombre)**

Esta función permite cargar una consulta desde memoria, en este caso la persistencia es un archivo de texto del que se lee la consulta.

```
public void cargarConsulta(String nombre) {
//Seleccionamos el archivo de consultas a leer
File archivo = new File (localizacionCon+File.separator+nombre);
                FileReader fr = new FileReader (archivo);
                BufferedReader br = new BufferedReader(fr);
                String linea="";
                String salida="";
          //Leemos línea a línea el documento
                while((linea=br.readLine())!=null){
                      salida=salida+linea+"\n";
                }
                fr.close();
                br.close();
          //Devolvemos la cadena leida
                return salida;
```

```
            }
            catch(IOException e){
                  return "archivo no encontrado";
            }
}
```

**4-Funcion guardarConsulta()**

Esta función permite guardar una consulta en formato txt.

```
      public boolean guardarConsulta(){
      //Utilizamos un FileWriter para escribir en un fichero
            try{
                  FileWriter fichero = null;
                  PrintWriter pw = null;

                  fichero = new
FileWriter(localizacionCon+File.separator+nombreConsulta+".txt")
;
                  pw = new PrintWriter(fichero);
            //Escribimos el contenido de la consulta
                  pw.println(consulta);
            //Cerramos los archivos.
                  fichero.close();
                  pw.close();
            }catch (Exception e){
                  return false;
            }
            return true;
      }
```

**5-Funcion de pruebaConexion().**

Esta función permite probar si los parámetros de conexión son correctos, intentando realizar una conexión con el servidor de BaseX.

```
public boolean probarConexion (String serverIP,String
nombreServer, String puerto, String usuario,String contrasena){
            XQDataSource ds = new BaseXXQDataSource();
            //define el elemento de conexión
            try {
            //definimos las propiedades
                  ds.setProperty(nombreServer,serverIP);
                  ds.setProperty("port", puerto);
            } catch (XQException e) {
```

```
                return false;
            }
            XQConnection xqc=null;
            try{
                xqc = ds.getConnection(usuario, contrasena);
                xqc.close();
            }
            //Si no nos contesta el servidor supones que la
conexión es fallida, en caso contrario lanzaría true.
            catch(Exception time){
                return false;
            }

            return true;
        }
```

### 4.4.5 Uso de librerías de BaseX, JOSM y GeoModule.

En el proyecto se han utilizado tres librerías, dichas librerías la utilización de dichas librerías responden a objetivos muy concretos.
Se ha usado la librería de BaseX XQJ API para la comunicación entre el servidor BaseX y nuestro programa Java, la librería JOSM API 0.6 para la comunicación con el programa principal JOSM y para incorporar varias de sus características, y por último, el módulo GeoModule dentro del servidor de BaseX, que es la encargada dentro del servidor de realizar el procesamiento de la consulta urbana.

#### *4.4.5.1 BaseX XQJ API*

Es una librería Java [RW97] que permite la comunicación con un servidor de BaseX, ya sea local o remoto. Esta Liberia permite realizar consultas en XQuery, utilización de módulos de BaseX, así como la realización de tareas de administración de servidores.
Las principales características [RW98] son las siguientes:

**La API de XQuery para Java está basada**:

- Es un estándar de interfaz de Java para XML DataSources que apoyan XQuery 1.0.
- La API XQJ es a bases de datos XML como el API JDBC es a bases de datos relacionales.
- Es un diseño ligero y es fácil de utilizar.

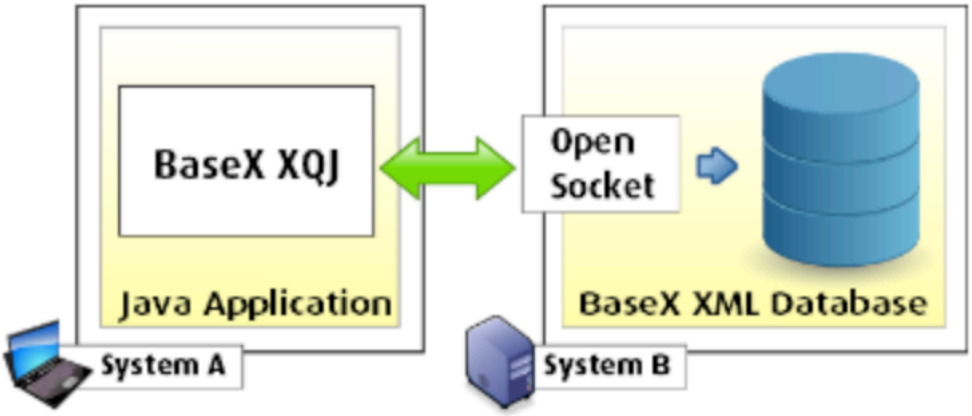
**Que permite de la interfaz API de XQuery**

- Permite la ejecución de XQuery en contra de una base de datos XML y procesar los resultados en Java.
- La compilación de variables de XQuery de Java.
- La creación de expresiones y Adhoc XQuery.
- Las transacciones ACID.
- Procesamiento de XML a través StAX, SAX y DOM.

- Permite el acceso pormenorizado a los datos.

Las secuencias de resultados proporcionan información completa XDM de datos sobre cada elemento resultado.

- ☞ Lectura y escritura de los datos en cualquier codificación, incluyendo UTF-8, UTF-16 e ISO-8859-1.




Las clases que más hemos utilizado son:

**BaseXXQDataSource** -> Permite utilizar el método setProperty para configurar la conexión, el método getConexion para probar la conexión.
**XQConnection**-> permite abrir una conexión, se iguala al getConexion del anterior para abrir una conexión con unos parámetros definidos, con createExpression nos permite crear una expresión que mediante la conexión nos permita ejecutar un comando.
**XQExpression**->nos permite ejecutar un comando con e executeCommand.
**XQConnection2**-> Abriendo una conexión como en el caso anterior, permite enviar archivos con insertItem.
**XQPreparedExpression**->permite preparar una expresión que será lanzada con la XQConnection.
**XQResultSequence**-> permite guardar la salida de la expresión realizada, de forma que se pueda mostrar.

### *4.4.5.2 GeoModule de BaseX*

GeoModule es un módulo de BaseX, que contiene funciones que se pueden aplicar a la geometría de datos que cumplen con el modelo de datos de Open Geospatial Consortium (OGC) Simple Feature (SF).

Este módulo se basa en el módulo de Geo XPath y utiliza la biblioteca Java Topology Suite .

Geometrías introducidas en GML 2 son: Point, LineString, LinearRing, Poligon, MultiPoint, MultiLineString, MultiPolygon.

Todos los nodos consultados por BaseX deben tener una geometría válida. El único tipo de geometría que no está soportada por BaseX es el geométrico múltiple.

En nuestro caso dado que los datos se encuentran en formato xml antes de ejecutar de poder realizar cualquier consulta, es necesaria una conversión a GML. Una vez realizada la conversión ya podemos utilizar este módulo.

Funciones principales son:

- **dimension**($geometry as *node()*) as xs:*integer?*
  Devuelve la dimensión de la geometría. Siempre es menor o igual que el espacio que ocupa.
- **coordinate-dimension**($geometry as *node()*) as xs:*integer*
  Devuelve el número de dimensiones del nodo, en el sistema geométrico.

- **geometry-type**($geometry as *node()*) as *xs:QName?*
  Devuelve el nombre del tipo de la geometría en tipo GML. La solución será del tipo gml:Point, gml:Curve, gml:LineString, gml:Surface, gml:Polygon, gml:MultiPoint, gml:MultiCurve, gml:MultiLineString, gml:MultiSurface, o ml:MultiPolygon.
- **srid**($geometry as *node()*) as *xs:anyURI?*
  Devuelve la URL o bien una cadena vacía en el caso de que no la encuentre.
- **envelope**($geometry as *node()*) as *element(gml:Envelope)?*
  Devuelve una solución del tipo gml:Envelope con el recubrimiento.
- **as-text**($geometry as *node()*) as *xs:string?*
  Devuelve la representación WKT de la geometría.
- **As-binary**($geometry as *node()*) as *xs:base64Binary?*
  Devuelve codificado en base-64 la geometría seleccionada.
- **is-empty**($geometry as *node()*) as *xs:boolean*
  Devuelve un booleano si es vacío o no.
- **is-simple**($geometry as *node()*) as *xs:boolean*
  Devuelve un booleano si es una figura simple o no.
- **is-3d**($geometry as *node()*) as *xs:boolean*
  Devuelve un booleano si es una figura en tres dimensiones o no.
- **is-measured**($geometry as *node()*) as *xs:boolean*
  Devuelve un booleano si la geometría tiene m valores.
- **boundary**($geometry as *node()*) as *element()**
  Devuelve el borde de la figura en formato de elementos en gml:Point o en gml:LinearRing.

#### *4.4.5.3 La API JOSM y la API OSM*

Es necesario diferenciar entre la API de OSM [RW99] que es la encargada de realizar las transacciones con los servidores OSM y la API de JOSM creada para la construcción del programa JOSM y sus complementos. A su vez la API de JOSM [RW100] necesita de la API de OSM para funcionar, dado que es la que le permite realizar extraer los mapas de los servidores, subir los cambios realizados, etc. Primero hablaremos brevemente de la API de OSM.

**La API de OSM** está basada en arquitectura RESTFul. El formato de intercambio estándar de la API es un XML compuesto por combinaciones de elementos.

La API de OSM permite conectarse con el componente del servidor al que se dirigen las solicitudes REST. Las peticiones REST recogen en forma de mensajes HTTP GET, PUT, POST y DELETE . Cualquier carga de datos se realiza en XML, utilizando el tipo MIME "text / xml" y UTF-8 codificación de caracteres.

En un principio **la API de JOSM** se desarrolló con unas ciertas peculiaridades, las cuales difieren de los estándares y buenas prácticas de Java. Los criterios con los que se desarrolló no son del todo correctos, como son poner campos públicos en clases de Java, además de poner getters y setters también públicos, entre otras cosas. Estas decisiones tenían como propósito hacer el código más limpio posible, de forma que al ver una clase se conozca exactamente su contenido.
También hay que entender esta decisión dado que no es un programa comercial, si no uno libre en el que no tiene porqué seguir unas reglas establecidas.

Otro punto de controversia es la utilización de variables globales estáticas, esto no es muy normal en una aplicación de cierto tamaño ya que crea confusión, actualmente el código que está desarrollado sigue estos patrones aunque el código desarrollado a partir de 2009 aconseja seguir el enfoque tradicional, usando los principios de encapsulación y ocultación de la información.

Actualmente estos principios se siguen a su vez en los complementos y sus derivados, de hecho, se está trabajando en una nueva versión de JOSM implementándose completamente con estos principios.
En la API existen una serie de objetos globales y estáticos accesibles desde cualquier punto de la aplicación.

Dichos objetos son los siguientes:

| | |
|---|---|
| **Main.parent** | Este es el padre de todos los elementos de la GUI, es la clase más importante dado que todas heredan de ella. |
| **Main.pref** | Es el archivo de preferencias globales, cargado desde $ {} JOSM.home / pref. Utiliza Main.pref.get (...) y Main.pref.put (...) para acceder a las preferencias.<br>Estos métodos se encargan de guardar después cambiar una opción, por tanto es aconsejable no ponen nada que no se quiere tener guardado. Para su utilización es necesario fijar el nombre del plugin para guardar sus preferencias. |
| **Main.proj** | Este permita la traducción entre diferentes idiomas. Mediante la inserción de comandos permite que se traduzcan los diferentes caracteres. |

| | |
|---|---|
| **Main.map.mapView** | Este es el principal componente de interfaz de usuario en JOSM.<br>Proporciona la vista con las capas cargadas en el sistema. Lo más normal es utilizar métodos de carga de capas, centrado, o zoom en una zona concreta. Cabe destacar que Main.map puede ser nulo cuando no hay capas cargadas. |

### 4.4.6 Targets y Empaquetados

#### *4.4.6.1 Targets*

Para crear JOSM-Urban se han utilizado una serie de targets (objetivos) que permiten diferentes funciones, lo primero, para configurar los diferentes targets es necesario modificar el archivo common-build.xml donde se encuentra la información básica, como el compilador de JOSM y a su vez es necesario generar el archivo build.xml.

La especificación de cada uno de los targets se encuentra en common-build.xml.

Para compilar el archivo build.xml es necesario utilizar un programa, en este caso utilizamos el que nos proporciona eclipse, llamado ANT.
En ANT cada en un mismo build.xml pueden existir varios targets.
Con ANT podemos ejecutar los diferentes targets, los plugins de JOSM tienen varios targets diferentes los más importantes son:

- **Clean** -> Borra la información que haya creado plugin y deja el espacio de trabajo en el estado del plugin vacío.
- **Commit-Current**-> Realiza un commit a través del svn de la versión actual del plugin y pasa a ser la versión actual en los servidores SVN OSM.
- **Compile**-> Realiza la compilación del plugin y de JOSM detectando los errores de compilación.
- **Dist**-> Genera un archivo .jar que permite integrar el plugin dentro de JOSM.
- **runJOSM**-> Empaqueta el plugin dentro de un .jar y lo guarda correctamente, después ejecuta JOSM junto con el plugin. Esto permite que se pueda testear la funcionalidad y los posibles errores el dmismo.

- **publish**-> Se sube a la plataforma SVN OSM y se publica le plugin que estará a disposición de los usuarios una vez indexado.

#### *4.4.6.2 Empaquetar un Plugin*

Una vez ejecutado el dist en el menú de Ant, se despliega como un solo archivo jar. El nombre del archivo jar tiene que ser igual al nombre del plugin, es decir JOSM-URBAN.jar.

El archivo jar tiene que incluir una serie elementos:

- Las clases Java necesarias para su ejecución, incluidas las bibliotecas adicionales que se utilicen.
- Los iconos (png o .svg), archivos de datos y así como cualquier otro tipo de recurso que sea necesario.
- Un archivo de manifiesto con entradas específicas JOSM (ver más abajo).

Al realizar el despliegue se genera un archivo "manifest" en el cual se encuentra la información del plugin, para configurar la información que se va a añadir es necesario rellenar el archivo *build.xml* con la información pertinente.

La información que se puede añadir al archivo es la siguiente:

| | | |
|---|---|---|
| **Plugin-Mainversion** | obligatorio | La versión más baja JOSM con la cual se puede ejecutar el plugin. |
| **Plugin-Version** | obligatorio | La versión actual del plugin. |
| **Plugin-Class** | obligatorio | Localización de la clase principal del plugin |
| **Plugin-Description** | obligatorio | Descripción del plugin que se muestra en la página de referencia. |

| | | |
|---|---|---|
| **Autor** | opcional | El nombre y el correo electrónico del autor del plugin. Esto se utiliza en la ventana de informe de errores, si se detecta un error en el código de plugin. |
| **Plugin-Date** | opcional | La fecha de creación del plugin en formato ISO. |
| **Plugin-Early** | opcional | Si se pone a true, el plugin se carga lo antes posible. Esto es muy útil si su plugin altera la interfaz gráfica de usuario o el proceso JOSM-startup de ninguna manera. |
| **Plugin-Link** | opcional | URL Informativo a una página web u otra fuente de información acerca de ese plugin. |
| **Plugin-Icon** | opcional | El icono que se mostrará en la lista de plugins. La imagen debe ser un archivo PNG o .svg y se incluirá en el archivo jar. |
| **Plugin-Require** | opcional | Una lista de otros plugins que se requiere que se inicien antes que el nuestro. La lista está separada por ";". |
| **Plugin-Lap** | opcional | Se pone el número en el que se debe de cargar nuestro plugin, sabiendo que cada plugin tiene un número.<br>Los números más pequeños se cargan en primer lugar, por lo que si usted tiene conflictos con otros plugins, puede aumentar o reducir este número para obtener el control sobre el orden de carga. El valor predeterminado es 50. |
| **Class-Path** | opcional | Permite añadir una nueva ruta con clases o código que permita la ejecución del plugin. |
| | opcional | Permite cargar versiones anteriores de JOSM |

| | | |
|---|---|---|
| **<Xxx> _Plugin-Url** | | en nuestro plugin. Esta información es utilizada por el controlador internamente para seleccionar la versión. Las entradas se hacen de esta manera <*JOSM_version*> *_Plugin-Url: <plugin_version>; <url>* |
| **<Lang> _Plugin-Description** | opcional | Permite añadir una descripción sobre la traducción del texto. Por ejemplo *Plugin-Description* contiene la traducción al alemán. |

## Traducción de un plugin

Un aspecto importante en JOSM es la traducción, actualmente el 99,6 % de las funciones de JOSM están traducidas a 26 idiomas o dialectos.
Dada la complejidad del proyecto no hemos hecho hincapié en este punto pero merece la pena recogerlo dada su importancia.
JOSM utiliza un sistema de traducción compatible con "gettext", pero utiliza su propio formato de archivo para almacenar los datos. Para utilizar la función de traducción que tiene hay que hacer lo siguiente:

- Únicamente hay que utilizar **tr ()** , **trn ()** , **trc ()** , **trnc ()** , **marktr ()** al igual que otras aplicaciones basadas en gettext.
- Extraer las cadenas y traducirlas:
  - a) Utilización de las herramientas normales de gettext
    - Para extraer cadenas:

      xgettext -k -ktrc: 1c, 2 -kmarktrc: 1c, 2 -ktr -kmarktr -ktrn: 1,2 -ktrnc: 1c, 2,3 ...

    - Para actualizar los archivos de traducción msgmerge.

- Utilizar las herramientas de Ant de JOSM para la traducción de cadenas.

- Para crear los archivos y almacenarlos es necesario seguir los siguientes pasos:

  - Los archivos de idioma se almacenan en directorio "data" de nuestro proyecto, suponiendo que no exista se tiene que crear, y luego nombradas con el código de idioma en minúsculas con extensión **.lang** .
  - Estos archivos son un conjunto. El archivo de base es el Inglés y los archivos de traducción deben crear junto con el base o no funcionarán correctamente.
  - Hay un scrip de perl (i18n.pl) que tiene que ser llamado con un directorio de destino y los archivos **.po** para crear los datos de la traducción.

#### *4.4.6.3 La publicación de un plugin*

La lista de plugins disponible en la página web para los usuarios se genera automáticamente a intervalos regulares (en la actualidad alrededor de 10 minutos). Esto recoge los archivos JAR ubicados en JOSM / dist en el repositorio SVN OSM.org. La web también recoge plugins externos únicamente guardando su enlace a la página web de origen. Cada forma de publicarlo tiene ventajas y desventajas.
Ventajas de alojamiento de código externo:

- Se pueden utilizar diferentes servidores para el control de versiones, VCS distinta de SVN, Git o Mercurial
- Permite aprovecharse de las diferentes ventajas de los servicios como GitHub.

Desventajas de alojamiento de código externo:

- Los plugins son menos propensos a ser traducido debido a la dificultad de encontrarlos en la red.
- Los cambios en JOSM pueden provocar que el plugin no funcione correctamente al no informar directamente al autor.
- No serán puestos en los accesos directos de JOSM de forma que es más difícil que las personas los encuentren y usen.
- Las páginas de ayuda del plugin no se mostraran en las páginas de JOSM.

**Gestión de código internamente a través del SVN de JOSM**

Hay un repositorio SVN para JOSM plugins internos en el repositorio principal SVN OSM.org. Hay que tener en cuenta que este repositorio es diferente del principal repositorio SVN JOSM que sólo gestiona el núcleo JOSM.

Se puede conseguir acceso de escritura al repositorio de plugins JOSM escribiendo a los administradores del repositorio. Para que pueda estar disponible para otros usuarios e integrarlo en el proceso de actualización plugin de JOSM es necesario enviarlo al repositorio de plugins JOSM.

La gestión de un plugin en OSM SVN en vez de sistemas propios (como hemos hablado anteriormente) tiene algunas ventajas:

- Otros usuarios pueden corregir errores y mejorar el código
- Las traducciones se realizan en conjunto con los desarrolladores de JOSM y así llegar a un mayor número de traductores.
- Los enlaces para compatibilidad del núcleo JOSM se insertan automáticamente.

#### *4.4.6.4 Licencia*

JOSM está creado bajo GPL [RW101] por lo que cualquier código que se cree a partir de JOSM, tiene que estar bajo GPL también. Cualquier plugin de JOSM es una obra derivada, por lo que solamente se pueden realizar bajo licencia. Si necesitáramos incluir código no GPL, se tiene que realizar de forma separada a la de las clases que forma JOSM.
Para ello se puede utilizar el valor de Manifest que hemos visto antes "Class-Path " que nos permite incluir otras clases o archivos.

### 4.4.7 Pruebas y documentación

En este punto se explica como se han realizado las pruebas y a que clases se han realizado. Es necesario dividir entre los distintos tipos de clases según su arquitectura, es decir, es necesario dividir entre el modelo, la vista y el controlador.

Las vistas se van a probar únicamente mediante las pruebas de funcionalidad, ya que únicamente contienen los elementos gráficos.

Las clases modelo serán testeados con pruebas de caja negra y únicamente los métodos de carga, modificación y eliminación de elementos, ya que no tiene métodos de procesos y solamente se encarga de guardar la información.

Las clases controlador serán testeados con pruebas de caja negra comprobando el resultado final del mismo.

Acotando el alcance de las pruebas se propone una batería de pruebas que se ajuste al proyecto.

#### *4.4.7.1 Pruebas de cobertura*

##### *4.4.7.1.1 Qué son*

La ***cobertura de código*** [RW102] es una medida (porcentual) en las pruebas de software que mide el grado en que el código fuente de un programa ha sido testeado. Este método sirve para determinar la calidad del test que se lleva a cabo. Además, puede determinar las partes críticas del código, o partes que no han sido testeadas. Este método se puede utilizar como técnica de optimización.

##### *4.4.7.1.2 Pruebas realizadas*

Se especifica la clase que se va a probar y a continuación los métodos que se van a probar en los test.

***Batería de pruebas:***

*Controladores:*

*-ControladorAnadirConsultas* -> probarConsulta(String consulta).

*-ControladorConexion*->probarConexion(…)

*-ControladorConsultas*-> lanzarOpcionesConsulta(), cargarTraza(), realizarConsultaRapida(…), realizarConsulta(…)

*-ControladorInfo*-> lanzarFrame()

*-ControladorOpcionesConsulta*-> borrarConsultas()

*-ControladorPreview*->lanzarAnadirConsulta(), actualizarPreview()

*-ControladorTutorial*-> lanzarOpcionesServidor(),

*Modelos:*

*-ModeloConexion*-> guardarConexion(), cargarConexion(), borrarConexion()

*-ModeloConsultas*-> cargarTabla(), cargarConsultas(), listarConsultas(), guardarSolucionRapida(), guardarConsultas(), guardarSolucion(), modificarConsulta(), borrar consulta, borrarConsultaTabla()

*-ModeloPreview*-> cargarConsulta()

### *4.4.7.2 Pruebas de funcionalidad*

#### *4.4.7.2.1 Qué son*

Las pruebas funcionales [RW103] tienen como objetivo comprobar que los sistemas desarrollados funcionan cumpliendo las especificaciones y requisitos del cliente, esto ayuda a detectar los posibles defectos generados en la fase de programación. Las pruebas incluyen la navegación, entrada de datos, procesamiento y obtención de resultados.

En nuestro caso nos especializaremos en probar la interfaz gráfica y su correcto funcionamiento.

#### *4.4.7.2.2 Rruebas realizadas*

***Batería de pruebas:***

*Vistas:*

-*VistaTutorial* -> Se comprueba gráficamente el uso de la interfaz y que los botones lancen las operaciones descritas.

-*VistaSur* -> Se comprueba gráficamente el uso de la interfaz comprobando que cambia el estado de la previsualización y que el botón lanza el añadir consultas.

-*VistaPrincipal* -> Se comprueba que se lanza el programa correctamente, no tiene prueba grafica.

-*VistaOpcionesConsulta* -> Se comprueba gráficamente el uso de la interfaz, además de la carga correcta de la consulta y del uso de los botones eliminar y modificar.

-*VistaNorte* -> Solamente se comprueba que se muestre la información y que el botón de información lance la pantalla siguiente.

-*VistaConexion* -> Se comprueba gráficamente el uso de la interfaz además del correcto funcionamiento del guardado y la prueba de la conexión.

-*VistaCentro* -> Se comprueba gráficamente el uso de la interfaz, además de los dos botones de opciones consulta y ejecutar. También se comprueba que la tabla permita seleccionar y ejecutar doble click para que actualice la ventana de previsualizacion.

-*VistaAnadirConsultas* -> Se comprueba gráficamente el uso de la interfaz, además de probar que permite probar correctamente la consulta y el guardado de la consulta y la solución.

#### *4.4.7.3 Documentación*

A lo largo de la codificación del código, se ha realizado una documentación del mismo. Para ello se ha utilizado JavaDoc [RW104]. Javadoc permite la generación de documentación sobre el código en

formato HTML a partir del código fuente Java. Javadoc es el estándar de la industria para documentar clases de Java y la mayoría de los IDEs son capaces de generarlo automáticamente.

# 5. Conclusiones y trabajo futuro

Cabe destacar que el proyecto ha cumplido los objetivos que se marcaron en un primer momento. El principal objetivo consistente en desarrollar desde cero una aplicación funcional que permitiera las consultas en XQuery para JOSM permitiendo utilizar las librerías de XQuery propuestas por J. Almendros - A. Becerra.

La gran cantidad de datos que soporta la base de datos de OSM está llegando a un nivel que hace que sea difícil procesarlos sin una infraestructura de servidores costosos, por lo que cada vez se hace más necesario herramientas que permitan el procesado de datos y la realización de consultas.

El impacto en este campo espero que sea alto dada la versatilidad de XQuery como motor de consultas. El proyecto tiene un ámbito internacional dado el amplio desarrollo a nivel mundial de JOSM y OSM.

Actualmente existen pocas herramientas dedicadas a la consulta de datos en OSM, y espero que la aplicación tenga un impacto positivo en la comunidad, llegando quizás en un futuro a ser mantenida y a desarrollar nuevas funcionalidades.

La aplicación ha sido desarrollada, pero no sin tener que resolver diferentes problemas que se iban presentando conforme el proyecto avanzaba, tales como la integración de la aplicación con el programa JOSM, la comunicación entre JOSM, JOSM-Urban y BaseX, así como problemas de diseño y restricciones al realizar la compilación junto con JOSM.

A pesar de que la aplicación es operativa, sigue teniendo mucho margen de mejora, como la propuesta de desarrollo que propone el aumento de la estabilidad mejorando el encapsulamiento previsto en el modelo MVC, siendo integrada desde diferentes plataformas, especialmente la plataforma móvil añadiendo una batería completa de consultas que sean fácilmente interpretables por el usuario.

# 6. Bibliografía

## 6.1 Referencias Bibliográficas [RB]

**[RB01]** Eliseo Clementini and Paolino Di Felice: A comparison of methods for representing

**[RB02]** Max J. Egenhofer and Robert Franzosa. Point-set topological spatial relations. International Journal of Geographical Information

**[RB03]** E. Clementini and P. Di Felice. An algebraic model for spatial objects with indeterminate boundaries

**[RB04]** A. Corral Transparencias de clase - Desarrollo rápido de aplicaciones

## 6.2 Miscelánea Web - Referencia Web [RW]

**[RW01]** - **OpenStreetMap** España | Creando el mapa libre del mundo www.**openstreetmap**.es/

**[RW02]** – **JOSM** – https://JOSM.openstreetmap.de/

**[RW03]** – **Antonio Becerra** – Universidad de Almería - http://www.ual.es/~abecerra/

**[RW04]** - What is XQuery? - XML.com http://www.xml.com/pub/a/2002/10/16/XQuery.html

**[RW05]** - BaseX | The XML Database - http://BaseX.org/

**[RW06]** - Modelo Vista Controlador http://www.comusoft.com/modelo-vista-controlador-definicion-y-caracteristicas

**[RW07]** - Mapa de google maps - https://**maps.google**.es

**[RW08]** - XML Path Language (XPath) - http://www.w3.org/TR/xpath20/

**[RW09]** - Overpass API - OpenStreetMap Wiki - http://wiki.openstreetmap.org/wiki/Overpass_API

**[RW10]** - World Wide Web Consortium (W3C) – http://www.w3c.es/

**[RW11] - XML**.com – http://www.xml.com/

**[RW12]** - Geo Module - EXPath – http://expath.org/spec/geo

**[RW13]** - The SAXON XSLT and XQuery Processor - http://www.saxonica.com/documentation/using-XQuery/

**[RW14]** - JTS Topology Suite - http://tsusiatsoftware.net/jts/main.html

**[RW15]** - Artículos académicos para Graph Modelling Language XQuery http://scholar.google.es/scholar?q=Graph+Modelling+Language+XQuery&hl=es&as_sdt=0&as_vis=1&oi=scholart

**[RW16]** - The SAXON XSLT and XQuery Processor – http://saxon.sourceforge.net/

**[RW17]** - Oracle XML DB – http://www.oracle.com/technetwork/database-features/xmldb/overview/oracle-xmldb-11gr2-1974916.html

**[RW18]** - Extensiones espaciales de MySQL - http://dev.mysql.com/doc/refman/5.0/es/spatial-extensions.html

**[RW19]** - PostGIS Spatial and Geographic Objects for PostgreSQL – http://postgis.net/

**[RW20]** - OpenOffice.org – https://www.openoffice.org/es/

**[RW21]** - Extensible Markup Language (XML) - World Wide Web Consortium - http://www.w3.org/XML/

**[RW22]** - XSLT Tutorial – http://www.w3schools.com/xsl/

**[RW23]** - XSL-FO Tutorial – http://www.w3schools.com/xslfo/

**[RW24]** - Using Qualified Names - QName – http://www.w3.org/TR/REC-xml-names/#NT-QName

**[RW25]** - XML Namespaces – http://www.w3schools.com/xml/xml_namespaces.asp

**[RW26]** - Identificador de recursos uniforme URI –
http://es.wikipedia.org/wiki/Identificador_de_recursos_uniforme
**[RW27]** - Definición de tipo de documento DTD -
http://es.wikipedia.org/wiki/Definici%C3%B3n_de_tipo_de_documento
**[RW28]** - xml:lang en esquemas de documentos XML
http://www.w3.org/International/questions/qa-when-xmllang.es.php
**[RW29]** - xml:space en XML
http://msdn.microsoft.com/es-es/library/ms788720(v=vs.110).aspx
**[RW30]** - Normas y recomendaciones XML -
http://www.mclibre.org/consultar/xml/lecciones/xml_normas.html#L1692
**[RW31]** – Requisitos de reusabilidad
http://es.slideshare.net/ignacio31/mvc-23997377
**[RW32]** – Modelo Vista Controlador Java.
http://www.academia.edu/5217432/El_patron_de_diseno_Modelo-Vista-Controlador_MVC_y_su_implementacion_en_Java_Swing
**[RW33]** – Arquitectura cliente-servidor
http://catarina.udlap.mx/u_dl_a/tales/documentos/lis/marquez_a_bm/capitulo5.pdf
**[RW34]** – Características cliente-servidor
http://www.ecured.cu/index.php/Arquitectura_Cliente_Servidor
**[RW35]** - XPath Location Path
http://www.sidar.org/recur/desdi/traduc/es/xml/xpath.html#function-last
**[RW36]** - Ejes en XPath
http://www.programacionweb.net/articulos/articulo/ejes-en-xpath/
**[RW37]** – Estándar IEEE 754
http://es.slideshare.net/alticoru/cmo-se-escribe-un-nmero-en-el-estndar-ieee-754
**[RW38]** – Arquitectura cliente-servidor, ventajas e inconvenientes.
http://www.monografias.com/trabajos24/arquitectura-cliente-servidor/arquitectura-cliente-servidor.shtml
**[RW39]** – Asocicación interaction persona ordenador.
http://aipo.es/
**[RW40]** – ExprWhitespace
http://www.sidar.org/recur/desdi/traduc/es/xml/xpath.html#NT-ExprWhitespace
**[RW41]** –MultiplyOperato
http://www.sidar.org/recur/desdi/traduc/es/xml/xpath.html#NT-MultiplyOperator
**[RW42]** - NCName
http://www.w3.org/TR/REC-xml-names/#NT-NCName

**[RW43]** - OperatorName
http://www.sidar.org/recur/desdi/traduc/es/xml/xpath.html#NT-OperatorName
**[RW44]** – ExprWhitespace
http://www.sidar.org/recur/desdi/traduc/es/xml/xpath.html#NT-ExprWhitespace
**[RW45]** – NodeType
http://www.sidar.org/recur/desdi/traduc/es/xml/xpath.html#NT-NodeType
**[RW46]** - FunctionName
http://www.sidar.org/recur/desdi/traduc/es/xml/xpath.html#NT-FunctionName
**[RW47]** – AxisName
http://www.sidar.org/recur/desdi/traduc/es/xml/xpath.html#NT-AxisName
**[RW48]** – XQuery –
http://msdn.microsoft.com/es-es/library/ms189075.aspx
**[RW49]** – SQL –
http://es.wikipedia.org/wiki/SQL
**[RW50]** - Consultas en SIG´s.
http://catarina.udlap.mx/u_dl_a/tales/documentos/msp/aragon_p_sm/capitulo1.pdf
**[RW51]** Modelo de base de datos -
http://es.wikipedia.org/wiki/Modelo_de_base_de_datos
**[RW52]** – Teoría de la decisión e incertidumbre
http://digital.csic.es/bitstream/10261/7734/1/eserv.pdf
**[RW53]** – WGS84 –
http://es.wikipedia.org/wiki/WGS84
**[RW54]** - Modelo para la implementación de la figura del operador urbano -
http://revistapostgrado.eia.edu.co/Revista%20Edici%F3n%20N%BA.2/Soluciones%202%20art%2012.pdf
**[RW55]** – Licencia BSD
https://www.freebsd.org/doc/es/articles/explaining-bsd/article.html
**[RW56]** – Java Runtime Environment
http://www.oracle.com/technetwork/Java/Javase/downloads/Java-se-jre-7-download-432155.html
**[RW57]** – HTML –
http://es.wikipedia.org/wiki/HTML
**[RW58]** – Applets en Java
http://aprenderinternet.about.com/od/Glosario/g/Applet-En-Java.htm
**[RW59]** – World Duty Free Group. Normas y procedimientos
http://www.worlddutyfreegroup.com/es/gobierno-corporativo/governance-reports/

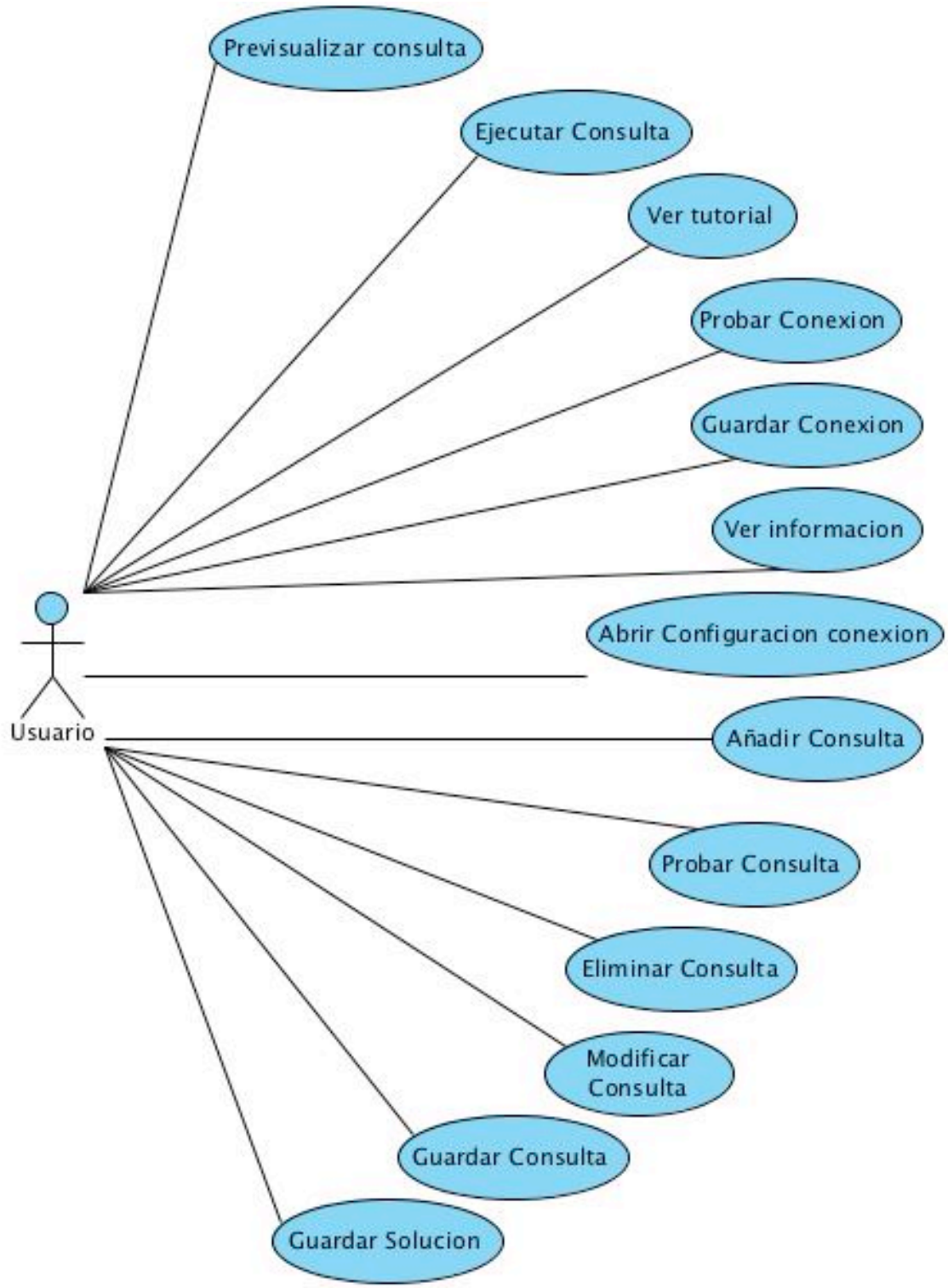
**[RW60]** - El lenguaje de programación Java
http://www.uv.es/~sto/cursos/seguridad.Java/html/sJava-30.html
**[RW61]** – JAVAWORD - Introduction to the AWT
http://www.Javaworld.com/article/2077188/core-Java/introduction-to-the-awt.html
**[RW62]** – Programación en C++
http://es.wikibooks.org/wiki/Programaci%C3%B3n_en_C%2B%2B
**[RW63] -** The UNIX® System –
http://www.unix.org/
**[RW64]** – Protocolo TCP/IP
http://protocolotcpip.galeon.com/
**[RW65]** - XmlDeclaration.Standalone (Propiedad)
http://msdn.microsoft.com/es-es/library/system.xml.xmldeclaration.standalone(v=vs.110).aspx
**[RW66]** – Browser –
http://Java.com/es/download/help/sysreq.xml
**[RW67] -** Bytecode basics **–**
http://www.Javaworld.com/article/2077233/core-Java/bytecode-basics.html
**[RW68] -** La máquina virtual Java **-**
http://www.aprenderaprogramar.com/index.php?option=com_content&id=392:la-maquina-virtual-Java-jvm-o-Java-virtual-machine-compilador-e-interprete-bytecode-cu00611b&Itemid=188
**[RW69]** – API Java –
http://es.wikipedia.org/wiki/API_Java
**[RW70]** - Parsear y extraer la información de un XML -
http://www.desarrolloweb.com/articulos/extraer-informacion-xml-php-domdocument.html
**[RW71] -** Descripción del compilador Just-In-Time
http://support.microsoft.com/kb/154580/es
**[RW72]** – MakeFiles –
http://es.wikipedia.org/wiki/Make
**[RW73]** – Java SE –
http://es.wikipedia.org/wiki/Java_SE
**[RW74]** – Java JEE –
http://es.wikipedia.org/wiki/Java_JEE
**[RW75]** – Creative Commons –
http://creativecommons.org/licenses/by-sa/2.0/deed.es

**[RW76]** - OpenStreetMap –
http://www.openstreetmap.org/
**[RW77]** – Planet.OSM –
http://wiki.openstreetmap.org/wiki/Planet.OSM
**[RW78]** – TomTom –
http://www.tomtom.com/es_es/
**[RW79]** – Waypoints –
http://es.wikipedia.org/wiki/Waypoints
**[RW80]** – Mapping parties –
http://wiki.openstreetmap.org/wiki/ES:Mapping_parties
**[RW81]** – Misión topográfica Radar Shuttle -
http://es.wikipedia.org/wiki/Misi%C3%B3n_topogr%C3%A1fica_Radar_Shuttle
**[RW82]** – WGS84 -
http://www.senderosbtt.com/index.php?option=com_content&view=article&id=145:open-street-map&catid=35:
**[RW83]** – GpsDrive –
http://www.gpsdrive.de/man-es.shtml
**[RW84]** – JOSM –
https://JOSM.openstreetmap.de/
**[RW85]** Algoritmos de busqueda -
http://www.dma.fi.upm.es/gregorio/grafos/IAGraph/busqueda.html
**[RW86]** – Vivid Solutions –
http://www.vividsolutions.com/
**[RW87]** – Licencia LGPL –
http://proint.info/wiki/Licencia_Gnu_LGPL
**[RW88]** - Simple Features Specification -
http://www.opengeospatial.org/standards/sfo
**[RW89]** - The GIS Spatial Data Model
https://courses.washington.edu/gis250/lessons/introduction_gis/spatial_data_model.html
**[RW90]** – Java getX –
http://stackoverflow.com/questions/10221327/Java-point-difference-between-getx-and-point-x
**[RW91]** - Java setX –
http://stackoverflow.com/questions/5765351/having-problem-creating-setxx

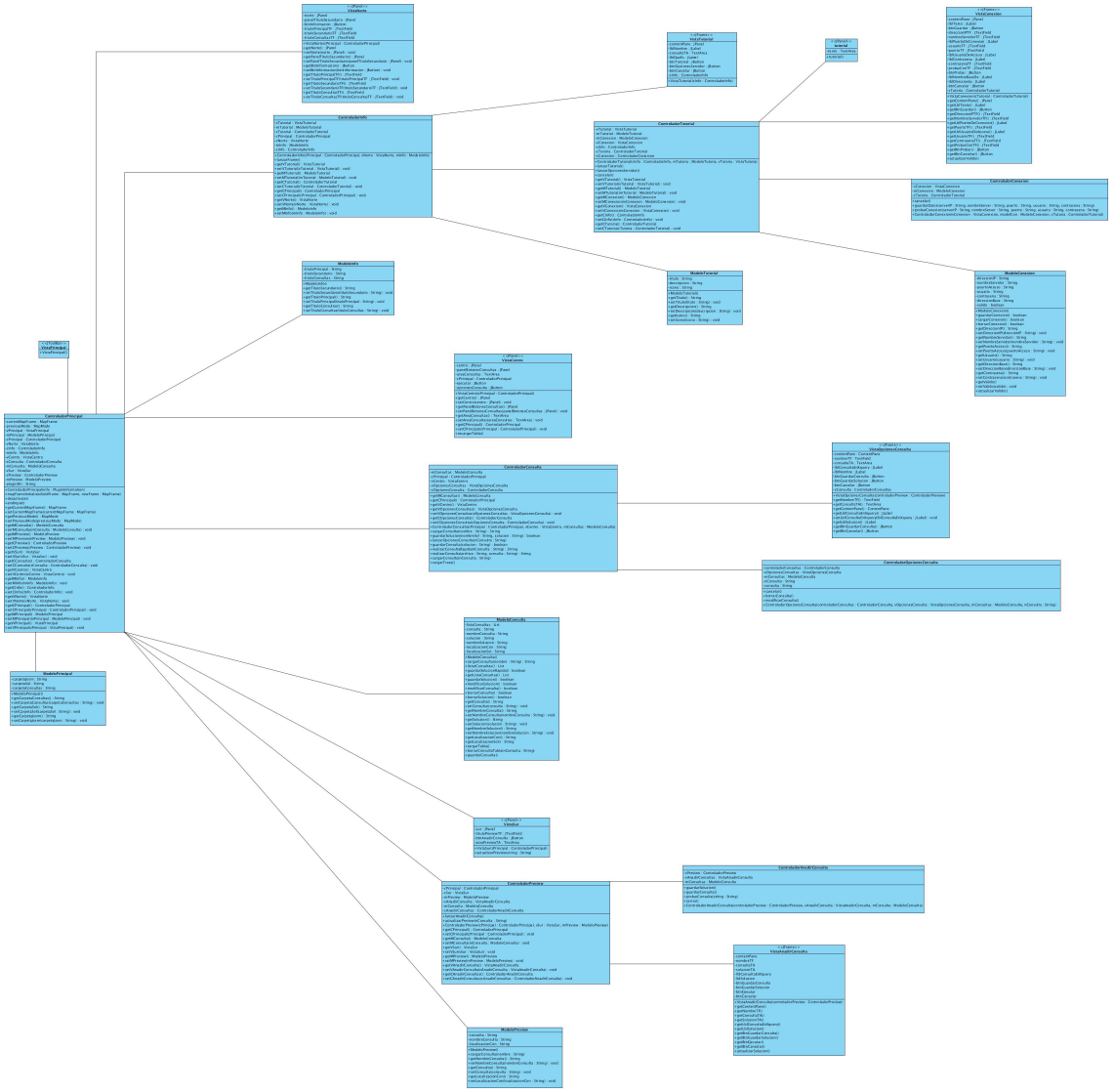
**[RW92]** – La convención ISX
http://azul.bnct.ipn.mx/tesis/repositorio/2341_2007_CIC_MAESTRIA_montesdeoca_morales_victor.pdf
**[RW93]** - Ant
http://es.wikibooks.org/wiki/Manual_b%C3%A1sico_de_ANT
**[RW94]-** Subversion SVN
https://subversion.apache.org/
**[RW95]**- Modulos BaseX
http://docs.basex.org/wiki/Module_Library
**[RW96]**- GeoModule
http://expath.org/spec/geo
**[RW97]-** XQJ Library
http://xqj.net/basex/
**[RW98]-**Caracteristicas XQJ Library
http://prezi.com/hpxhsjrcr5gg/basex-and-xqj/
**[RW99]**- Api de OSM
http://wiki.openstreetmap.org/wiki/API_v0.6#
**[RW100]-** Api de JOSM
https://josm.openstreetmap.de/wiki/DevelopersGuide
**[RW101]-** Licencia GPL
http://www.gnu.org/licenses/old-licenses/gpl-2.0-faq.es.html
**[RW102]**- Pruebas de cobertura
http://msdn.microsoft.com/es-es/library/dd537628.aspx
**[RW103]**- Pruebas de funcionalidad
http://es.wikipedia.org/wiki/Pruebas_funcionales
**[RW104]**- JavaDoc
http://www.oracle.com/technetwork/java/javase/documentation/javadoc-137458.html

# Anexo I – Modelado y Diseño II

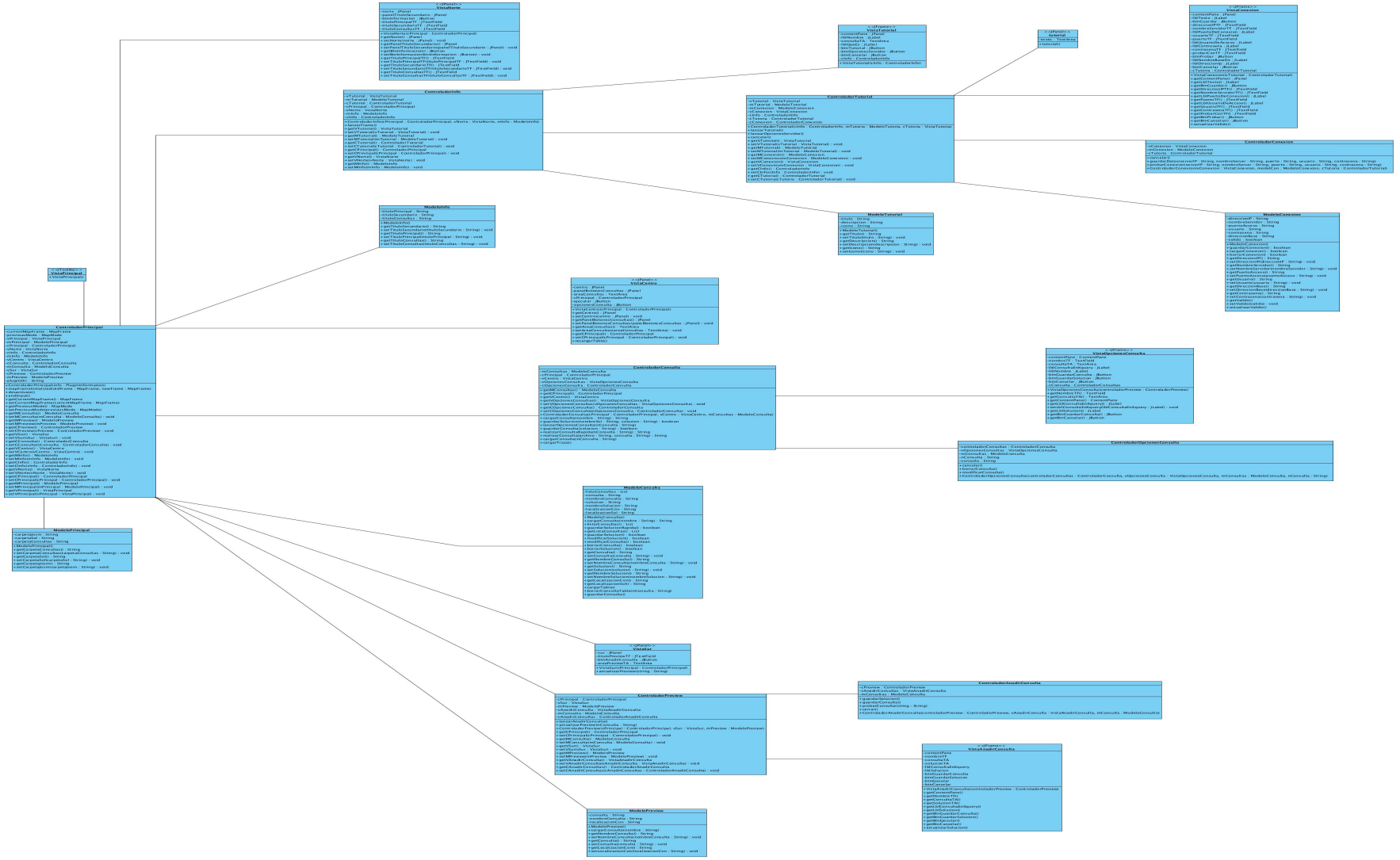
## Diagrama de Casos de uso



# Diagrama de Clases



## *Diagrama de clases*



## Diagrama de Secuencias

Diagrama de secuencias previsualizar consulta.

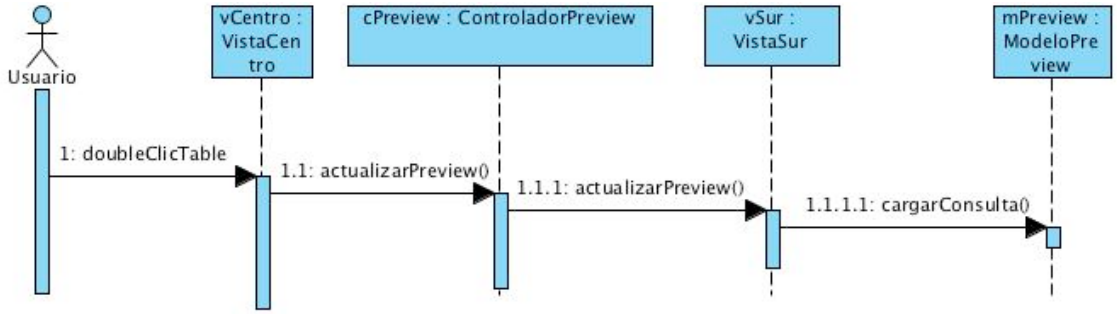



Diagrama de secuencias ejecutar consulta

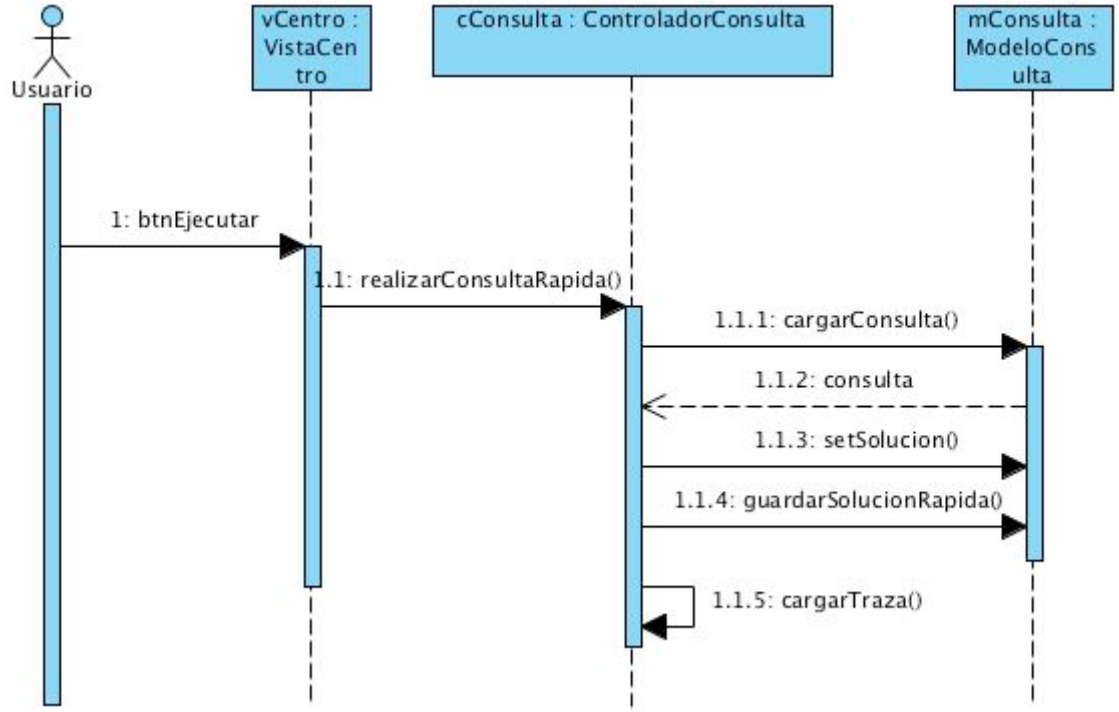



Diagrama de secuencias ver tutorial

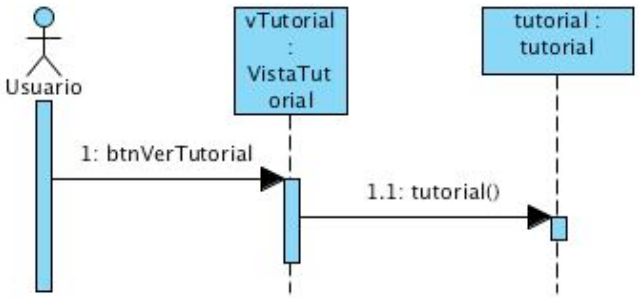


Diagrama de secuencias probar conexión

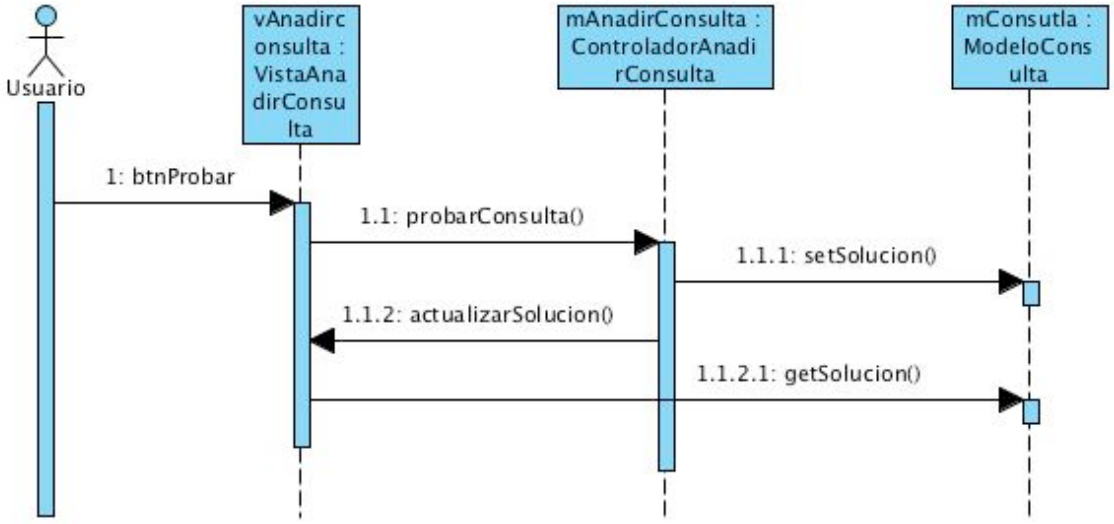



Diagrama de secuencias guardar conexión

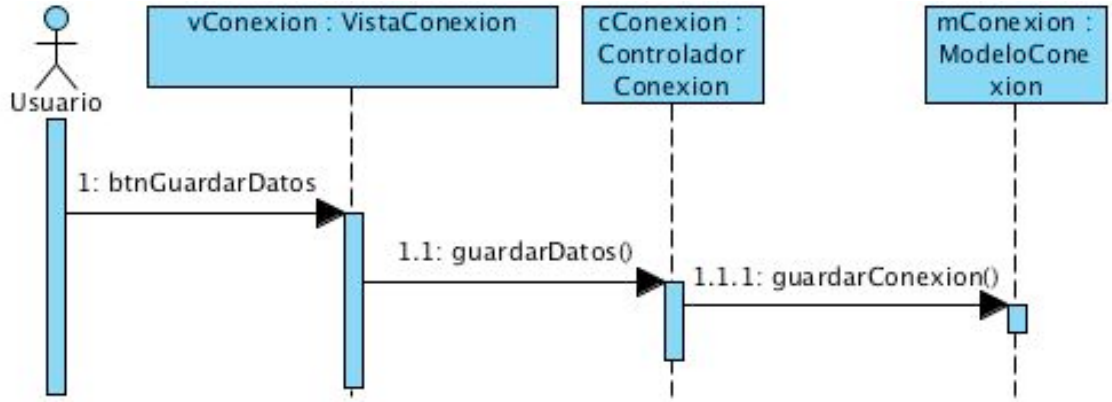



Diagrama de secuencias ver información

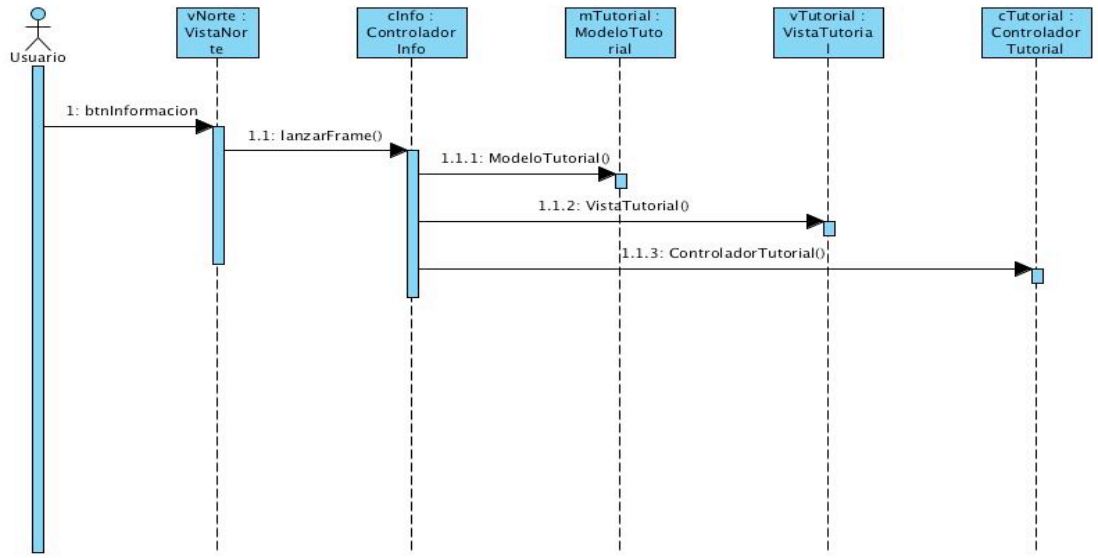


Diagrama de secuencias abrir configuración conexión

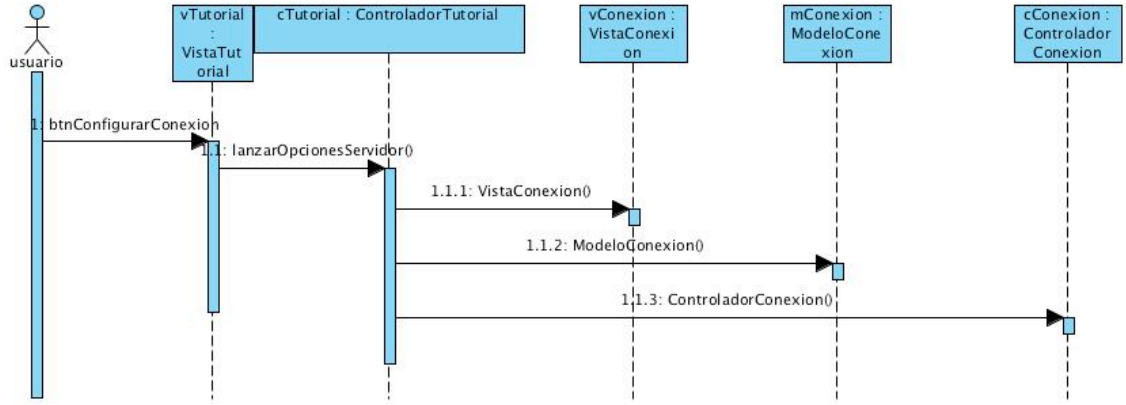



Diagrama de secuencias añadir consulta

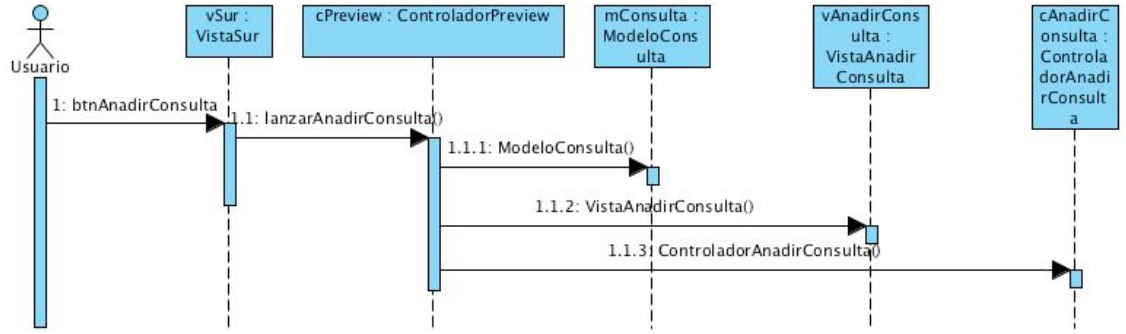



Diagrama de secuencias probar consulta

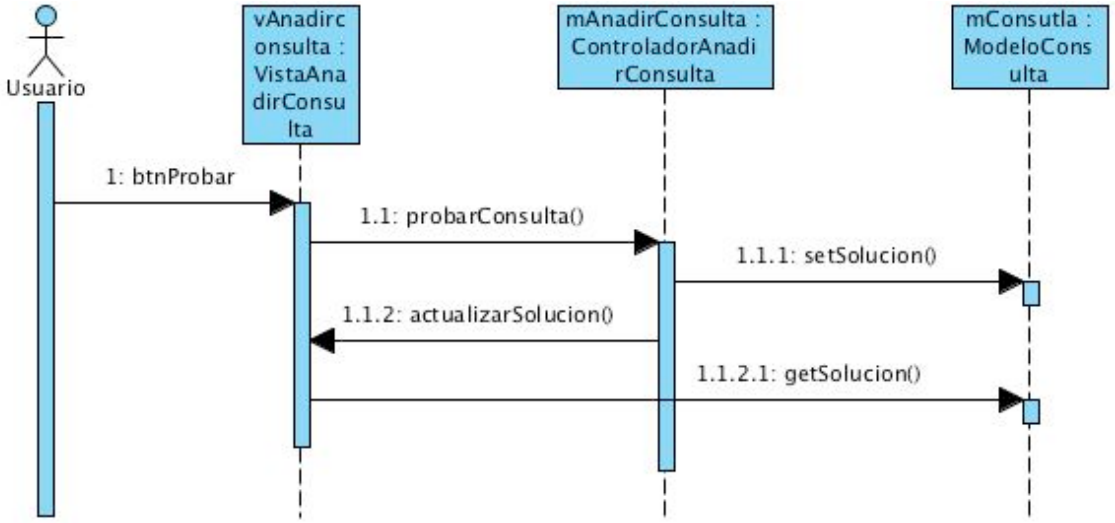


Diagrama de secuencias eliminar consulta

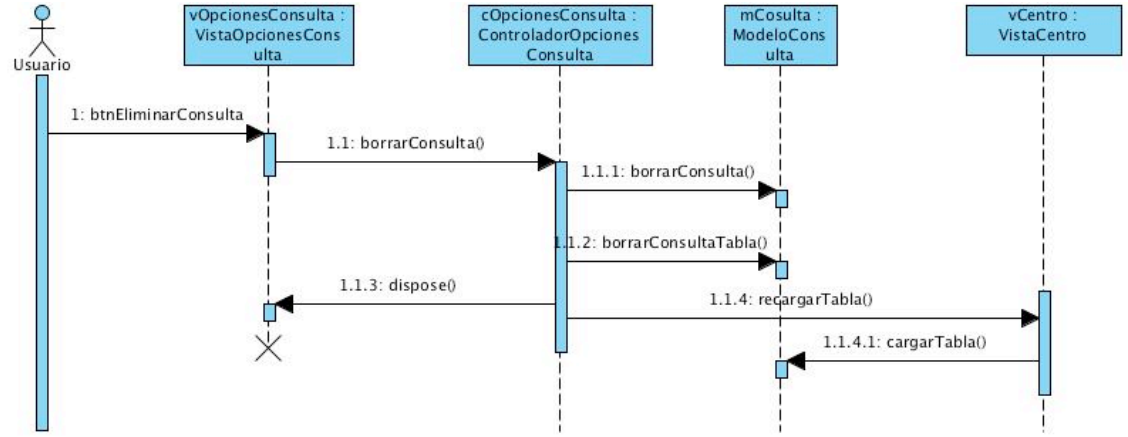



Diagrama de secuencias modificar consulta

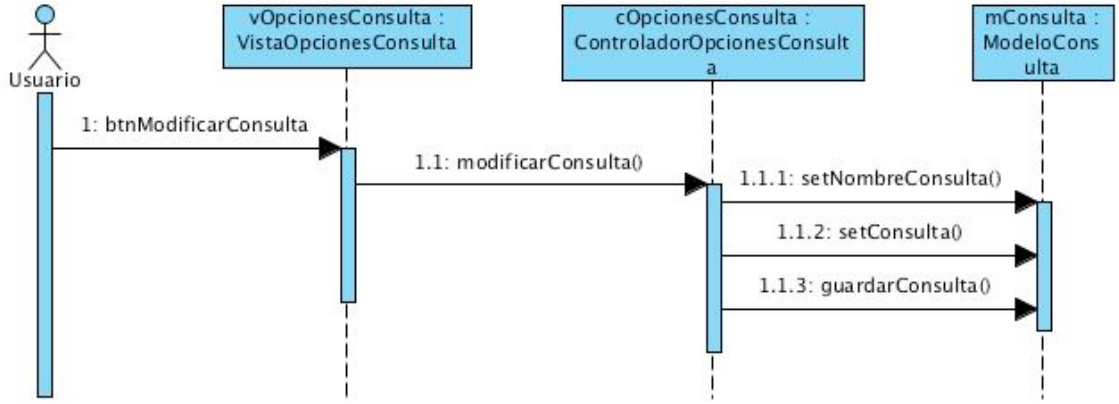



Diagrama de secuencias guardar consulta

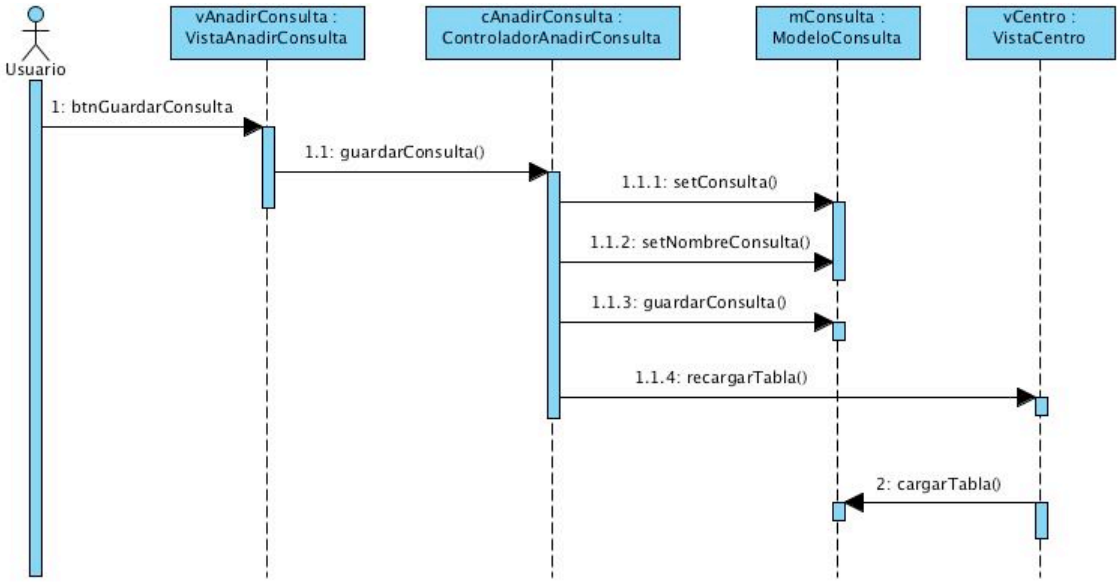
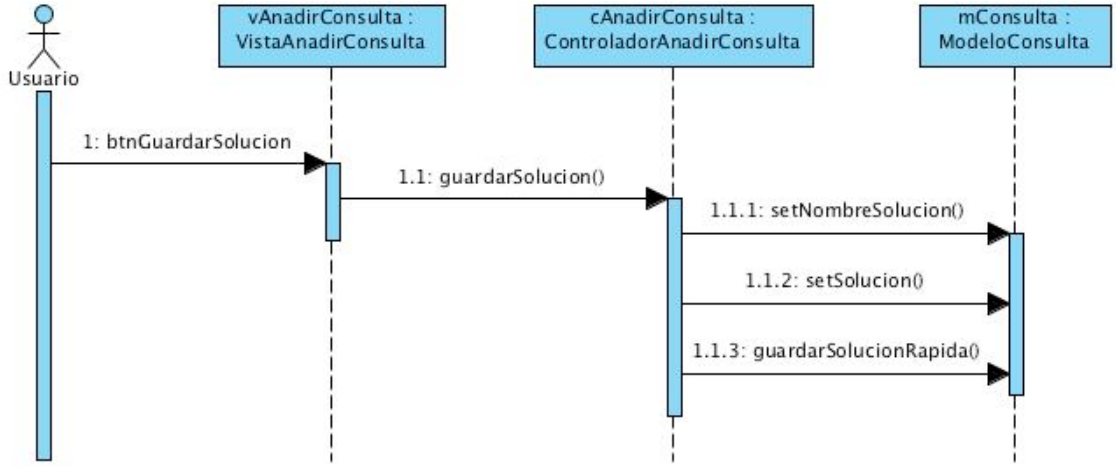
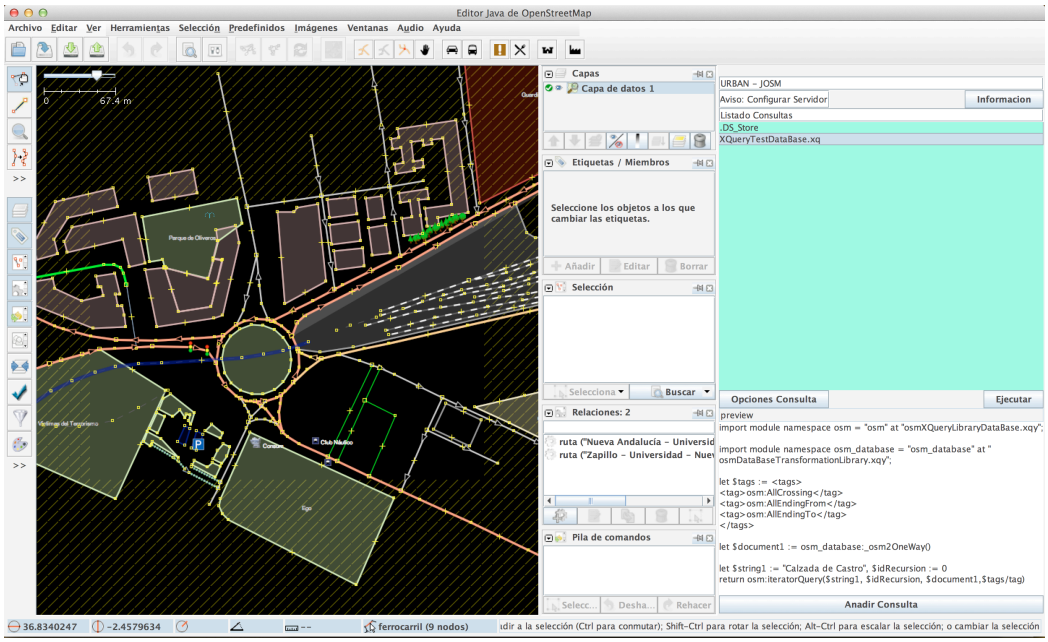


Diagrama de secuencias guardar solución



# Anexo II – Tutorial aplicación.

Dentro del JOSM se carga el plugin automáticamente dando como resultado la siguiente imagen.



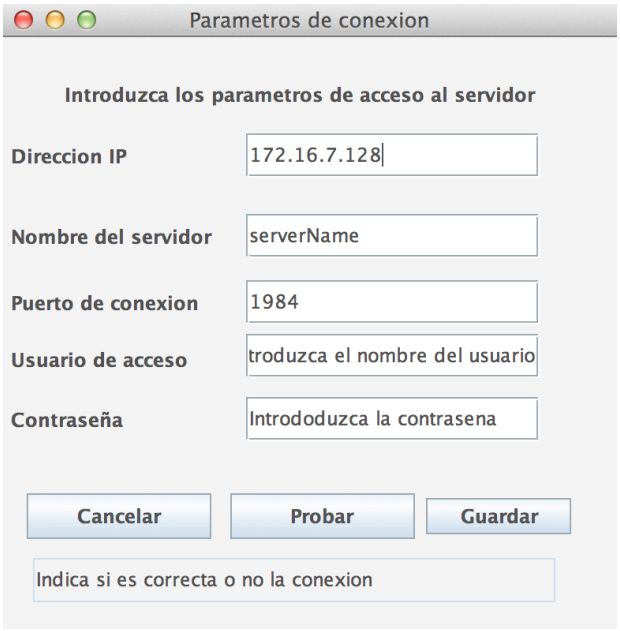

El plugin se compone de tres partes principales, una zona superior en la que se puede acceder a la información del mismo, una zona de consultas situada en el centro del programa y una zona de previsualización de consultas situada en la zona inferior.


La vista principal del plugin es la siguiente:

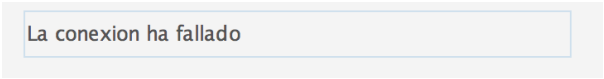
El panel superior es la primera de las zonas la cual tiene el siguiente aspecto:


En la zona superior únicamente tenemos acceso a un botón de información, el cual nos muestra, si lo pulsamos, una ventana de información sobre este plugin, un pequeño resumen acerca de JOSM-Urban, así como su icono, un botón para configurar las opciones del servidor y otro para acceder al tutorial.

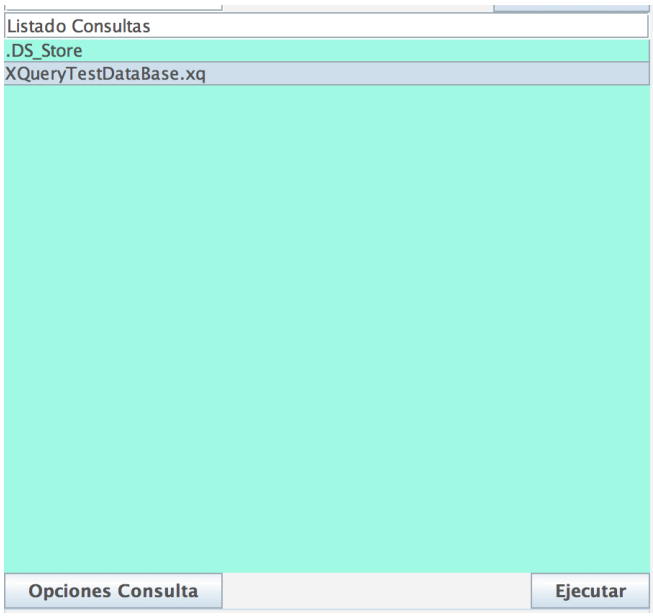
Si pulsamos en el botón de Opciones de servidor, nos aparece la siguiente ventana, en la que podremos modificar todos los datos que aparecen. Se guarda en un archivo .txt los parámetros introducidos. En caso de no introducir ningún dato, se guardará un archivo .txt con los parámetros de conexión por defecto para conectarse a una máquina virtual. Dicho archivo se guarda automáticamente en la carpeta raíz del plugin.




En la parte inferior saldrá el resultado de probar la conexión indicándonos si se realiza con éxito o no.



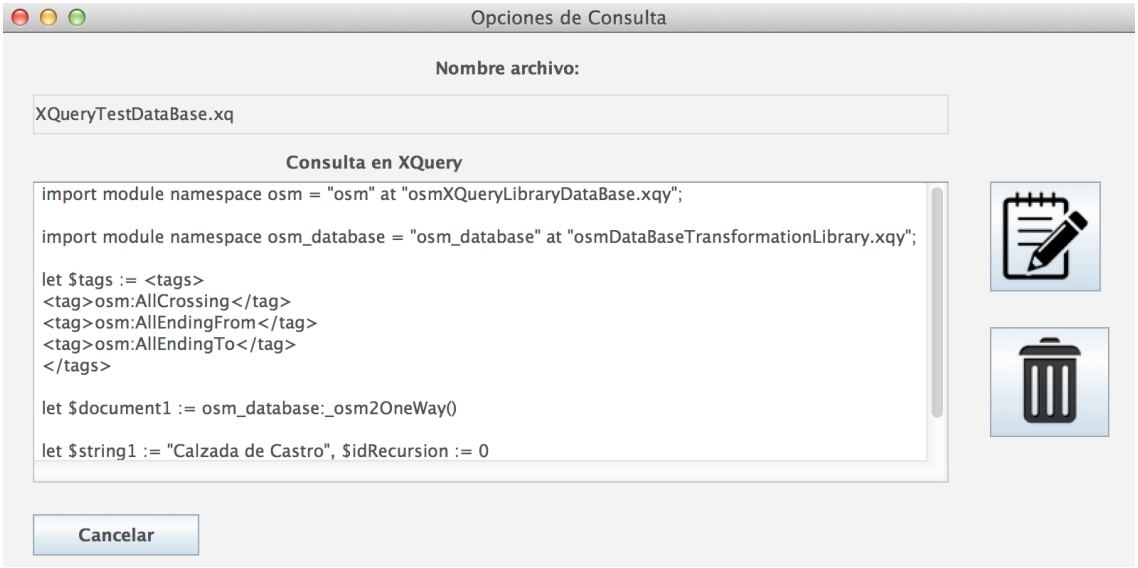

En la zona intermedia podremos ver un listado de las consultas cargadas en el plugin. Para poder ejecutar una consulta, así como para acceder a todas las opciones que ofrece el mismo, es necesario seleccionar una consulta de la lista.




Si se pulsa el botón ejecutar con una consulta pulsada , se ejecuta la consulta y se guarda la solución en la carpeta JOSM-URBAN/Soluciones con el nombre "CapaActivaSolucion.OSM" a la vez que se carga en el programa.

Se pueden añadir consultas manualmente al plugin, solamente hay que copiar los archivos en la carpeta JOSM-URBAN/Consultas y se mostrarán en la lista al reiniciar el plugin.

Si se selecciona una consulta de la lista y hacemos click en Opciones de Consulta, se muestra una ventana en la que solamente se permite modificar el contenido de la consulta seleccionada. También se muestran dos iconos, uno para guardar la modificación y otro para eliminar la consulta. Dicha opción elimina el archivo de la lista de consultas y el archivo del disco.

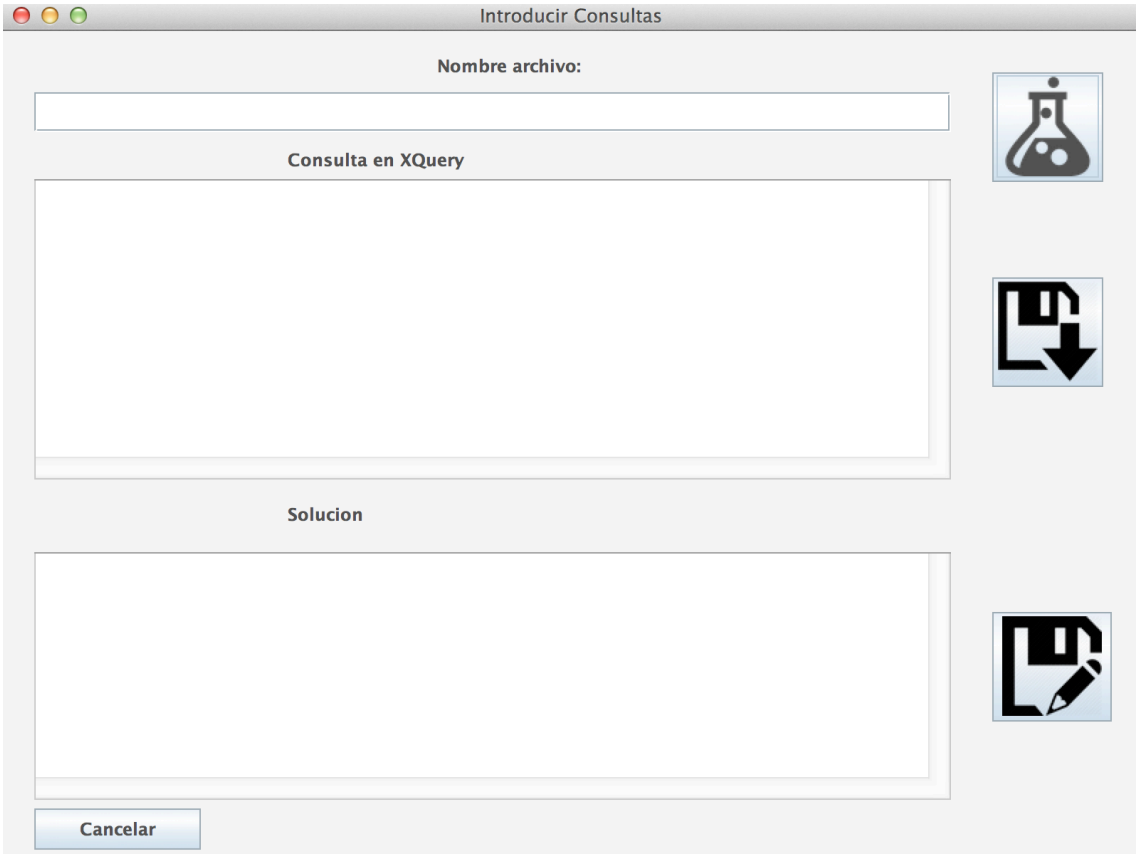



En la zona inferior, se muestra una ventana de previsualización de la consulta seleccionada de la lista de consultas así como un botón para poder añadir una nueva consulta.


Si pulsamos el botón añadir consulta, nos permite crear una nueva consulta, guardar la

consulta realizada, y el resultado de la misma, utilizando los botones que tenemos anexos a donde se muestran las consultas. Se permite probar una consulta y mostrar el código de solución en el área de solución con el botón ubicado al lado del nombre del archivo.

Para poder guardar una consulta o una solución es necesario introducir un nombre de archivo.



# URBAN-JOSM

JUAN CARLOS FERNÁNDEZ ZAMORA

UNIVERSIDAD DE ALMERÍA


## *Resumen*

El principal objetivo de este PFG es la creación de un plugin que permita realizar consultas urbanas complejas en Xquery para el programa JOSM

Este plugin permite la utilización de los lenguajes de consulta XPath y Xquery para realizar las consultas, además de incluir la librería de operadores topológicos propuesta por A. Becerra y. J. Almendros

El plugin será desarrollado en lenguaje Java, usando diferentes librerías. Seguirá una arquitectura cliente-servidor, utilizando como servidor de Xquery el programa BaseX. Seguirá las directrices marcadas por IPO en la interfaz gráfica, desarrollado bajo el patrón arquitectónico **ModeloVistaControlador.**

En el proyecto se siguen las diferentes etapas de desarrollo del software (planificación, modelado, codificación, pruebas y documentación) explicando el trabajo desarrollado en cada u una de las fases.

Entre las principales funcionalidades se encuentran, configurar la información de la conexión por el usuario, una **previsualización de consultas** y su ejecución rápida, guardando las soluciones generadas, además de mostrar las consultas en forma de lista y otras funcionalidades básicas como eliminar, añadir y modificar consultas.

Dentro del sistema existe, un tutorial explicativo sobre su utilización, Este tutorial muestra la información detallada acerca de sus funcionalidades.

## *Abstract*

The main target of this TFG is a plug ing development that allows performing complex urban queries for the JOSM programme.

This plugin allows the use of XQuery and XPath query languages to execute queries, besides including the topological operators library proposed by A. Becerra and J. Almendros.

The plugin is developed in Java using different libraries. It follows a Client-Server arquitecture, using BaseX as an XQuery server. The GUI follows the guidelines set by IPO. In addition, uses a View Model Controller pattern arquitecture.

The different stages of development that are followed in this proyect (planning, modeling, construction, test and documentation) explaining the developed work in each one of the stages.

The main functionalities that can be found: organize the information about the user's connection, show up a query preview, save the solution generated by the programme, besides showing up the queries on a list and many other basic functionalities such as delete, create and modify queries.

It exists inside the system an explanatory tutorial guide. This tutorial shows the detailed information about its functionalities.


---

GRADO EN INGENIERÍA INFORMÁTICA, 2013/2014