Universidad Autónoma Metropolitana Unidad Azcapotzalco
División de Ciencias Básicas e Ingeniería
Licenciatura en Ingeniería en Computación

Buscador semántico de recursos académicos de la DCBI en una plataforma de cómputo móvil.

Henoch Cruz Carrera
Matrícula: 208367636

Asesores

Dra. Maricela Claudia Bravo Contreras
Profesor Asociado
Departamento de Sistemas

Dra. María Lizbeth Gallardo López
Profesor Asociado
Departamento de Sistemas

Índice de contenido

Índice de figuras

Índice de códigos

1. Introducción

El significado (semántica) de la información permite ”entender” y satisfacer las necesidades de búsqueda de información en la web, ir más allá de la simple estructura sintáctica, dando lugar a lo que se conoce como la web semántica[1], en donde los recursos están anotados de tal forma que las computadoras pueden comprender su significado y su función.

Un fundamento que se usa en la web semántica es la ontología, la cual se puede definir como la forma en que se describen los conceptos del mundo o de algún dominio dado, algunas de sus propiedades y las relaciones que existen entre estos conceptos, y está especificada en un lenguaje formal que puede ser procesada por computadoras y no solo por personas, ver Figura 1.

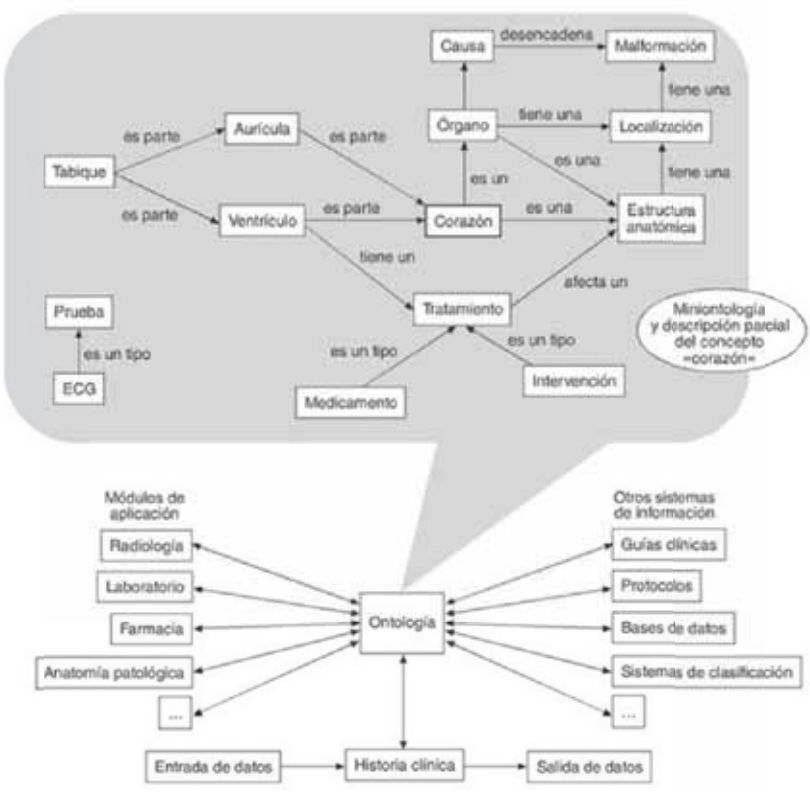


Figura 1: Ejemplo de una ontología

En muchas ocasiones se realizan búsquedas de información que está dispersa por la red, y en algunos casos dicha información no está relacionada con lo que realmente se busca, para esto se puede usar una ontología, relacionando los datos y dando un resultado más preciso a la búsqueda.

Al realizar una búsqueda en un buscador convencional los resultados son obtenidos a través de métodos estadísticos de clasificación, por este motivo los usuarios no siempre consiguen la mejor información, y en muchas ocasiones consiguen aquella mejor posicionada y optimizada.

Al usar un buscador semántico que use el fundamento de ontología se tiene una nueva manera de gestionar la información, y los resultados obtenidos son más aproximados a los criterios de búsqueda, un buscador semántico puede estar en varias plataformas ya sea una aplicación de escritorio, en la web, o en dispositivos móviles (teléfonos inteligentes, tabletas electrónicas).

Por esta razón este proyecto propone realizar un buscador semántico en una plataforma de cómputo móvil dentro de un entorno académico contestando preguntas por medio de palabras clave por ejemplo: ubicación del cubículo de un profesor, formato para servicio social, coordinador de la ingeniería en computación.

Es importante, para comprender mejor el objeto de este trabajo mostrar un marco de referencia sobre los temas que se relacionan con el desarrollo de este proyecto.

La Web semántica y los servicios Web semánticos, ambos son una extensión de la Web tradicional en donde estos recursos están anotados de tal forma que puedan ser comprendidas por las personas y las computadoras, también se puede ver como a las Web Semántica como una interesante opción para integrar aplicaciones[2].

Servicios Web, ¿Qué es un servicio Web?, existen múltiples definiciones sobre lo que son los Servicios Web, lo que muestra su complejidad a la hora de dar una adecuada definición que englobe todo lo que son e implican. Una posible sería hablar de ellos como un conjunto de aplicaciones o de tecnologías con capacidad para interoperar en la Web. Estas aplicaciones o tecnologías intercambian datos entre sí con el objetivo de ofrecer unos servicios. Los proveedores ofrecen sus servicios como procedimientos remotos y los usuarios solicitan un servicio llamando a estos procedimientos a través de la Web[3]. Esto se ilustra en la Figura 2.

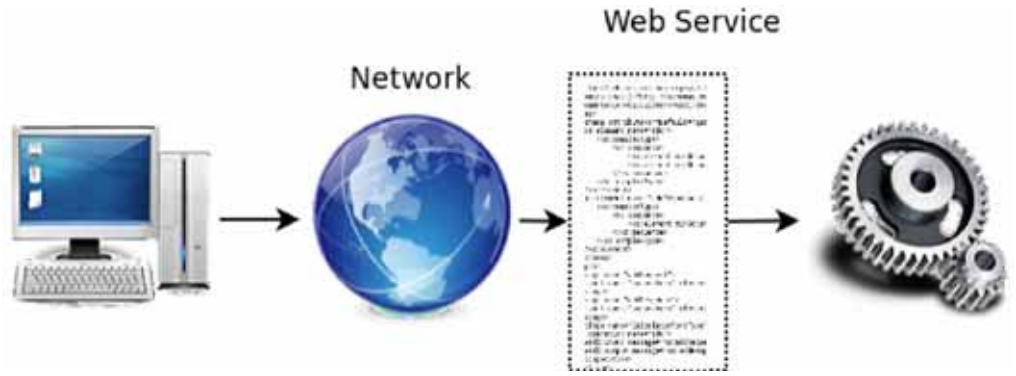


Figura 2: Acceso a un servicio web usando tecnologías estándar de Internet

Las aplicaciones móviles son extensiones informáticas para dispositivos portátiles, tales como teléfonos inteligentes o tabletas.

1.1. Problemática

Actualmente la UAM Azcapotzalco no cuenta con un buscador semántico; un sistema de cómputo móvil que realice búsquedas semánticas sería de gran ayuda a los alumnos y docentes, porque se respondería de una forma sencilla y rápida a sus preguntas, reduciendo el tiempo de búsqueda, por ejemplo saber los formatos para realizar el trámite del servicio social o saber la ubicación de la coordinación de movilidad, posteriormente se puede realizar un sistema de búsqueda en diferentes

plataformas y ampliar el dominio en donde se realizan las búsquedas.

1.2. Objetivos

1.2.1. Objetivo general

Diseñar e implementar un sistema de cómputo móvil que permita realizar búsquedas de información acerca de recursos (oficinas administrativas, trámites para alumnos, ubicación de cubículos de profesores) considerando un entorno académico de la División de Ciencias Básicas e Ingeniería de la UAM Azcapotzalco (CBI).

1.2.2. Objetivos específicos

Responder preguntas de un usuario por medio de palabras clave.

Desarrollar una aplicación móvil que realice búsquedas con base a las palabras que el usuario proporcione.

Realizar búsquedas en una ontología[4] con datos de la UAM Azcapotzalco, mostrar los resultados más relacionados con las palabras clave. Realizar pruebas de conexión entre el móvil y el servidor; realizar pruebas sobre la búsqueda y la representación de la información para el usuario; así como identificar fallos que se puedan presentar en la aplicación móvil.

Diseñar una interfaz de usuario de fácil uso e intuitivo.

Recopilar y registrar información acerca de los recursos académicos de CBI.

2. Análisis y Diseño del sistema

2.1. Arquitectura general del sistema

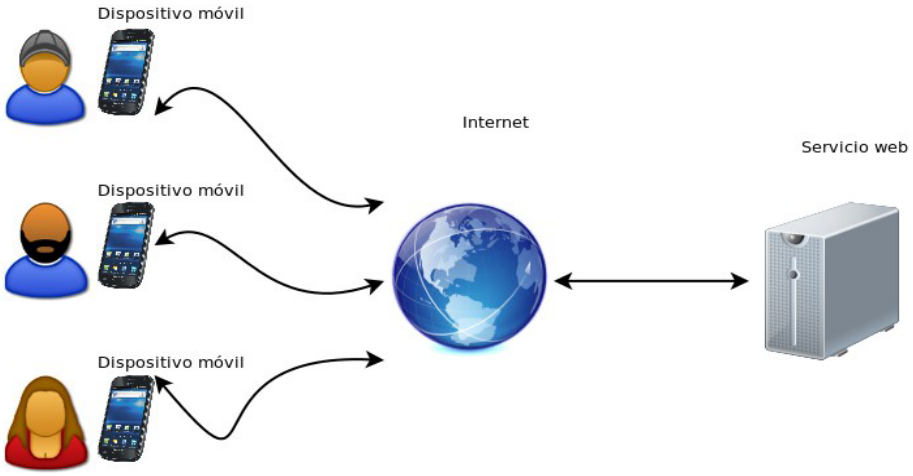


Figura 3: Arquitectura general del sistema

En la Figura 3 se muestra la arquitectura general de lo que es el sistema, por uno lado se encuentra el servidor que actuará proporcionando el servicio web para que la aplicación móvil consuma dicho servicio.

Dentro del servidor se aloja el servicio web así como la base de conocimiento, esta base está compuesta de un motor de inferencias, un motor de consultas, las reglas SWRL (*Semantic Web Rule Language*)[5] y por una serie de ontologías que describen el contexto de la búsqueda.

En el lado del cliente se encuentra la interfaz de navegación que consta de menús, recursos para poder consumir el servicio web y de la interfaz donde el usuario captura la pregunta a base de palabras clave para ser enviada al servicio web y obtener la respuesta.

2.1.1. Diseño por capas

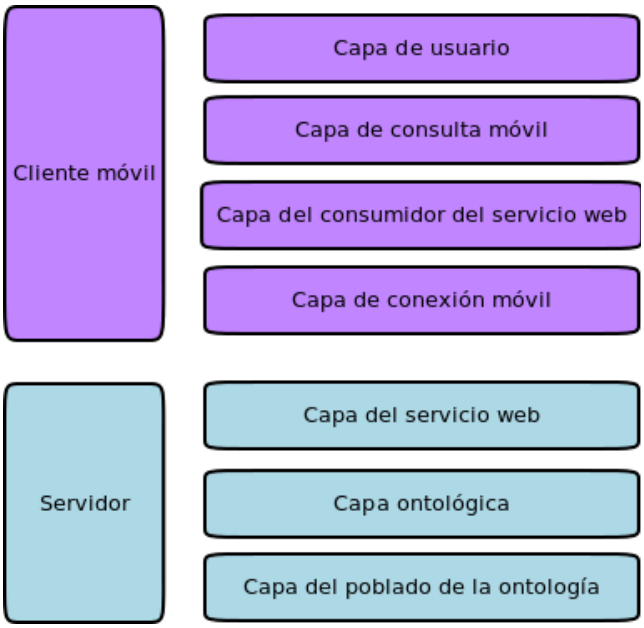


Figura 4: Capas de la arquitectura

Para entender de una forma más sencilla a la arquitectura se presenta un diagrama por capas, la Figura 4 muestra las capas que intervienen en el lado del cliente móvil y los que están en el lado del servidor.

2.1.1.1. Capa de usuario

El usuario dentro de la interfaz del programa realiza una búsqueda dentro de un cuadro de texto, en este módulo también se mostrarán los resultados que se recibirán del servidor en pantalla.

2.1.1.2. Capa de consulta móvil

Se toma las palabras ingresadas por el usuario y se procesa para mandarla a la capa del consumidor web donde se crea una petición para el servicio web.

2.1.1.3. Capa de conexión móvil

Esta parte es donde se realizará la conexión entre el sistema cliente del dispositivo móvil y el servidor, inicialmente se pretendió en implementar la comunicación con un protocolo de conexión TCP/IP con el uso de *sockets* [1], posteriormente en la implementación se realizó la interacción con el servicio web

1 Método para la comunicación entre un programa del cliente y un programa del servidor en una red. Un *socket* se define como el punto final en una conexión. Los *sockets* se crean y se utilizan con un sistema de peticiones o de *llamadas de función.*

usando SOAP (*Simple Object Access Protocol*)[6], enviado a través del protocolo HTTP (*Hypertext Transfer Protocol*) con una serialización XML[7] .

2.1.1.4. Capa del servicio web

Capa que se encarga de mediar las peticiones de los usuarios y el sistema de consulta que se encuentra en el servidor.

Se toman la peticiones de búsqueda descritas con palabras clave y se transforman en reglas SWRL para poder ser procesadas en la capa de la consulta en el servidor.

Esta capa se encarga de tomar las reglas y procesarlas dentro de las ontologías, la sintaxis de las reglas tienen la siguiente forma:

Antecedentes ⇒ Cuerpo.

Un ejemplo:

```
Tramites:Tramites(?x)
^ Tramites:TieneNombre(?x, ?Nombre_tramite)
^ Tramites:TieneGuia(?x, ?Guia)
 ^ swrlb:containsIgnoreCase(?Nombre_tramite, "credencial")
-> sqwrl:select(?Nombre_tramite,?Guia)
```

Lo cual en el ejemplo anterior da como resultado la búsqueda de trámites que contengan la palabra "**credencial**", el resultado obtenido está conformado por el nombre del trámite y la URL de la guía que tiene asignado, esto se regresa al cliente del dispositivo móvil y lo presentará como resultado.

2.1.1.5. Capa Ontológica

En esta capa se administra la base de conocimiento en el contexto de la División de Ciencias Básicas e Ingeniería de la Universidad Autónoma Metropolitana.

2.1.1.6. Almacenamiento y Población de la ontología

Se realiza el almacenamiento de los individuos (personas, lugares, trámites) que forman parte de la ontología, se capturan los datos y las relaciones que existen entre estos datos, así como las actualizaciones de la ontología.

Ejemplos de datos que se capturarán para la población de la ontología ver Tabla 1:

Ontologías	Clases que lo conforman
Persona	Alumnos, profesores.
Espacio físico	Áreas, edificios, salones.
Trámites	Seguro facultativo, servicio social, movilidad, credencial

Tabla 1: Ontologías y clases que conforman el modelo

2.2. Modelado de la Capa del cliente móvil

2.2.1. Casos de uso de capa de usuario

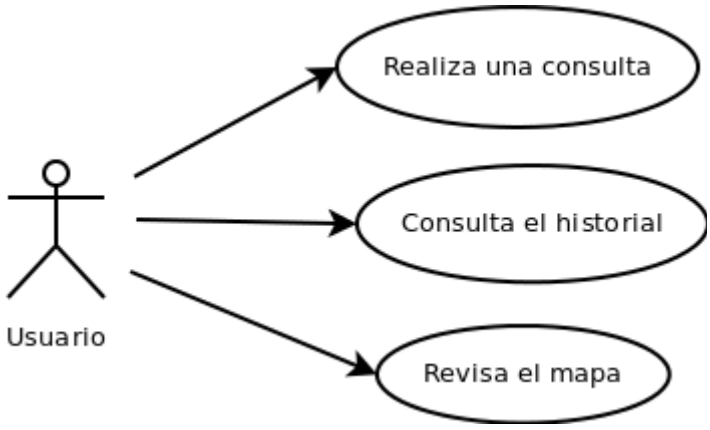


Figura 5: Casos de uso de capa de usuario

Actores

- Usuario. Es quien realiza una consulta, lo hace por medio de palabras clave que ingresa en un cuadro de texto a través de un dispositivo móvil.

Casos de Uso

1. Realiza una consulta. Se activa por el usuario cuando este quiere realizar una consulta, se realiza desde el dispositivo móvil.
2. Consulta el historial. Cuando el usuario quiere ver las preguntas y las respuestas a las peticiones que realizó previamente.
3. Revisar el mapa. Permite ver al usuario mapas de la unidad.

2.2.2. Secuencias de la capa de usuario

En la Figura 6 se muestra el diagrama de secuencia de la capa de usuario donde se nota el menú y las opciones que tiene.

1. Se tiene el menú con las opciones.
2. Si se selecciona la opción “Consulta”.
3. Si se selecciona la opción “Historial”.
4. Si se selecciona la opción “Mapa”.
5. Se inicia la actividad seleccionada.

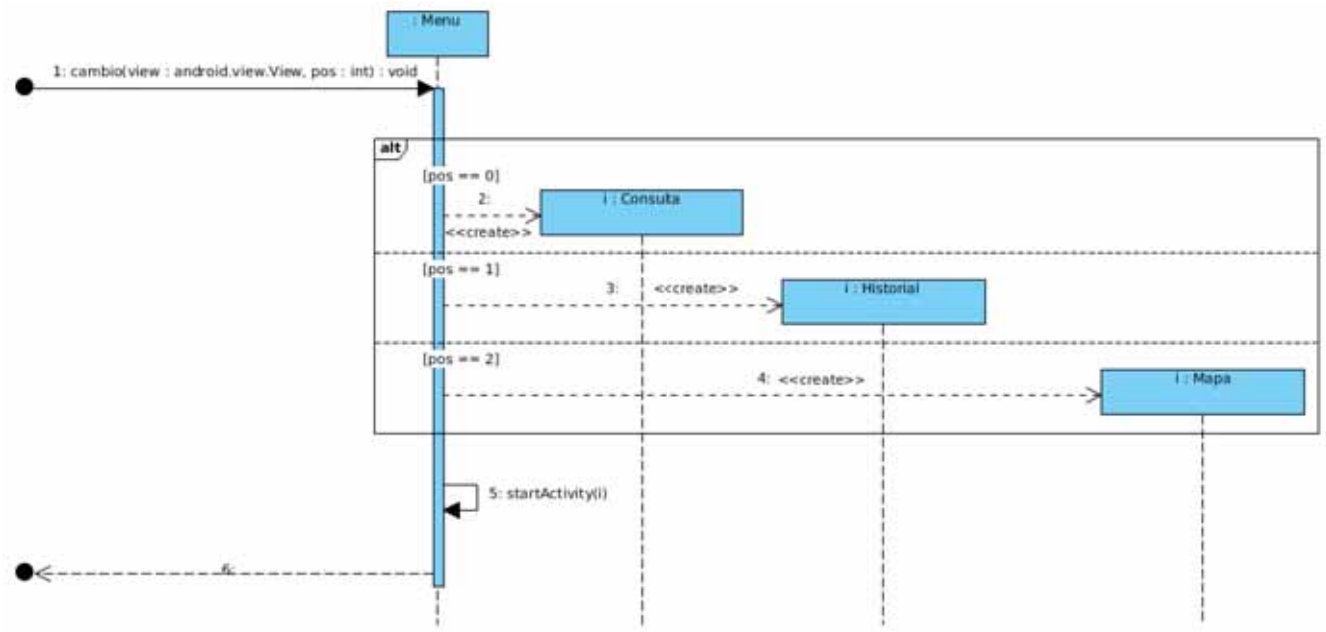


Figura 6: Diagrama de secuencia capa de usuario

2.2.3. Secuencias de la capa de consulta móvil

Secuencia de consulta

La secuencia con que se realiza la consulta se puede ver en la Figura 7.

1. El usuario selecciona la opción consulta del menú principal de la aplicación.
2. Se obtiene la pregunta que el usuario escribió en el *EditText*, se presiona el botón de envío.
3. Se verifica si la cadena tiene algo escrito para poder ser enviado, en caso de que no contenga caracteres no se realiza el envío del mensaje, si contiene caracteres se procede a enviar el mensaje al servicio web.
4. Se añade el mensaje a una lista para poder ser guardada en el historial de consultas.
5. Se enviá el mensaje al módulo de envío del mensaje al servicio web.
6. Se ejecuta el módulo de envío del mensaje al servicio web.
7. Si el resultado obtenido es diferente a un `NULL` entonces termina mostrando el resultado.
8. Si es `NULL` esto quiere decir que ocurrió un error y regresa una cadena que dice “Error al conectarse con el servidor”

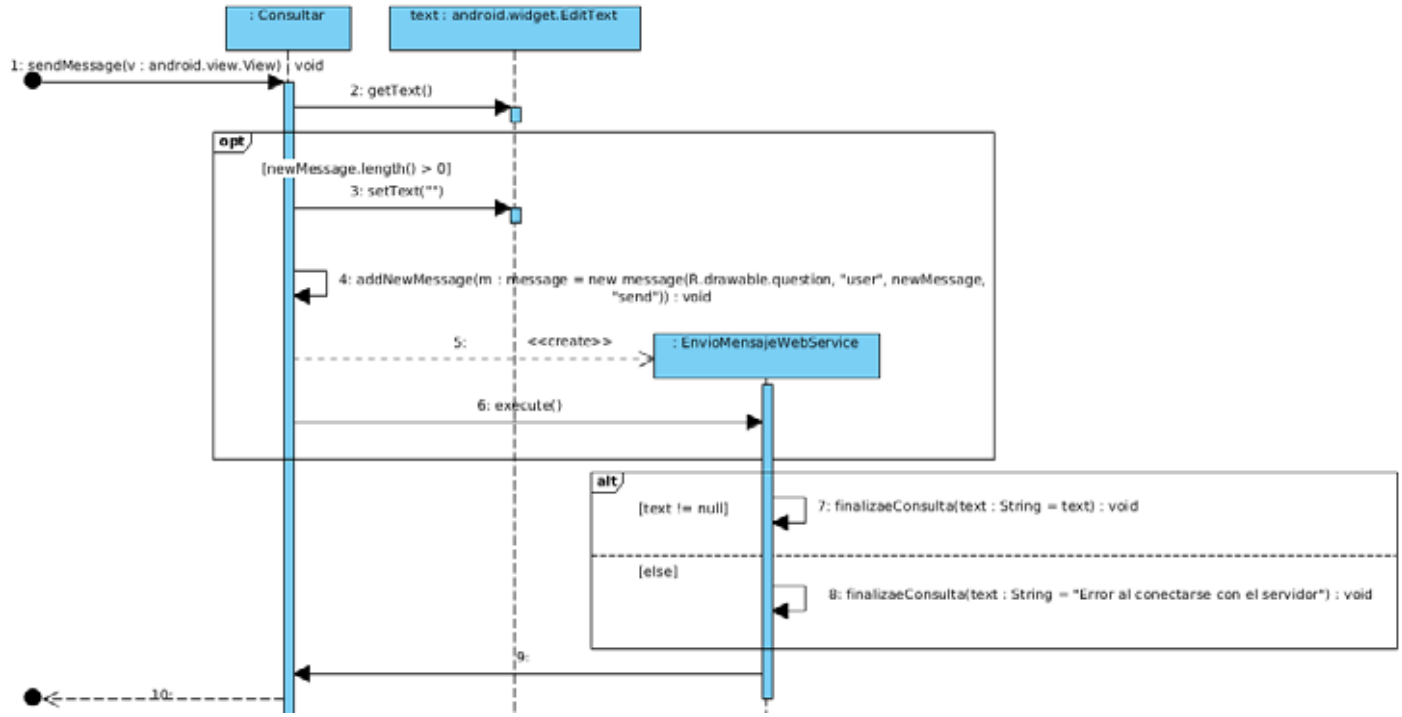


Figura 7: Diagrama de secuencia de consulta

2.2.4. Secuencia de la capa del consumidor del servicio web

Para la consulta al servicio web se tiene el módulo del consumidor detallado en la Figura 8.

1. Dentro del consumidor está la operación `consultarWs` donde se recibe una cadena.
2. Se instancia un `request`[8] (solicitud).
3. Se anota en el `log` si se creó correctamente el `request`.
4. Se añaden los parámetros que son requeridos por el servicio web.
5. Se crea un `envelope` (envoltorio).
6. Se anota en el `log` si se creó correctamente el `envelope`.
7. Se envuelve el `request` con el `envelope`.
8. Se anota en el `log` si se envolvió correctamente el `request`.
9. Se crea un transporte con el protocolo HTTP.
10. Se anota en el `log` si se creó correctamente el transporte.
11. Se realiza la llamada del transporte mandando el nombre de la consulta (`SOAP_ACTION`) y el `envelope`.
12. Se anota en el `log` si se realizó la llamada correctamente.
13. Una vez realizada la llamada, se recupera las respuesta que el servicio web mandó.

14. Se anota en el `log` si se obtuvo la repuesta del servicio web.
15. Se recupera la cadena y se devuelve.

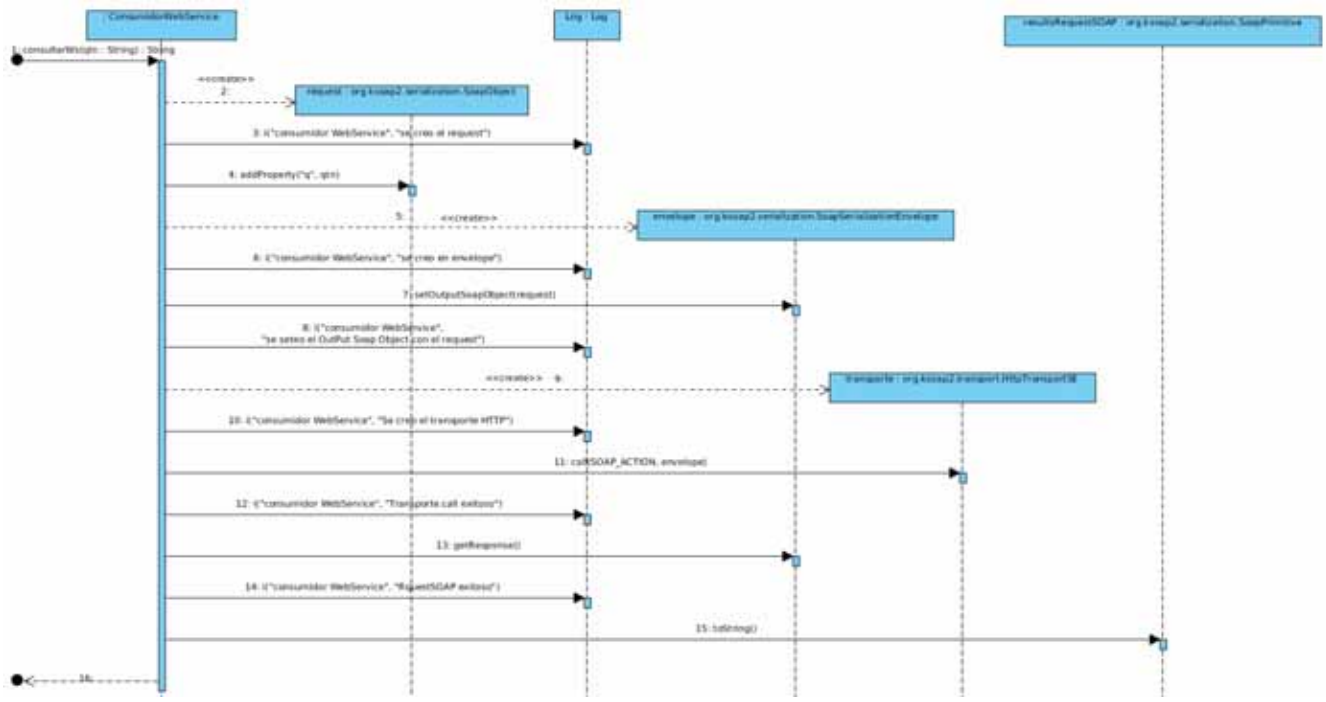


Figura 8: Diagrama de secuencia de la capa del consumidor del servicio web

2.2.5. Secuencia de la capa de conexión móvil

Para describir la secuencia en la capa de conexión ver Figura 9, donde se detalla como se realiza la conexión de la aplicación móvil al servicio web por medio de Internet

1. Se manda la búsqueda en formato de palabras clave hacia el consumidor de servicios web
2. Envío del mensaje SOAP.
3. El servicio web realiza la consulta y regresa el resultado.

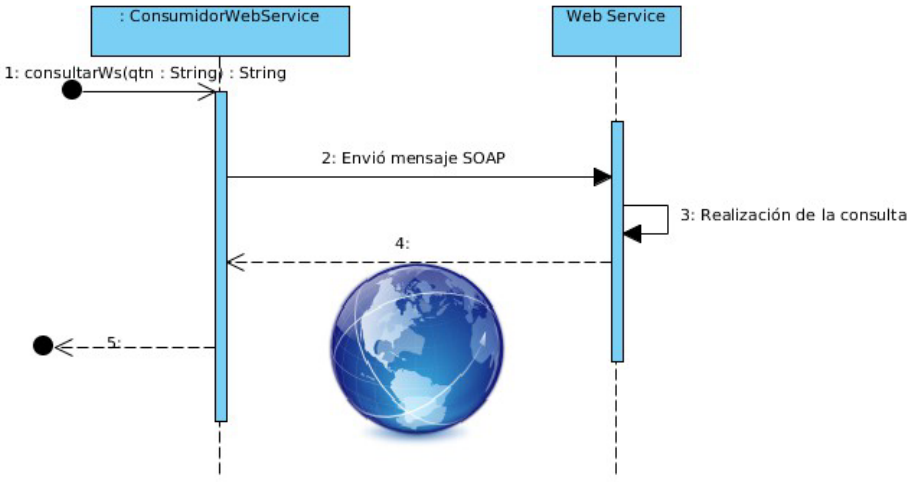


Figura 9: Diagrama de secuencia capa de conexión móvil

2.3. Modelado de la Capa del servidor

Diagrama de secuencia de la capa del servicio web

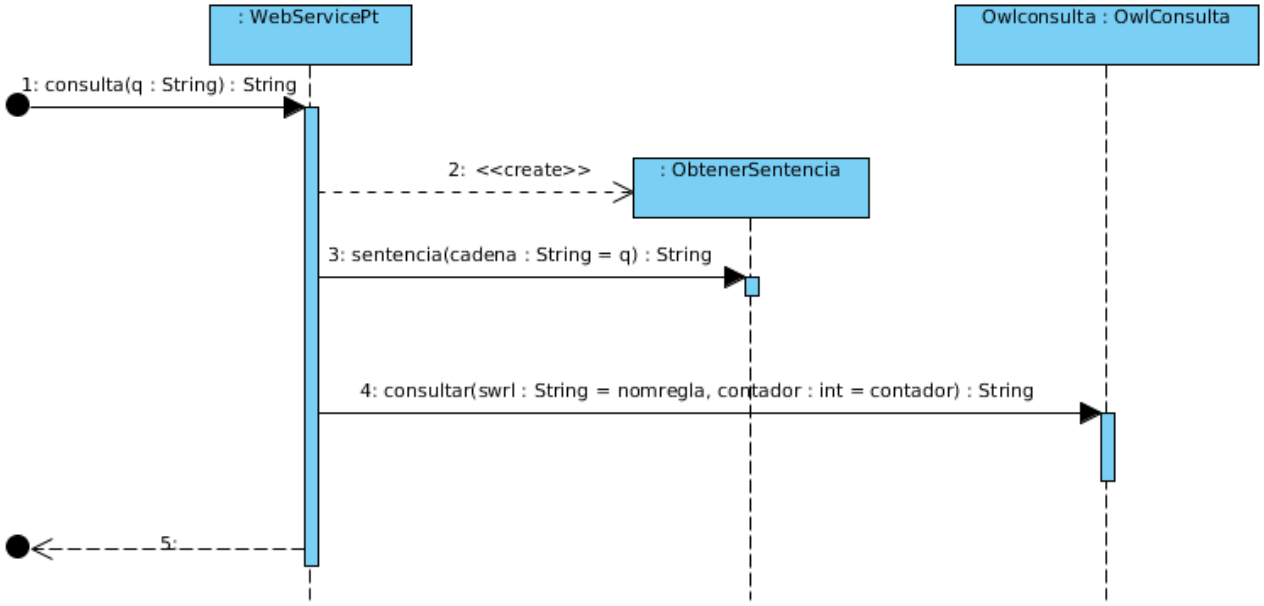


Figura 10: Diagrama de secuencia de la capa del servicio web

En la Figura 10 se muestra la secuencia de acciones que se llevan acabo en el servidor para realizar la consulta a la ontología.

1. Llega la petición de consulta del cliente móvil al servicio web.
2. Se crea una instancia para obtener la sentencia a partir de la pregunta con las palabras clave.
3. Se busca la sentencia en SWRL correspondiente a las palabras claves que forman la pregunta del usuario.
4. Se manda la sentencia SWRL para realizar la consulta a la ontología y se obtiene la respuesta.

2.4. Modelado de la capa ontológica

En esta capa se administra la base de conocimiento en el contexto de la División de Ciencias Básicas e Ingeniería, el modelo consta de 5 ontologías.

1. Ontología Persona
2. Ontología Espacio Físico
3. Ontología Académico CBI
4. Ontología Red
5. Ontología trámites

2.4.1. Ontología persona

Esta ontología representa a todo el personal involucrado en la base de conocimiento estos son:

- `Alumno`
  - `Licenciatura`
  - `Posgrado`
- `Empleado`
  - `Academico`
    - `Ayudante`
    - `Profesor`
  - `Administrativo`
  - `Servicio`
- `Visitante`

Y las propiedades de la clase `Persona`.

- `tieneNombre`
- `tieneApellido`
- `tieneEdad`
- `tieneGenero`
- `tieneDatosDeContacto`

Si es alumno contara aparte con las propiedades anteriores tiene la propiedad de `Matricula`.

En caso de que sea profesor se añadirá las propiedades de.

- `tieneNumeroEconomico`
- `tieneIndiceDeReprobación`

En la Figura 11 se muestra el diseño que se usó para la ontología persona con sus propiedades y relaciones.

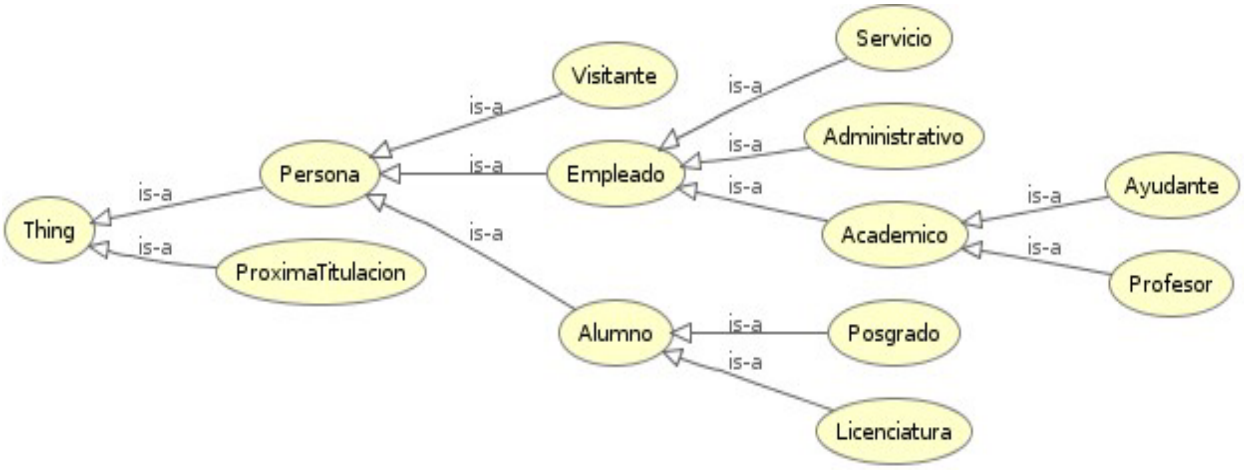


Figura 11: Diseño de la ontología persona

2.4.2. Ontología espacio físico

En la ontología espacio físico las clases que lo conforman son las siguientes:

- `Areas`
- `Salon`
- `Edificio`
- `Cubiculo`

Las propiedades de las clases son las siguientes:

- `AccesoDisponible ( TRUE, FALSE)`
- `HorarioInicio`
- `HorarioFinal`
- `Ubicacion`

Para la clase `Areas` se le añade una propiedad nombre del área.

Para la clase `Edificio` se le añade la propiedad tiene letra.

Para la clase `Cubiculo` se le añade la propiedad tiene nombre cubículo.

Para la clase `Salon` se le añade 3 propiedades extras nombre de salón, numero de salón y cupo.

Se puede apreciar el diseño de la ontología en Figura 12.

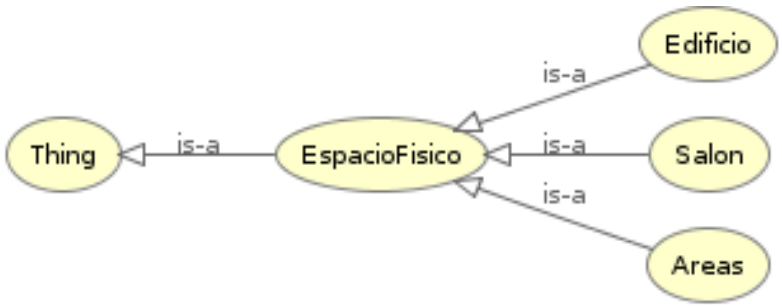


Figura 12: Diseño de la ontología espacio físico

2.4.3. Ontología académico CBI

Ontología que contiene los recursos de CBI: las coordinaciones, los departamentos, los programas académicos, las licenciaturas, así como las UEA que conforman cada tronco de la carrera.

Las clase que lo conforman son las siguientes:

- `Coordinacion`
- `Departamento`
- `ProgramaAcademico`
  - `Licenciatura`
  - `Posgrado`
    - `Doctorado`
    - `Maestria`
- `UEAPlanEstudio`
  - `TroncoInterMultidisciplinar`
    - `UEAObligatorias`
    - `UEAOptativas`
      - `OtrasIM`
      - `UEAArteHumanidades`
      - `UEAEstudiosCulturales`
      - `UEAFormacionCiudadana`
      - `UEAIntroduccionMercadoLaboral`
      - `UEALenguajesFormales`
  - `UEATroncoBasicoProfesional`
  - `UEATroncoGeneral`
  - `UEATroncoIntegracion`

        - UEAobligatorias
        - UEAoptativas
            - UEACientificoTecnicas
                - UEAAreaConcentracionAlgoritmosInteligenciaArtificial
                - UEAAreaConcentracionMecatronica
                - UEAAreaConcentracionSeguridadRedesComputadoras
                - UEAAreaConcentracionSistemasEmbebidos
                - UEAAreaConcentracionSistemasInformacion
            - UEAMovilidad
            - UEAOtrasIntegracion
            - UEATutoriales
    - UEATroncoNivelacionAcademica

Las propiedades de la clase de UeaPlanEstudio

- tieneClaveUEA
- tieneCreditosAsignados
- tieneNombreUEA

Las propiedades para la clase Doctorado.

- TieneDisertacionPublica
- tieneExamenPreDoctoral

Las propiedades para la clase Maestria

- tieneExamenGrado

Las propiedades para la clase de Licenciatura.

- tienePropuestaProyectoTerminal
- TieneNombreCarrera
- tieneProyectoTerminal
- tieneServicioSocial

Las propiedades para la clase Coordinacion

- tieneNombreCoordinacion

Las propiedades para la clase Departamento

- tieneNombreDepartamento

Las propiedades para la clase Posgrado

- `tieneNombreGrado`
- `tieneNombreTesis`

En la Figura 13 se muestra con más detalle el diseño de la ontología.

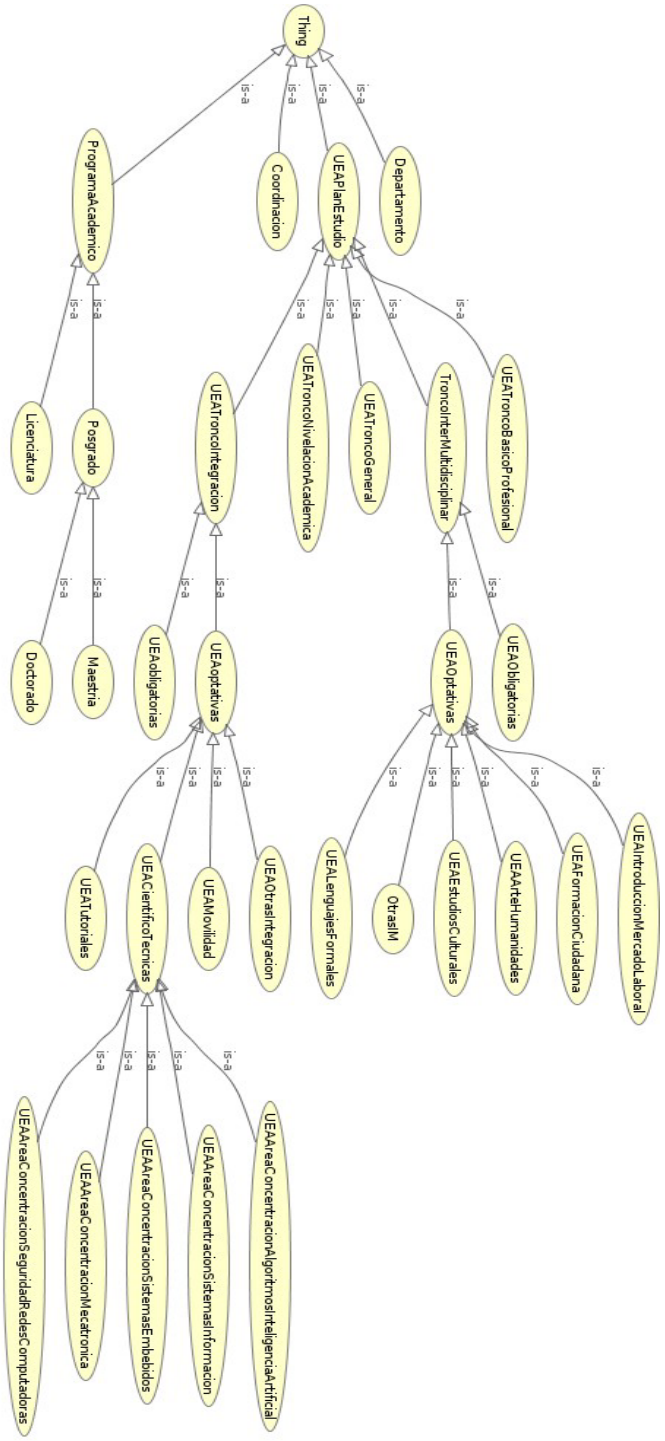


Figura 13: Diseño de la ontología académico CBI

2.4.4. Ontología red

La ontología red hace uso de las ontologías: persona, espacio físico, académico CBI, para crear relaciones entre ellas consta de las clases:

- `Clase`
- `Horario`

Donde `Clase` relaciona la clase `profesor` de la ontología persona, con la clase `ProgramaAcadémico` de la ontología académico CBI y con la clase `Salon` de la ontología espacio físico, a su vez con la clase `Horario` de la ontología red.

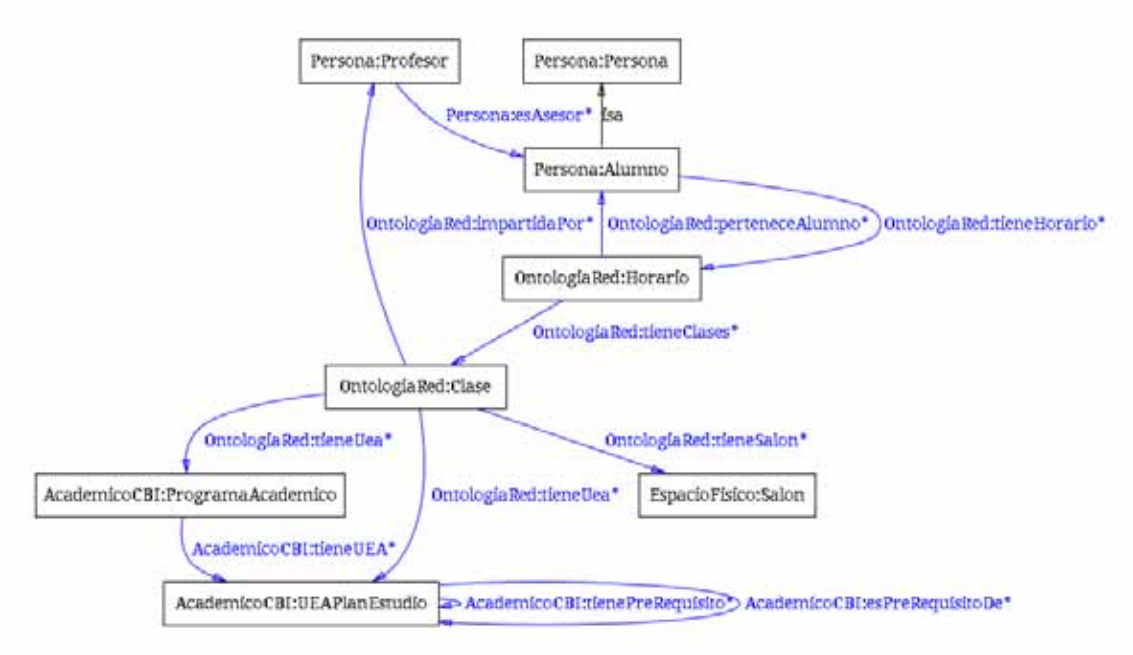


Figura 14: Diseño de la ontología red

Las propiedades de las clases de esta ontología son:

Para la clase `Clase` las propiedades de hora de inicio y hora fin, tomando en cuenta las instancias de las demás clases para relacionarse con la clase `Profesor` se toma la propiedad `impartidaPor` que es una propiedad de objeto, la propiedad `tieneUea` es usado para la relación con la clase `ProgramaAcademico`, para la relación con la clase `Salon` de la ontología espacio físico se usa la propiedad `tieneSalon`.

También se tiene la clase `Horario` esta clase relaciona un `Alumno` con una clase, con la propiedad `perteneceAlumno` y la propiedad `tieneClase`, en la Figura 14 se muestran estas relaciones.

2.4.5. Ontología trámites

Esta ontología contiene los trámites que un alumno puede realizar como por ejemplo la reposición de la credencial, el seguro facultativo, constancias, practicas profesionales, etc.

Las clases que lo conforman son:

- `AlumnoLicenciatura`
  - `Cambios`
  - `Constancias`
  - `Credencial`
  - `Movilidad`
  - `RevisionDeCalificacion`
  - `SeguroFacultativo`
  - `ServicioSocial`
  - `ProyectoTerminal`
- `AlumnoPosgrado`
  - `Constancias`
  - `Credencial`
  - `SeguroFacultativo`
- `EgresadoLicenciatura`
- `EgresadoPosgrado`

Las propiedades de la clase Tramites que engloba a las anteriores tiene las propiedades de sus datos son:

- `TieneDocumentos`
- `TieneGuia`
- `TieneNombre`
- `TieneRequisitos`

Las propiedades del objeto para la relación con otros objetos de las ontologías persona y espacio físico son las siguientes:

- `TieneResponsable`
- `TieneUbicacion`

El diseño de la ontología está descrito en la Figura 15.

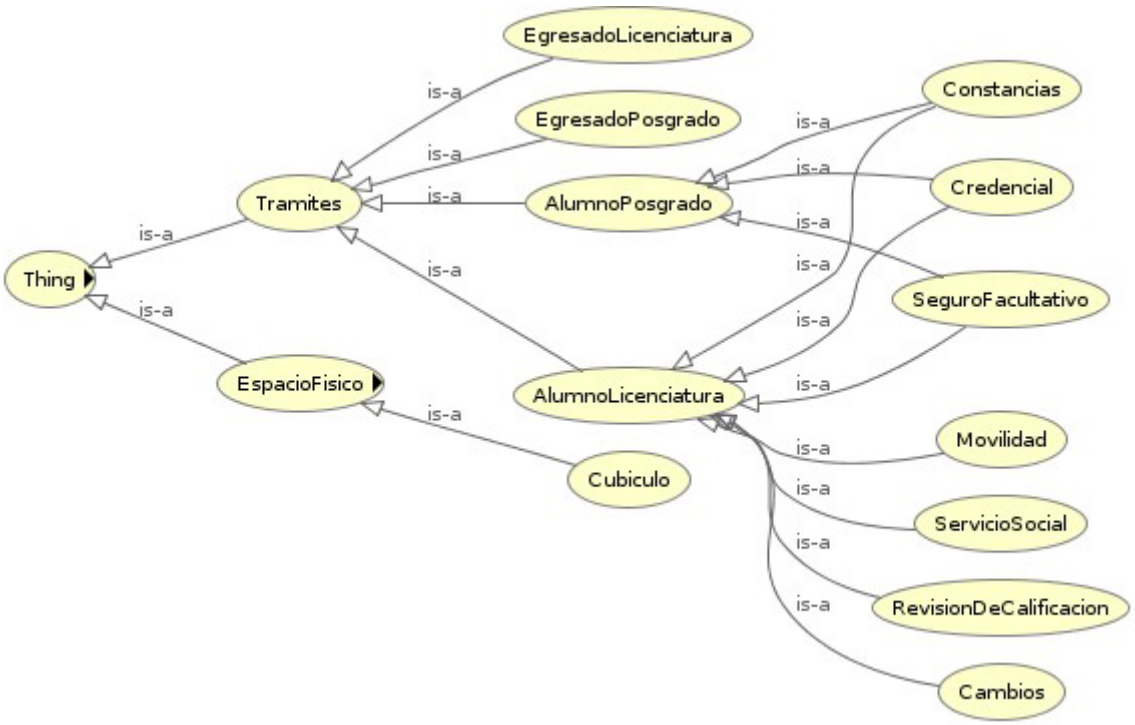


Figura 15: Diseño de la ontología trámites

3. Implementación

Después de haber realizado el análisis y diseño de un buscador semántico (especificar el objetivo del buscador!!!) para la UAM Azcapotzalco, se dispuso a codificar dicha solución, se muestra la viabilidad de la arquitectura propuesta, para ello, se describe la implementación de cada uno de los componentes del software tanto en el cliente móvil con los que se encuentran en el lado del servidor, se usaron herramientas tanto de código abierto y de distribución gratuita o con licencia GNU GPL (*General Public License)* para la implementación de la arquitectura.

Para la codificación de cada uno de los componentes que conforman el proyecto se hizo uso del lenguaje de programación Java.

En la base de conocimiento se requirió el uso de cuatro ontologías con un dominio particular, en la quita ontología se engloba el uso de las cuatro anteriores y crea las relaciones entre los conceptos de cada una de ellas.

Para la consulta y del conocimiento almacenado en estas ontologías se utilizó un motor de inferencia con capacidad de ejecutar reglas en el lenguaje SWRL y consultas SQWRL (*Semantic Query-Enhanced Web Rule Language*).

Para desarrollar la aplicación móvil en la capa del cliente se utilizó el lenguaje java, utilizando un entorno de desarrollo integrado, llamado ADT (*Android Developer Tools*), esta herramienta es gratuita y tiene integrado los siguientes componentes:

- *Eclipse + ADT plugin*.
- *Android SDK Tools*.

En la capa del consumidor del servicio web y en la capa de conexión del móvil para su implementación se usó una liberara llamada "*ksoap2-android*", escrita en java para poder realizar la peticiones con el protocolo SOAP.

Para la edición de las ontologías, se usó el editor de llamado Protégé en su versión 4.2, se utilizó en los módulos de la capa ontológica y de la capa de población de la ontología .

las interfaces de programación de aplicaciones o API (*Application Programming Interface*) : OWL-API y API Protégé-OWL.

3.1. Tecnologías de implementación

3.1.1. Lenguaje OWL

Para la representación de la base de conocimiento se utilizaron ontologías para la Web.

Las ontologías son utilizadas para almacenar conocimiento acerca de un dominio de interés en específico, describiendo los conceptos de dicho dominio y las relaciones existentes entre estos conceptos. Para la escritura de las ontologías existen diferentes lenguajes sin embargo el más reciente estándar es el lenguajes OWL (*Ontology Web Language*)[9] definido por la W3C.

Para el desarrollo de las antologías se empleó el sublenguaje OWL-DL ya que es más expresivo que OWL-Lite. El sublenguaje OWL-DL está basado en la lógica descriptiva; por lo tanto, forma parte de la lógica de primer orden y puede ser sujeto al razonamiento automático.

3.1.2. Editor de ontologías Protégé

Es una herramienta de código abierto que ayuda a los usuarios en la construcción de grandes bases de conocimientos y que permite a los desarrolladores crear y editar ontologías para la construcción de modelos de dominio basados en conocimiento, esta herramienta fue creada por Mark Musen[10].

Protégé es uno de los muchos editores de ontologías OWL y es de código abierto. Brinda una interfaz de usuario que facilita la edición de ontologías también cuenta con una API para poder ser usado dentro de códigos java.

3.1.3. API Protégé-OWL

El API Protégé-OWL[11] es una biblioteca de código abierto de Java para el *Web Ontology Language* y RDF (S) (*Resource Description Framework*)[12]. La API proporciona clases y métodos para cargar y guardar archivos OWL, para consultar y manipular los modelos de datos OWL, también para llevar a cabo el razonamiento.

3.1.4. SWRL y SQWRL

SWRL (*Semantic Web Rule Language*) es un lenguaje para la definición de reglas sobre una base de conocimiento como son las ontologías OWL. Una ontología OWL contiene una secuencia de hechos y axiomas. SWRL propone una extensión de OWL para proponiendo reglas a los axiomas ya establecidos dentro de una ontología.

SWRL propone una extensión de OWL proponiendo reglas a los axiomas ya establecidos dentro de una ontología. Concretamente, propone reglas de implicación de la forma antecedente y consecuencia, con una sintaxis:

Antecedente -> consecuente.

SQWRL (*Semantic Query-enhanced Web Rule Language*)[13] está basado en SWRL. SQWRL toma a SWRL y lo trata como un patrón de especificación para consultas. Remplaza el consecuente de la regla, con una especificación de recuperación, además define un conjunto de operadores para la construcción de especificaciones de recuperación.

3.1.5. Razonador JESS

Jess es un motor de reglas y entorno de programación escrito enteramente en lenguaje Java. Con este motor, se puede construir aplicaciones Java que tiene la capacidad de razonar usando conocimiento en forma de reglas declarativas. Jess es pequeño, ligero, y uno de los motores de reglas más rápidos actualmente disponibles[14].

3.1.6. kSOAP 2

kSOAP[15] es una biblioteca de cliente de servicios web SOAP para entornos Java limitados como *applets* o aplicaciones J2ME.

3.1.7. Rest

REST[16] es un estilo de arquitectura para el diseño de aplicaciones en red. La idea es que, en lugar de usar mecanismos complejos, tales como SOAP para la conexión entre máquinas, se use HTTP simple para realizar llamadas entre las máquinas.

REST es una alternativa ligera a mecanismos como RPC[1] (*Remote Procedure Calls*) y servicios web (SOAP, WSDL , et al.).

3.1.8. JSON

JSON (*JavaScript Object Notation*)[17] es un formato de intercambio de datos ligero. Se basa en un subconjunto del lenguaje de programación JavaScript. JSON es un formato de texto que es completamente independiente del lenguaje pero utiliza convenciones que son familiares para los programadores. Estas propiedades hacen de JSON un lenguaje ideal de intercambio de datos.

3.1.9. Apache Tomcat

Apache Tomcat[18] es una implementación de software de código abierto que funciona con un contenedor de aplicaciones de *Java Servlet*[2] y tecnologías *JavaServer Pages*[3], se usa en la capa del servidor proporcionando soporte para el servicio web.

3.1.10. Apache Axis2/Java

Apache Axis2/Java[19] es un motor para servicios web, es usado para el despliegue de servicios web dentro de *Apache Tomcat*.

1 RPC, Remote Procedure Call, una técnica para la comunicación entre procesos en una o más computadoras conectadas a una red.
2 Se refiere a pequeños programas que se ejecutan en el contexto de un navegador web.
3 Tecnología que ayuda a crear páginas web dinámicas basadas en HTML, XML entre otros tipos de documentos.

3.1.11. Android Developer Tools

El ADT (*Android Developer Tools*) para eclipse proporciona un entorno de desarrollo de nivel profesional para la creación de aplicaciones para *Android*[1]. Es una IDE de Java completo con funciones avanzadas para ayudar a construir, probar, depurar y empaquetar aplicaciones para Android.

3.2. Capa cliente móvil

Dentro de esta capa destaca la aplicación móvil, se describirá cada una de las capas que conforman a la aplicación del dispositivo móvil.

3.2.1. Capa de usuario

Al iniciar la aplicación se encuentra la capa del usuario que proporciona la navegación ente las actividades que puede realizar dentro de la aplicación, como son realizar las consulta, ver los mapas y revisar el historial de búsqueda, en la Figura 16 se detallan las clases que intervienen para tal fin.

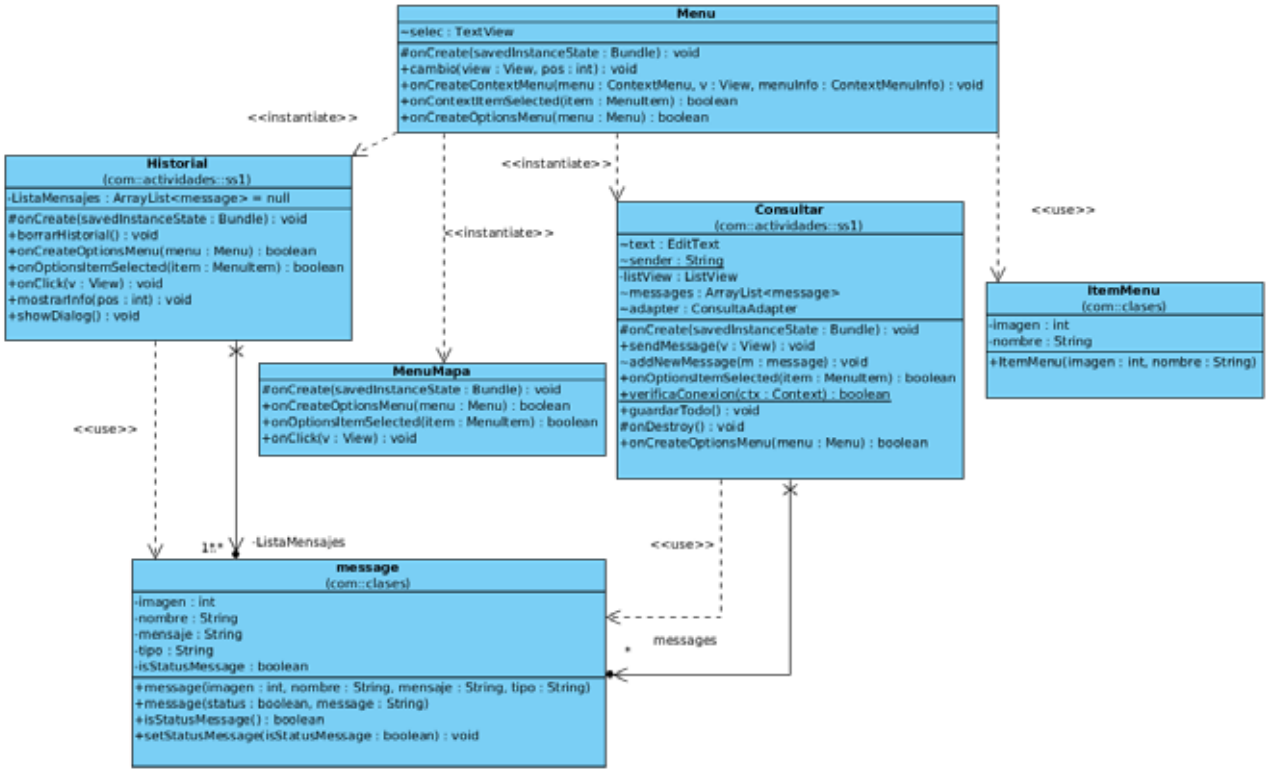


Figura 16: Diagrama de clases de la capa del usuario

Cuando el usuario ingresa a la aplicación se encuentra con una pantalla principal que contiene tres opciones: consultar, historial y mapa; esta pantalla de diseño de tal manera que las opciones sean de fácil acceso e intuitivas. En la Figura 17 se muestra la pantalla principal de la aplicación.

1 Android, es un sistema operativo basado en Linux creado especialmente para dispositivos móviles.

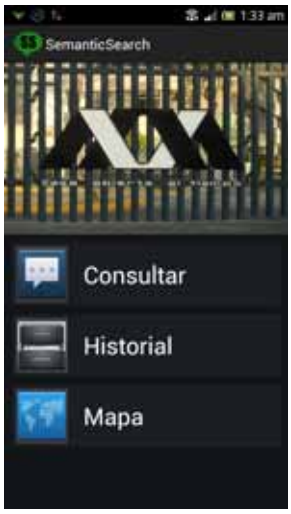


Figura 17: Pantalla inicial de la aplicación móvil

3.2.2. Capa de consulta móvil

La capa de consulta dentro del dispositivo está compuesto por las clases `EnvioMensajeWebService` encargado de instanciar un objeto de la clase `ConsumidorWebService` que se encargará de hacer la petición de consulta al servicio web, hace uso del objeto `message` donde este objeto contendrá la cadena con la consulta, también instancia el objeto `AyudaListActivity` este objeto es usado para las ayudas de como realizar una consulta en el sistema, ver Figura 18.

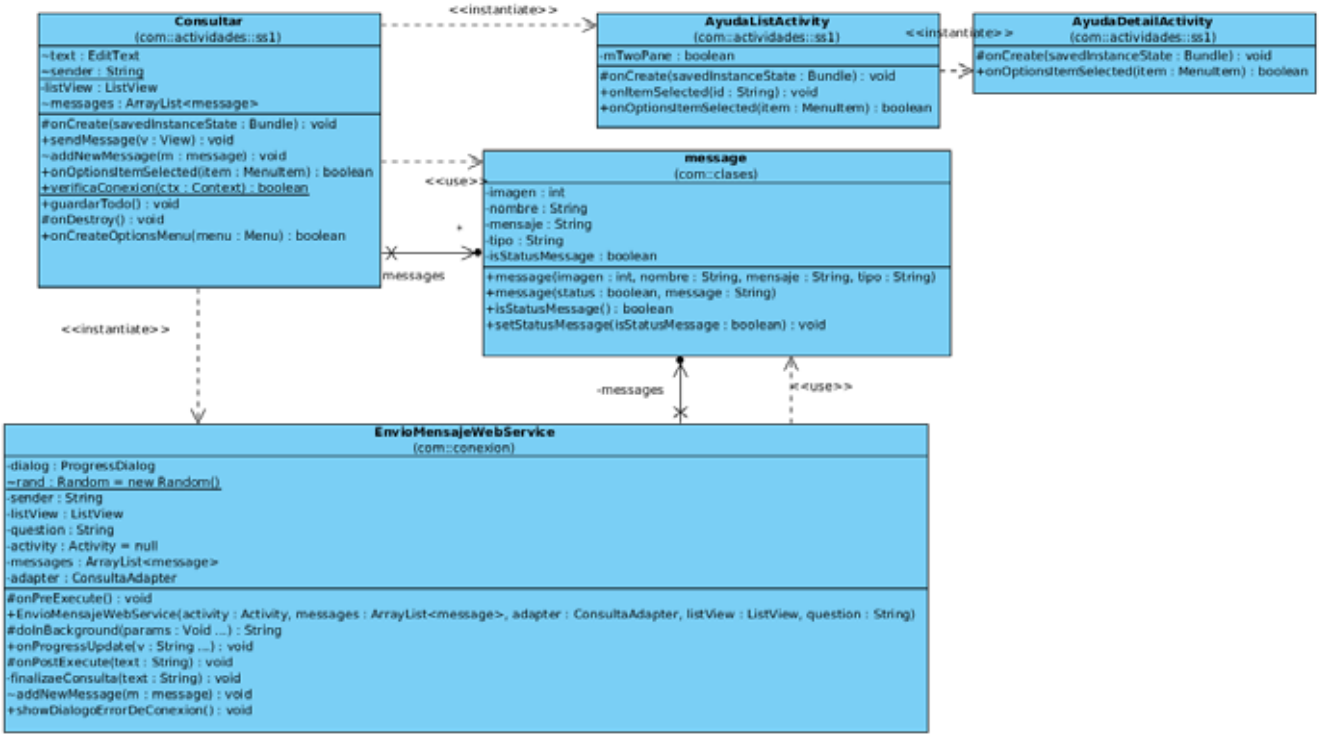


Figura 18: Diagrama de clases de la capa de consulta

Para la realización de la consulta en esta capa es necesario que la clase `EnvioMensajeWebService` sea declarada como una tarea asíncrona, ver Código 1.

```
1  package com.conexion;
2
3  import java.util.ArrayList;
16
17 public class EnvioMensajeWebService extends AsyncTask<Void, String, String> {
18
19     private ProgressDialog dialog;
20     private ArrayList<message> messages;
21     private ConsultaAdapter adapter;
22     static Random rand = new Random();
```

Código 1: Clase EnvioMensajeWebService declarada como extensión de una tarea asíncrona

```
89   * servicio web
90   */
91  public void sendMessage(View v) {
92      String newMessage = text.getText().toString().trim();
93      if (newMessage.length() > 0) {
94          text.setText("");
95          addNewMessage(new message(R.drawable.question, "user", newMessage,"send"));
96          new EnvioMensajeWebService(this, messages, adapter, listView,newMessage).execute();
97
98      }
99  }
```

Código 2: Método sendMessage de la clase Consultar

El método `sendMessage` está encargado de crear una instancia de tipo `EnvioMensajeWebService`, ver Código 2.

Dentro de la clase `EnvioMensajeWebService` se sobre escribe el método `doInBackground` para que realicé la creación de una instancia de la clase `ComsumidorWebService` y regrese la cadena con la respuesta obtenida por el servicio web, ver Código 3.

```
48      @Override
49      protected String doInBackground(Void... params) {
50
51          // try {
52          // Thread.sleep(500); // simulate a network call
53          // } catch (InterruptedException e) {
54          // e.printStackTrace();
55          // }
56
57          this.publishProgress(String
58                  .format("%s ,realizando la consulta", sender));
59          // try {
60          // Thread.sleep(2000); // simulate a network call
61          // } catch (InterruptedException e) {
62          // e.printStackTrace();
63          // }
64
65          return new ConsumidorWebService().consultarWs(this.question);
66      }
67
68      @Override
```

Código 3: Método doInBackgroud() de la clase EnvioMensajeSwebService

Figura 19: Pantalla de consulta

En la Figura 19 se muestra la pantalla principal de consulta, esta pantalla está compuesta por un cuadro de texto donde el usuario escribe su consulta y un botón para realizar el envío al servicio web.

3.2.3. Capa del consumidor del servicio web y la capa de conexión móvil

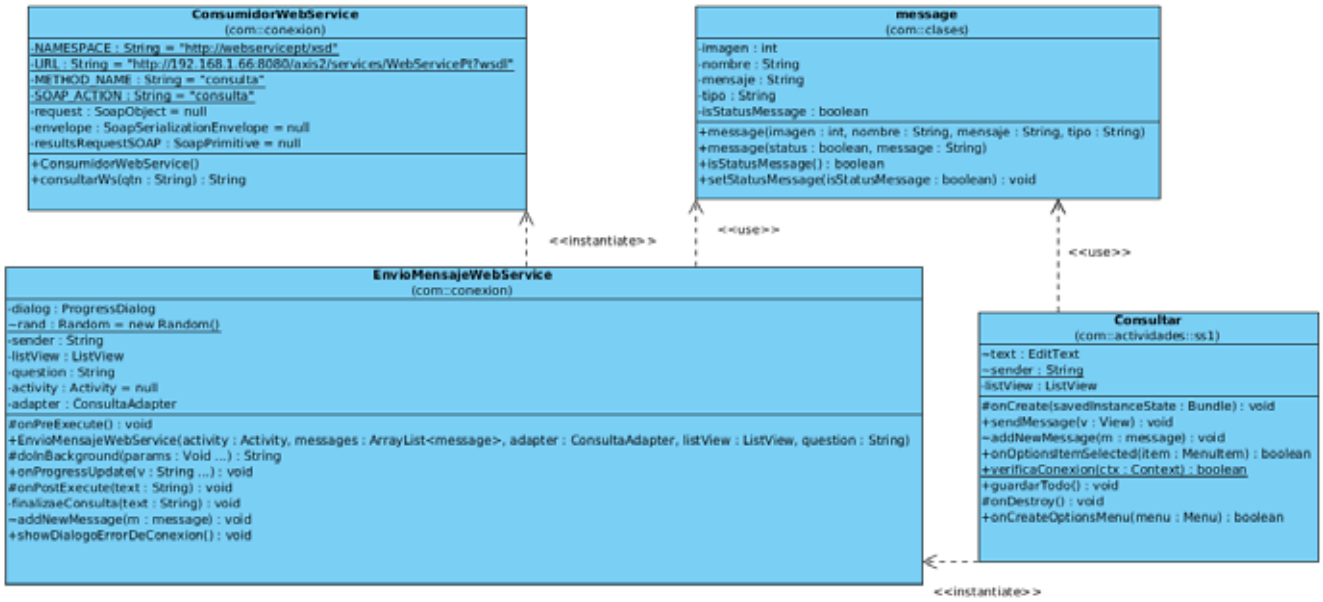


Figura 20: Diagrama de clases capa del consumidor del servicio web

Dentro de la clase `ConsumidosWebService` se tienen las variables:

```
NAMESPACE = "http://webservicept/xsd";
```

```
URL = "http://192.168.1.66:8080/axis2/services/WebServicePt?wsdl";
METHOD_NAME = "consulta";
SOAP_ACTION = "consulta";
```

Estas variables son las que se necesitan para poder realizar la llamada al servicio web, la URL especificada es de uso local pero puede ser sustituida con una URL de Internet, además de estas variables se necesita los objetos SOAP: `SoapObject`, `SoapSerializationEnvelope`, `SoapPrimitive`, ver Código 4.

```
14 public class ConsumidorWebService {
15
16     // Constantes para la invocación del web service
17     // Namespace definido en el servicio web como  targetNamespace="http://webservicept/xsd"
18     private static final String NAMESPACE = "http://webservicept/xsd";
19     // URL de definicion del servcio web
20     private static String URL = "http://192.168.1.66:8080/axis2/services/WebServicePt?wsdl";
21     // Metodo que queremos ejecutar en el servicio web
22     private static final String METHOD_NAME = "consulta";
23     // namespace + metodo
24     private static final String SOAP_ACTION = "consulta";
25
26     // Declaracion de variables para consumir el web service
27     private SoapObject request = null;
28     private SoapSerializationEnvelope envelope = null;
29     private SoapPrimitive resultsRequestSOAP = null;
30
```

Código 4: Variables y objetos necesarios para el consumo del servicio web

El proceso para el consumo del servicio web se detalla en el Código 5.

```
33     public String consultarWs(String qtn) {
34
35         request = new SoapObject(NAMESPACE, METHOD_NAME);
36         Log.i("consumidor WebService", "se creo el request");
37         request.addProperty("q", qtn);
38         envelope = new SoapSerializationEnvelope(SoapEnvelope.VER11);
39         Log.i("consumidor WebService", "se creo en envelope");
40         envelope.dotNet = false;
41         envelope.setOutputSoapObject(request);
42         Log.i("consumidor WebService","se seteo el OutPut Soap Object con el request");
43
44         HttpTransportSE transporte = new HttpTransportSE(URL,30000);
45         Log.i("consumidor WebService", "Se creo el transporte HTTP");
46
47         try {
48             transporte.call(SOAP_ACTION, envelope);
49             Log.i("consumidor WebService", "Transporte.call exitoso");
50
51             resultsRequestSOAP = (SoapPrimitive) envelope.getResponse();
52             Log.i("consumidor WebService", "RquestSOAP exitoso");
53
54             return resultsRequestSOAP.toString();
55
56         } catch (IOException e) {
57             Log.w("try del transporte Consumidor Web service",
58                     "Ocurrio un error en el transporte.call");
59
60             e.printStackTrace();
61         } catch (XmlPullParserException e) {
62             e.printStackTrace();
63         }
64
65         return null;
66
67     }
68 }
```

Código 5: Método consultarWS() de la clase ConsumidorWebService

Cuando se realiza esta tarea asíncrona se muestra al usuario un diálogo que le indica que se está realizando la consulta, en la Figura 21 se muestra la secuencia de consulta.

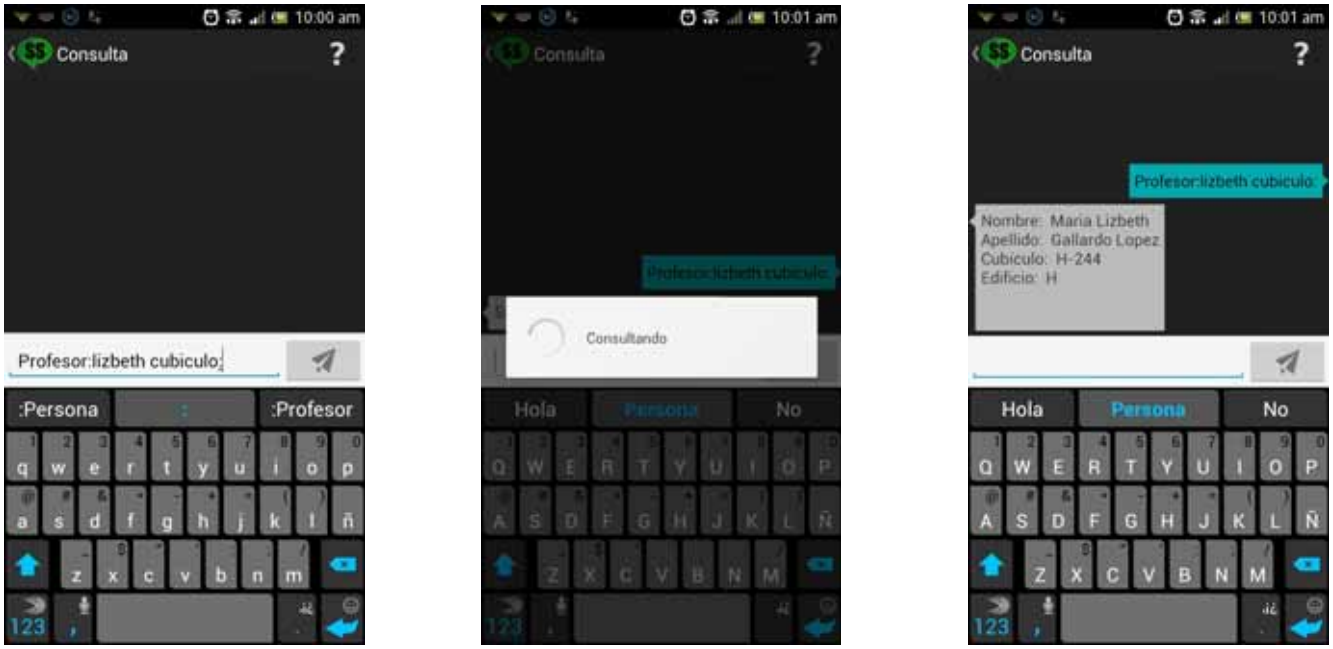


Figura 21: Secuencia para la realización de una consulta

3.3. Capa Servidor

En la capa del servidor web se encuentran tanto el servicio web como las antologías que son usadas para la base de conocimiento.

3.3.1. Capa del servicio web

El servicio web hace uso de la clase `WebServicePt` que crea instancias de la clase `ObtenerSentencia`, donde se procesa la consulta de las palabras clave que proceden del cliente móvil, convirtiendo una sentencia como `profesor:maricela cubiculo:` a su consulta correspondiente en el lenguaje SQWRL:

```
Persona:Profesor(?x) ^ Tramites:tieneCubiculo(?x, ?y) ^
Tramites:TieneUbicacion(?y, ?g) ^ EspacioFisico:TieneLetra(?g, ?
Edificio) ^ Tramites:tieneNombreCubiculo(?y, ?Cubículo) ^
Persona:tieneNombre(?x, ?Nombre) ^  Persona:tieneApellido(?x, ?
Apellido) ^ swrlb:containsIgnoreCase(?Nombre, ?nombre_profesor) ->
sqwrl:select(?Nombre, ?Apellido, ?Cubículo, ?Edificio).
```

Donde `?nobre_profesor` es sustituido por el parámetro que ingresó el usuario en este ejemplo se sustituye por `maricela`.

Para convertir las peticiones del usuario a las consulta SQWRL se hizo uso de expresiones regulares[1], las clases que se implementaron en la capa del servicio web se muestran en la Figura 22.

1 secuencia de caracteres que forma un patrón de búsqueda, principalmente utilizada para la búsqueda de patrones de cadenas de caracteres u operaciones de sustituciones

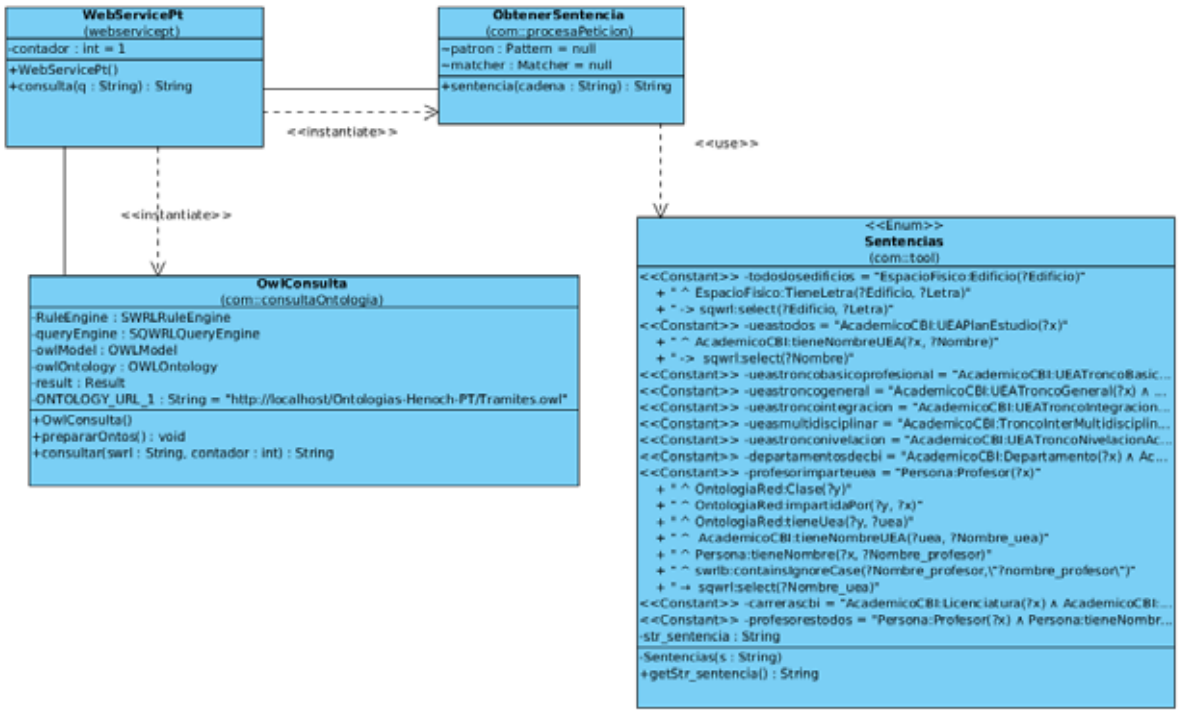


Figura 22: Diagrama de clases de la capa del servicio web

La clase consulta es la encargada de cargar la ontología en memoria y realizar la consulta con el lenguaje SQWRL.

El método del servicio web que se usada para realizar la consulta se de talla en el Código 6.

```
public String consulta(String q) {

    contador++;
    String nomregla = obtenerSentencia.sentencia(q);
    String respuesta = "";
    if (nomregla.startsWith("* Use palabras")) {
       respuesta = nomregla;
    } else {
        try {
            respuesta = Owlconsulta.consultar(nomregla, contador);
        } catch (Exception ex) {
            System.out.println(ex.getMessage());
            respuesta = "* Use palabras clave generales\n"
                + "* Use los filtros con sentido común\n"
                + "* Si su búsqueda no produce resultados, "
                + "intente cambiar algunos parámetros\n"
                + "* =( Ocurrio un error al procesar su solicitud, intentelo de nuevo";
            if(!Owlconsulta.errorGral.isEmpty()){
               respuesta+="\n"+Owlconsulta.errorGral.toString();
            }
        }
    }
    return respuesta;
}
```

Código 6: Método consulta de la clase WebServicePt

En el Código 7 se muestra un fragmento de la clase encargada de pasar de la sentencia que el usuario envió al servicio web a su consulta en el lenguaje SQWRL.

```
patron = Pattern.compile("^(\\s*profesor:(.+)esasesorde:\\s*$)");
matcher = patron.matcher(cadena);
if (matcher.find()) {
    if (matcher.group(2).isEmpty()) {
    } else {
        sentenciasObtenida = Sentencias.prodesoresasesorde.getStr_sentencia().replace("?nombre_profesor", m
    }
}
patron = Pattern.compile("^(\\s*alumno:\\s+conpt:\\s*$)");
matcher = patron.matcher(cadena);
if (matcher.find()) {
    sentenciasObtenida = Sentencias.alumnosconpt.getStr_sentencia();
}
patron = Pattern.compile("^(\\s*alumno:\\s+mayorde:(\\d+)\\s*$)");
matcher = patron.matcher(cadena);
if (matcher.find()) {
    sentenciasObtenida = Sentencias.alumnosmayorque.getStr_sentencia().replace("?anios", matcher.group(2));
}
patron = Pattern.compile("^(\\s*alumno:\\s+mayorde:(\\d+)\\s*cantidad:\\s*$)");
matcher = patron.matcher(cadena);
if (matcher.find()) {

    sentenciasObtenida = Sentencias.alumnosmayorquecantidad.getStr_sentencia().replace("?anios", matcher.gr
}
```

Código 7: Fragmento de la clase ObtenerSentencia

La clase Sentencias es un tipo `Enum`[1], en el Código 8 se muestra una parte de la clase, donde se definen todas las sentencias SQWRL que se requieran en la consulta. Cada sentencia tiene una clave marcada en color verde, la cual es usada en la clase `ObtenerSentencia`.

```
12  public enum Sentencias {
13
14      prodesoresasesorde("Persona:Profesor(?x)"
15      + " ^ Persona:esAsesor(?x, ?y)"
16      + " ^ Persona:tieneNombre(?y, ?Nombre)"
17      + " ^ Persona:tieneApellido(?y, ?Apellido)"
18      + " ^ Persona:tieneNombre(?x, ?Nombre_profesor)"
19      + " ^ swrlb:containsIgnoreCase(?Nombre_profesor,\"?nombre_profesor\")"
20      + " -> sqwrl:select(?Nombre, ?Apellido)"),
21      alumnosconpt("Persona:Licenciatura(?x)"
22      + " ^ Persona:tieneAsesorPT(?x, ?y)"
23      + " ^ Persona:tieneApellido(?x, ?Apellido)"
24      + " ^ Persona:tieneNombre(?x, ?Nombre)"
25      + " -> sqwrl:select(?Nombre, ?Apellido)"),
26      alumnosmayorque("Persona:Licenciatura(?x)"
27      + " ^ Persona:tieneEdad(?x, ?y)"
28      + " ^ swrlb:greaterThan(?y, ?anios)"
29      + " ^ Persona:tieneApellido(?x, ?Apellido)"
30      + " ^ Persona:tieneNombre(?x, ?Nombre)"
31      + " -> sqwrl:select(?Nombre, ?Apellido)"),
32      alumnosnemorque("Persona:Licenciatura(?x)"
33      + " ^ Persona:tieneEdad(?x, ?y)"
34      + " ^ swrlb:lessThan(?y,  ?anios)"
```

Código 8: Clase Enum Sentencias

1 Un tipo de enumeración es un tipo de datos especial que permite, por una variable a ser un conjunto de constantes predefinidas. La variable debe ser igual a uno de los valores que han sido predefinidas para ello.

```
110
111         creator.setOntologyUri(ONTOLOGY_URL_1);
112         try {
113             creator.create(errors);
114         } catch (OntologyLoadException ex) {
115             Logger.getLogger(OwlConsulta.class.getName()).log(Level.SEVERE, null, ex);
116              errorGral.add(ex.getMessage());
117         }
118
119         owlModel = creator.getOwlModel();
120
121         if (!errors.isEmpty()) {
122             System.err.println("Error loading imports.");
123
124         }
125         try {
126             RuleEngine = P3SWRLRuleEngineFactory.create("Jess", owlModel);
127         } catch (SWRLRuleEngineException ex) {
128             Logger.getLogger(OwlConsulta.class.getName()).log(Level.SEVERE, null, ex);
129              errorGral.add(ex.getMessage());
130         }
131         try {
132             queryEngine = P3SQWRLQueryEngineFactory.create(owlModel);
133         } catch (ResultException ex) {
134             Logger.getLogger(OwlConsulta.class.getName()).log(Level.SEVERE, null, ex);
135              errorGral.add(ex.getMessage());
136         }
137
138     }
```

Código 9: Creación del modelo de la ontología en memoria, motor de reglas y de consulta

Para la consulta se carga el modelo de la ontología en memoria y se crea una instancia del motor de reglas Jess, se crea también el motor de consulta al cual se le manda como parámetro el modelo de la ontología ya con el motor de reglas Jess definido con anterioridad, ver el Código 9.

3.3.2. Capa ontológica

Dentro de la capa ontológica se encuentran las antologías con sus relaciones, en la implementación se tomó el modelo multidimensional ver Figura 23, esto es que una ontología tiene relación con otra, estas relaciones son dadas de los individuos de una a los individuos de otra.

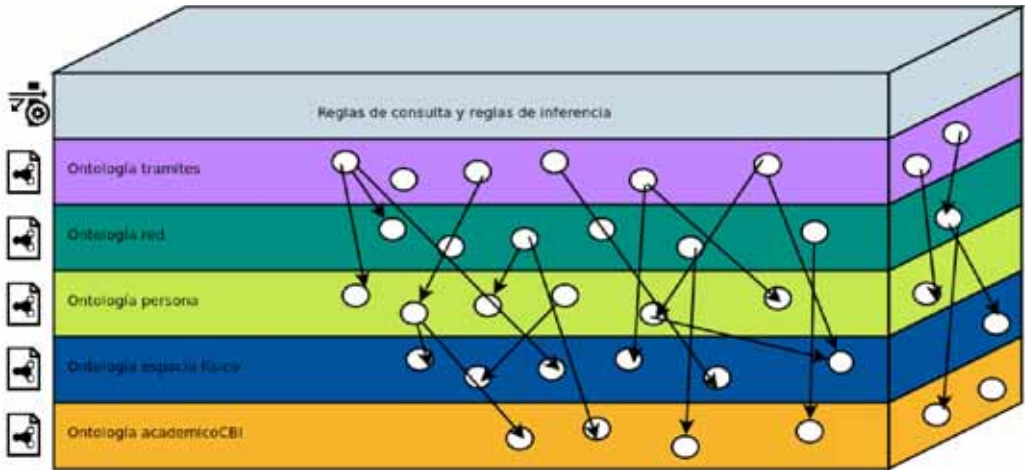


Figura 23: Modelo multidimensional de las relaciones de cada ontología

3.3.3. Capa de poblado de la ontología

Para poblar la ontología se utilizó la herramienta de Protégé, se recolectó información relacionada a la División de Ciencias Básicas e Ingeniería de la UAM unidad Azcapotzalco para el llenado de las ontologías, como ejemplo en la Figura 24 se muestra un individuo de la clase Tramites, este es el trámite para obtener el seguro facultativo.

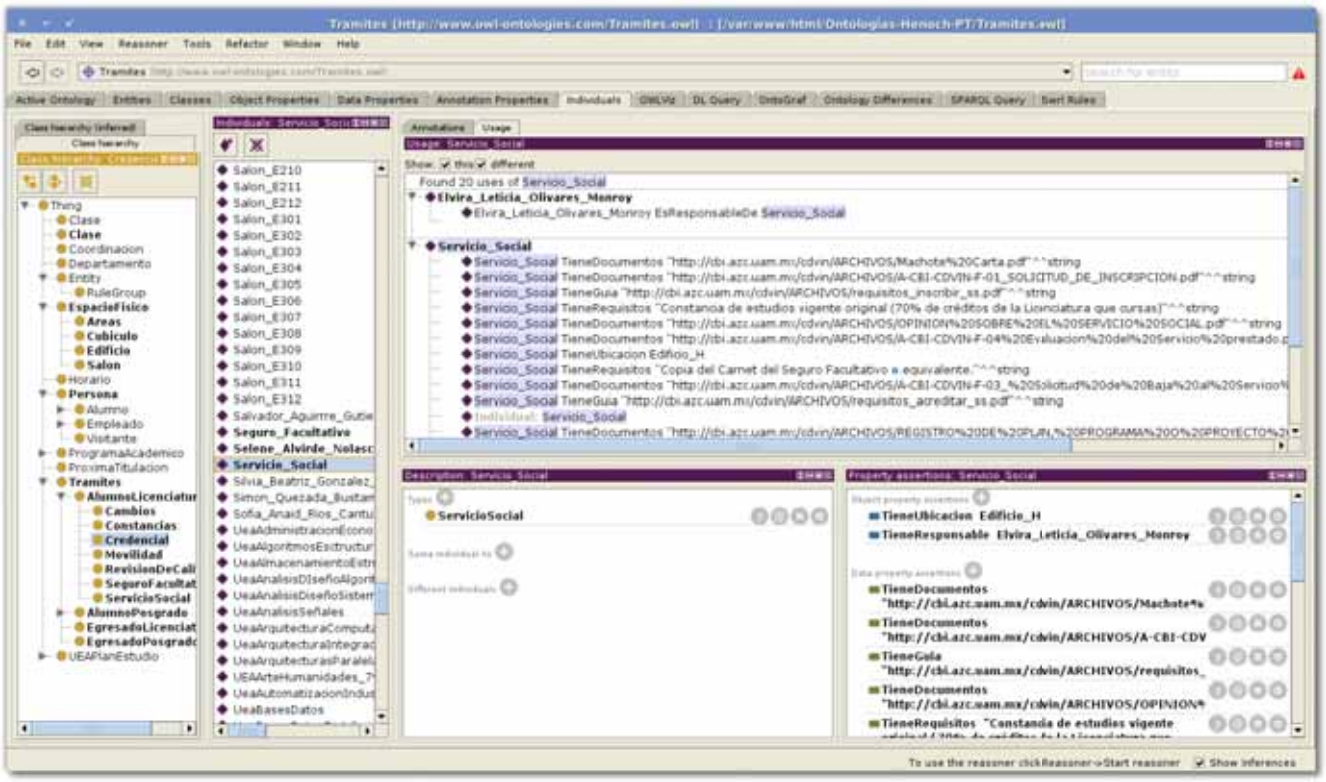


Figura 24: Poblado de la ontología trámites con Protégé

4. Pruebas y resultados

Para realizar las pruebas se contó con lo siguientes recursos:

Computadora portátil marca Dell ® con las siguientes características:

- Procesador intel® Core™ i7.
- 6 GB de memoria RAM.
- 500 GB de disco duro.
- Sistema operativo Ubuntu 12.04.

Servidor marca Dell® con las siguientes características:

- Procesador intel® Core™ 2 Duo.
- 4 GB de memoria RAM.

- 80 GB de disco duro.
- Sistema operativo CentOS 5.

Celular marca Sony® Xperia™ S con las siguientes características:

- Procesador dual core Qualcomm® a 1.5 Ghz.
- 1 GB de memoria RAM.
- 32 GB de memoria interna.
- Sistema operativo Google Android 4 (*Ice Cream Sandswich*).
- Pantalla táctil TFT de 4.3 pulgadas.

Para la verificación del funcionamiento de cada uno de los componentes, primero se realizaron pruebas al servicio web, posteriormente las pruebas en la aplicación móvil.

4.1. Pruebas al servicio web

Se realizaron dos tipos de prueba sobre el servicio web: la primera se realizó haciendo preguntas directamente con una petición Rest, la segunda fue con la aplicación móvil.

En la Figura 25 se muestras el archivo WSDL[20] del servicio web. La respuesta dada por el servicio web después de realizar una petición Rest con la consulta `edificios:any` se muestra en la Figura 26.

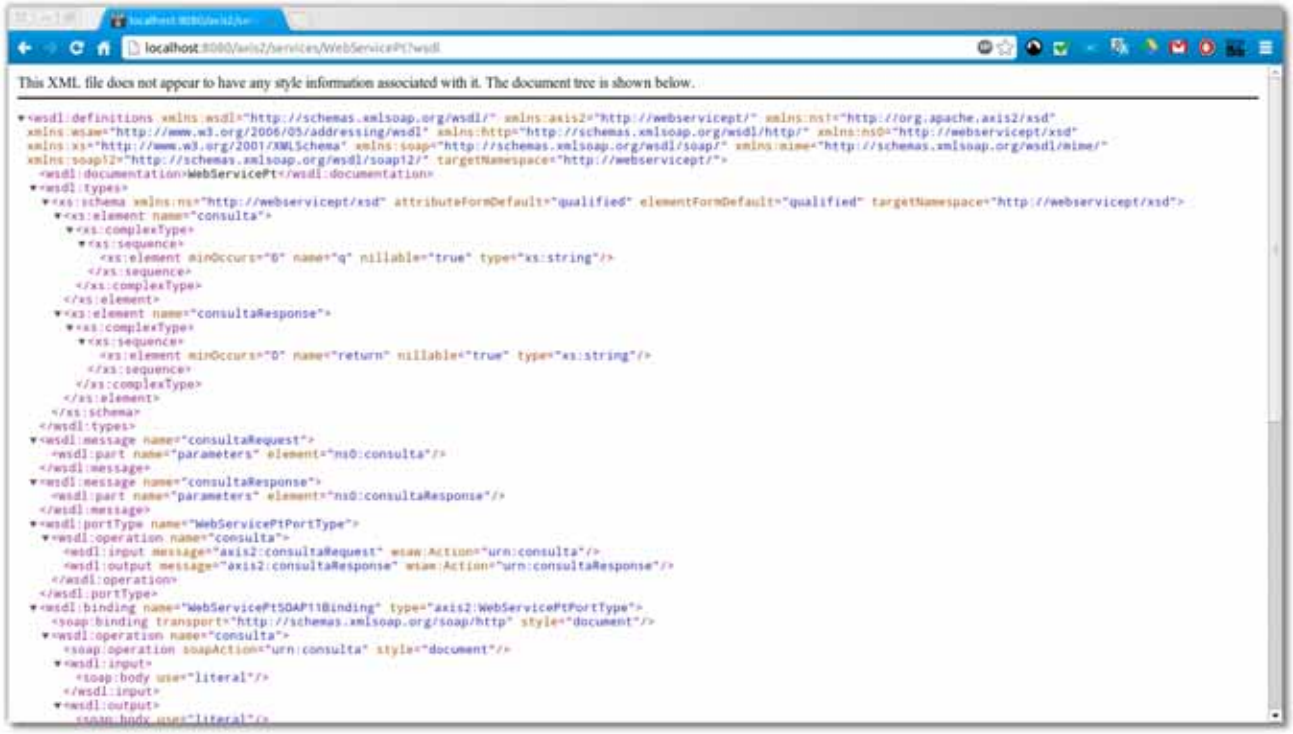


Figura 25: Archivo WSDL del servicio web mostrado en el navegador

Figura 26: Resultado de la consulta "edificios:any:"

La petición se formuló con la dirección del WSDL "http://localhost:8080/axis2/services/WebServicePt" seguido del nombre de la función y el parámetro que recibe en forma de palabras clave: "consulta?q=edificios:any" quedando de la siguiente forma.

`http://localhost:8080/axis2/services/WebServicePt/consulta?q=edificios:any`

La Figura 27 muestra las respuesta a la consulta "tramites:servicio social encargado:" nos muestra el nombre y los datos de la persona encargada del trámite.

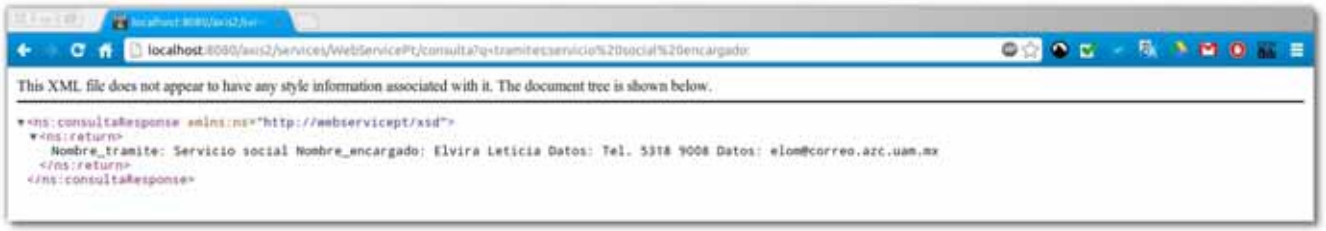


Figura 27: Resultado de la consulta "tramites:servicio social encargado:"

Consulta `salon:e305 horario:` el resultado se muestra en la Figura 28.

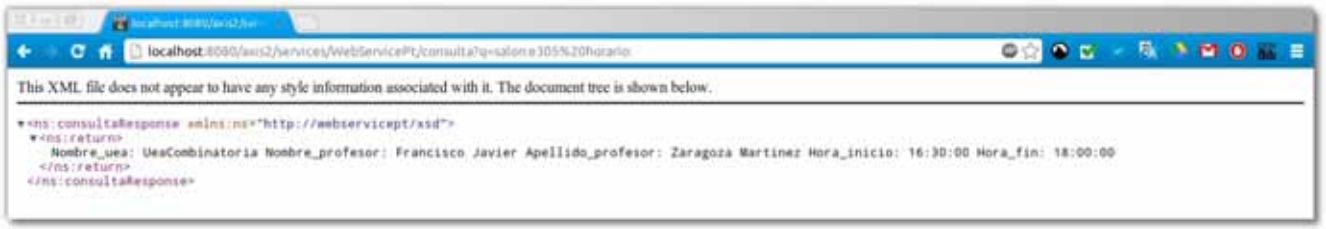


Figura 28: Resultado de la consulta "salon:e305 horario:"

4.2. Pruebas en la aplicación móvil

Dentro de la aplicación se hicieron las consulta siguientes:

`edificios:any, profesor:Maricela cubiculo:, tramites:credencial guia:, salon:e310 horario:, profesor:Lizbeth datos:, tramites:any, profesor:Francisco imparteueas:, uea:troncogeneral, uea:UeaBasesDatos creditos:, carrerascbi:any`, ver Figura 29.

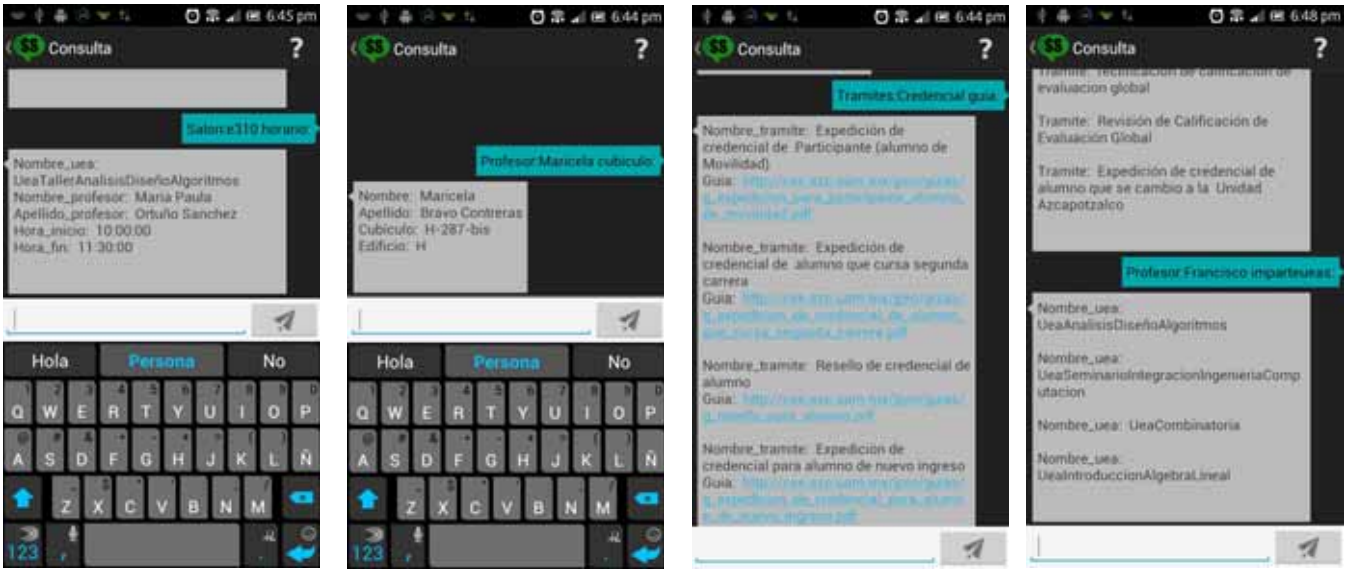


Figura 30: Error de algún operador mal escrito

Figura 31: Error de conexión al servicio web

Figura 32: Error al procesar el SQWRL en el servicio y muestra el origen de la falla

Cuando se produce algún error se muestran los mensajes siguientes: Error de conexión al servicio web Figura 31, error en algún operador mal escrito Figura 30, error que ocurre dentro del servicio web al procesar la consulta SQWRL Figura 32.

5. Conclusiones

El objetivo de este proyecto terminal fue la creación de una herramienta de búsqueda semántica de recursos académicos de la División de Ciencias Básicas e Ingeniería de la UAM unidad Azcapotzalco. El sistema de búsqueda se diseñó e implementó para una plataforma de cómputo móvil, concretamente para celulares con sistema operativo Android. Este buscador es una herramienta tanto para los alumnos como para los docentes para que puedan realizar consultas respecto a los recursos de la División de Ciencias Básicas e Ingeniería, tales como trámites para alumnos, ubicación de cubículos de profesores, las UEA que imparte algún profesor de la División, entre otros.

El buscador semántico se propone como un servicio web, donde por el momento el cliente es un dispositivo móvil. El buscador semántico está formado por una base de conocimientos académicos de la División de Ciencias Básicas e Ingeniería, un conjunto de reglas SWRL de inferencia para obtener el conocimiento, un conjunto de reglas SQWRL para las consultas, el motor de inferencia Jess y un motor de consultas de protégé API. La aplicación cliente para dispositivos móviles con sistema operativo Android fue desarrollada en lenguaje de java, mientras que la base de conocimientos está basada en ontologías que fueron implementadas en el lenguaje OWL. Para el diseño de las ontologías se empleó un modelo multidimensional; es decir, que cada ontología modela una parte del entorno general, por ejemplo: personas, espacios físicos, trámites, por mencionar algunas. Para que la aplicación del dispositivo móvil pueda conectarse a la base de conocimientos, la aplicación móvil cuenta con un módulo llamado `ConsumidorWebService`, a través del cual se realiza el intercambio de mensajes entre el buscador semántico y la base de conocimientos. La base de conocimientos está alojada en un servidor, y para hacer uso de ella, se desarrolló un servicio web llamado `WebServicePt`, el cual recibe las consultas, las procesa y devuelve un resultado para la búsqueda en cuestión a la aplicación móvil.

Gracias al modelo ontólogico, el cual describe el entorno académico de la División de Ciencias Básicas e Ingeniería, los resultados para una búsqueda son más cercanos a lo que el usuario espera, además de reducir el tiempo de búsqueda. Por ejemplo, para una búsqueda sobre los documentos involucrados en el trámite de servicios social, el sistema responde con URL a los documentos para su descarga directa; otro ejemplo, para buscar la ubicación del cubículo de un profesor, el sistema regresa al usuario la ubicación con el edificio y número de cubículo, así como el horario de estancia del profesor en la Universidad.

6. Trabajo a futuro

Dentro de este proyecto existen puntos que pueden ser mejorados como son: la base de conocimientos, creando nuevas reglas de inferencia para la creación de conocimiento nuevo así como nuevas

sentencias SQWRL para las consultas, ampliar esta base con nuevos individuos y nuevas relaciones.

El almacenamiento de cada unos de los individuos en las antologías representa un problema de rendimiento, lo cual podría ser combinado con algún otro medio de almacenamiento como por ejemplo una base de datos.

Siendo esta una aplicación móvil puede ser migrada a los distintos sistemas operativos existentes, no solo ser una aplicación nativa para *Android*, puede ser también para *Apple-IOS* o *BlackBerry OS* y con el uso de nuevas tecnologías.

El servicio web puede ser ampliado para integrar nuevas operaciones y poder ser usado en un cliente de escritorio o ser combinado con otros servicios web para formar parte de una aplicación mas grande y compleja.

7. Anexo

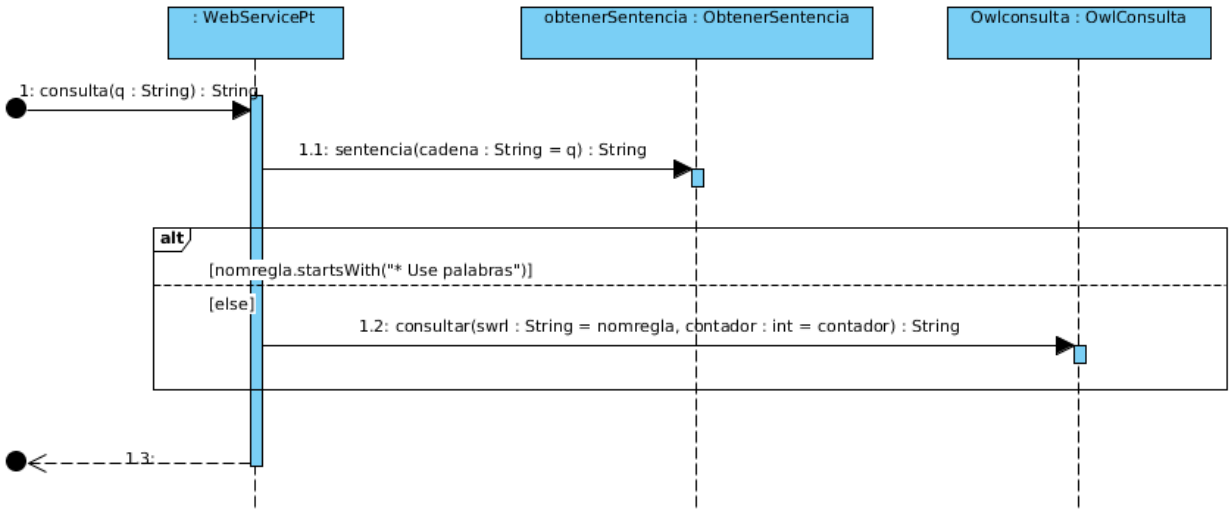


Figura 33: Diagrama de secuencia del servicio web

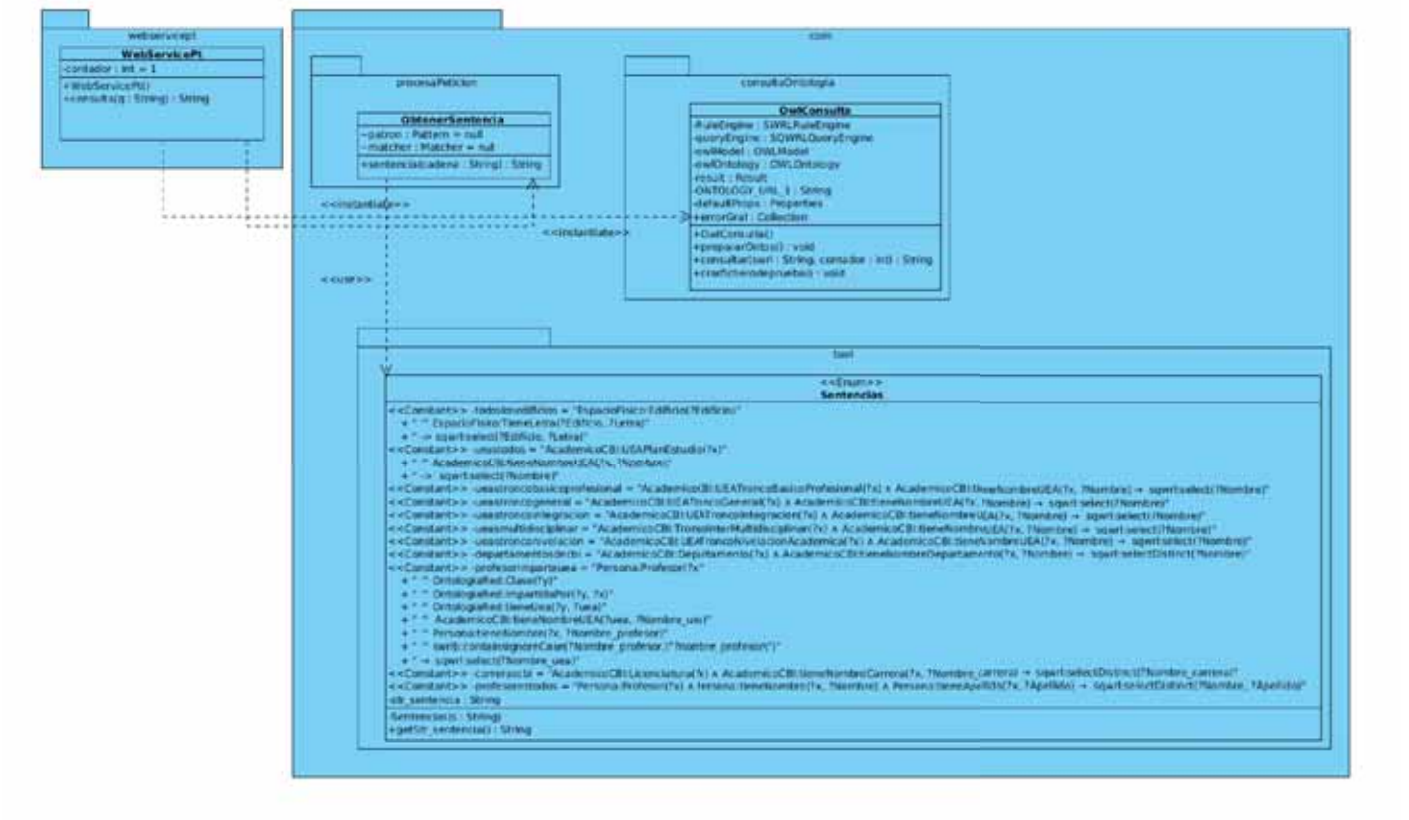


Figura 34: Diagrama de paquetes y clases del servicio web

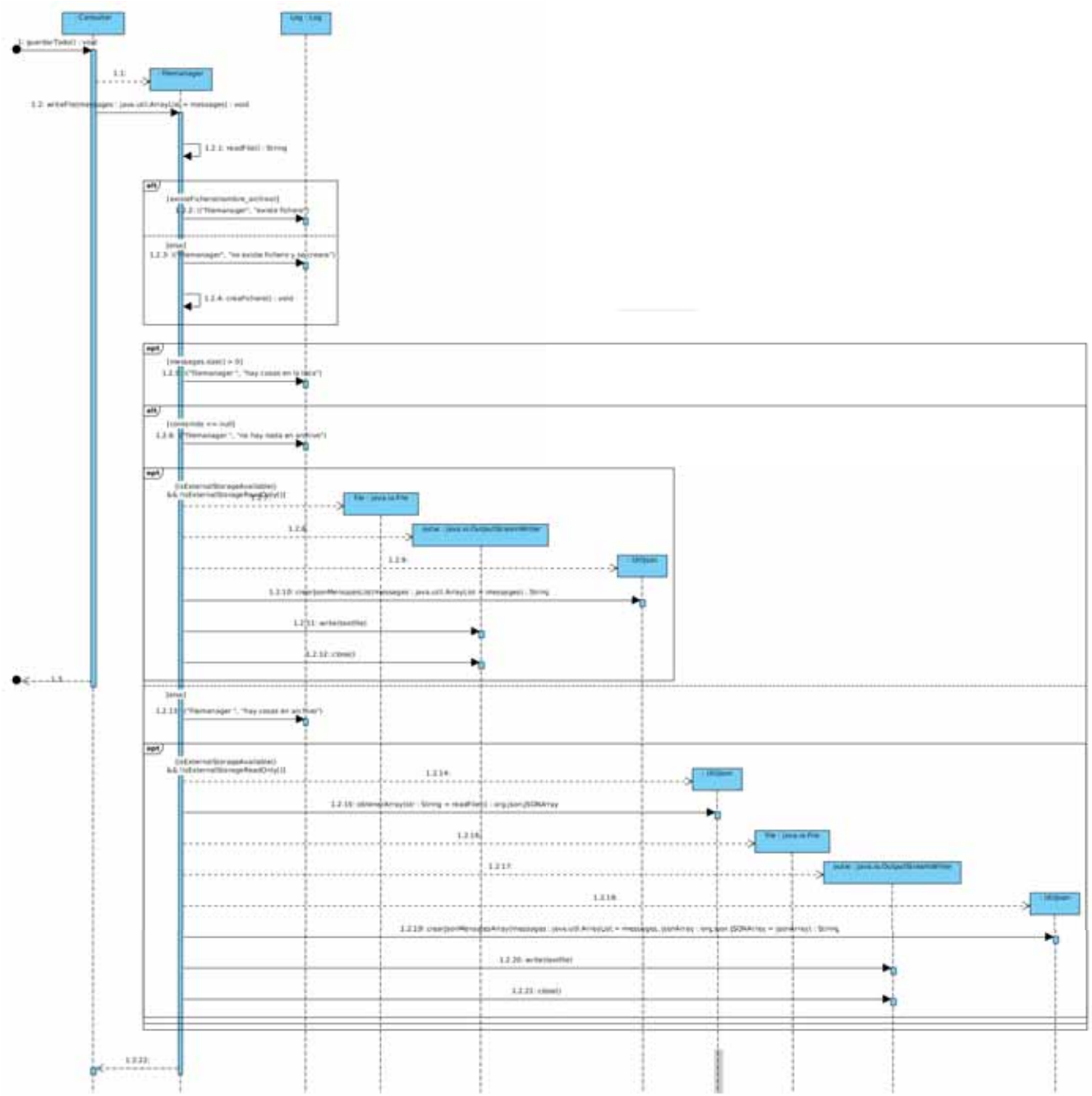


Figura 35: Diagrama de secuencia para el guardado del historial de búsquedas

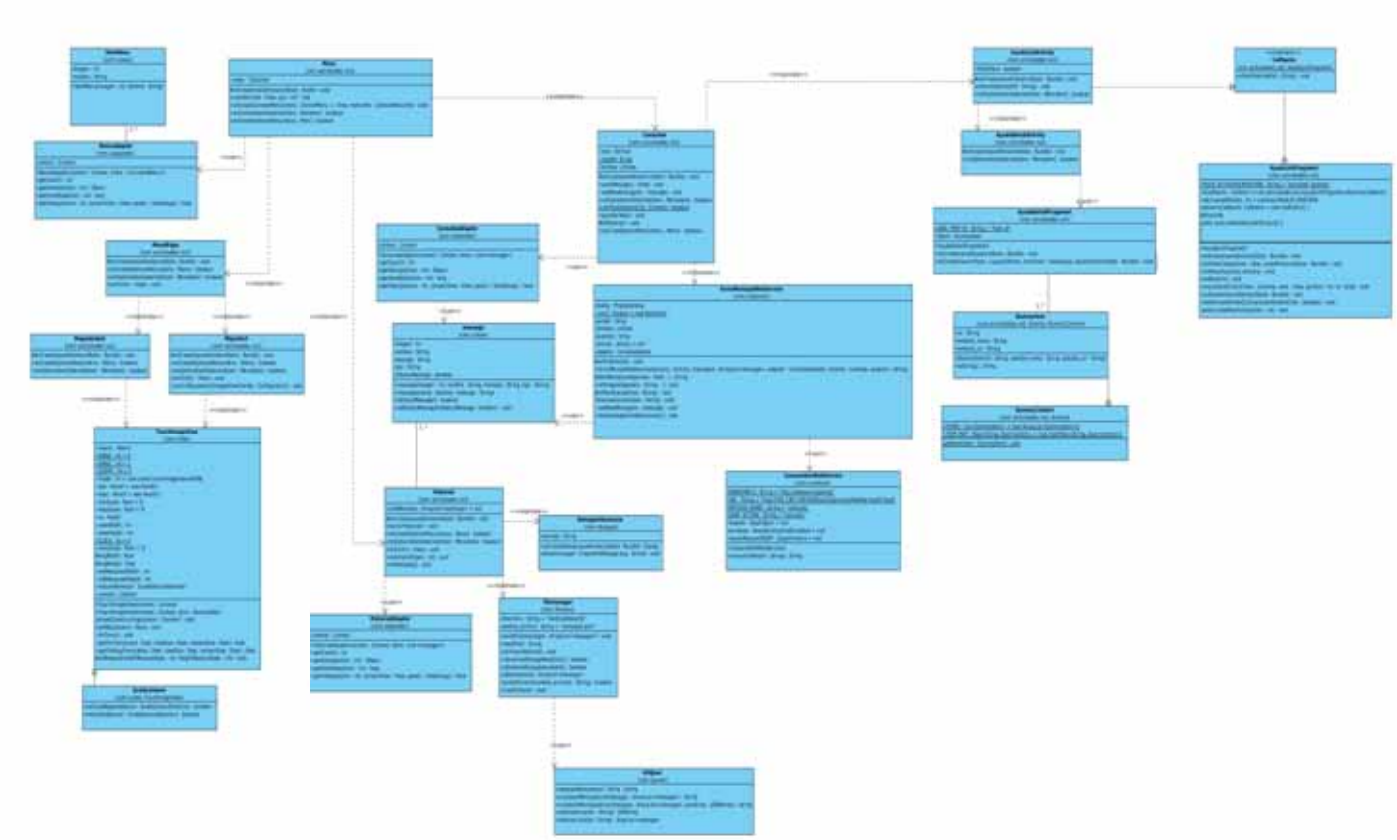


Figura 36: Diagrama de clases de la aplicación móvil

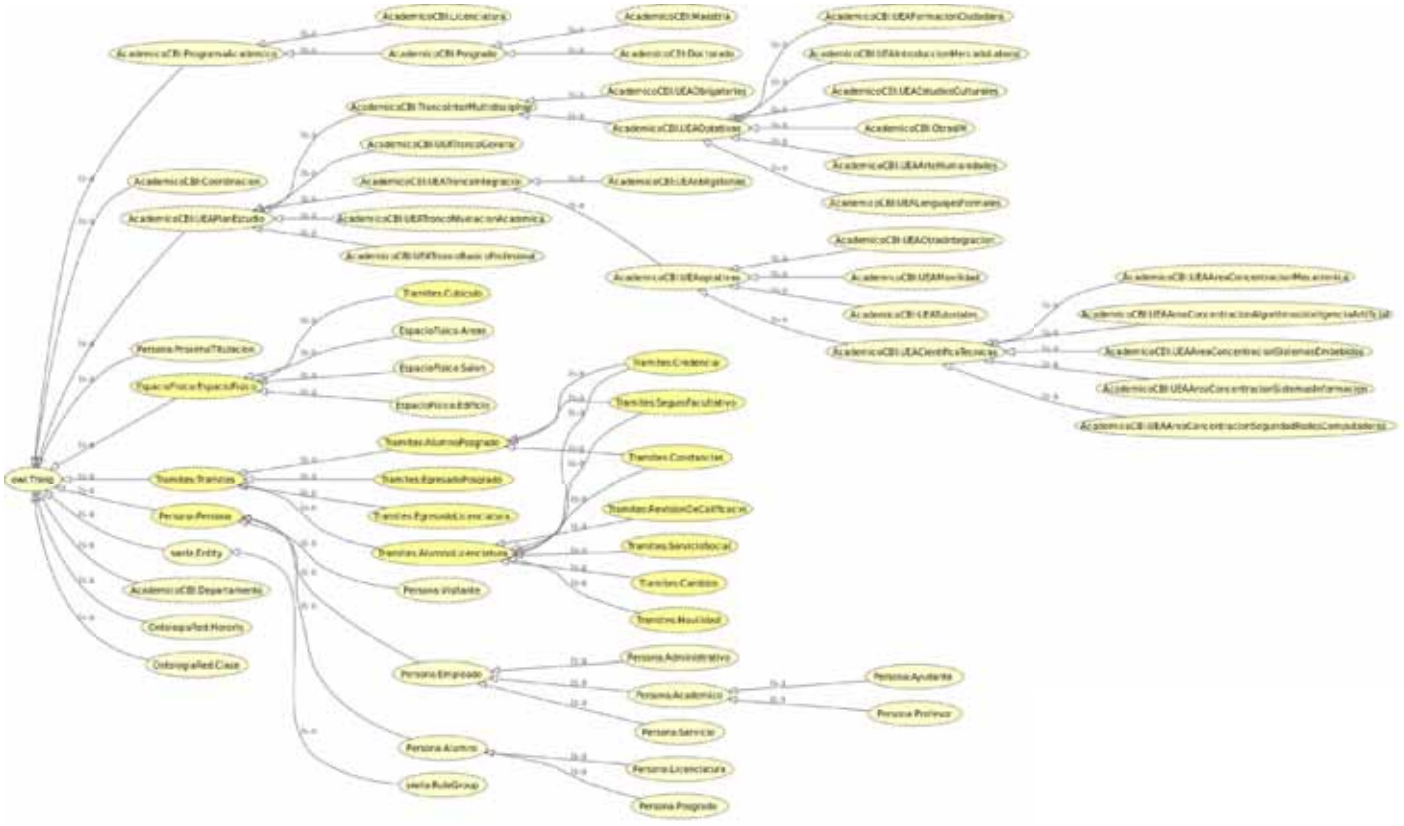


Figura 37: Diagrama general de la base de conocimiento

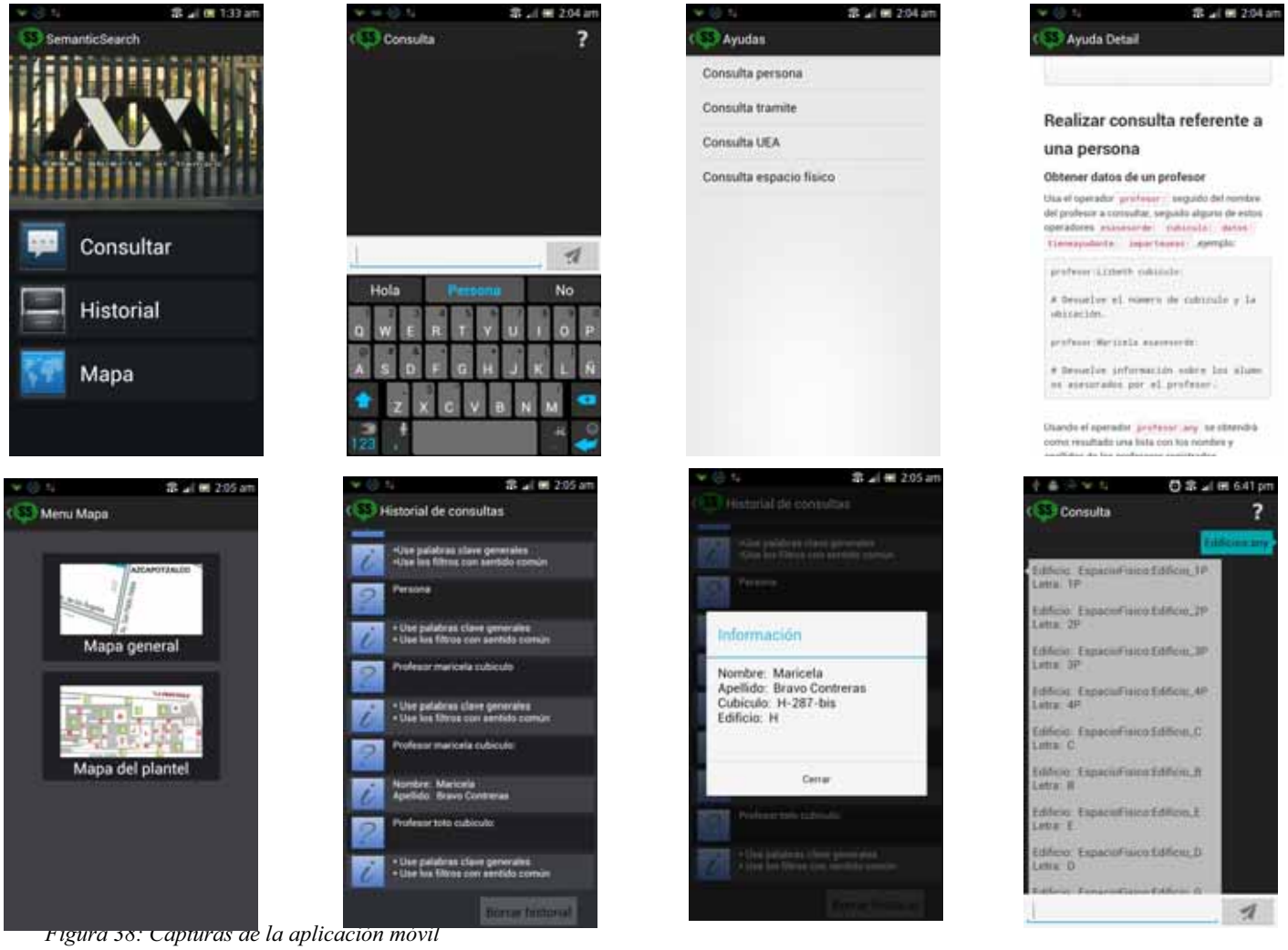


Figura 38: Capturas de la aplicación móvil

```java
package com.conexion;
import java.io.IOException;
import org.ksoap2.SoapEnvelope;
import org.ksoap2.serialization.SoapObject;
import org.ksoap2.serialization.SoapPrimitive;
import org.ksoap2.serialization.SoapSerializationEnvelope;
import org.ksoap2.transport.HttpTransportSE;
import org.xmlpull.v1.XmlPullParserException;
import android.util.Log;

public class ConsumidorWebService {

    private static final String NAMESPACE = "http://webservicept/xsd";
    private static String URL = "http://192.168.1.66:8080/axis2/services/WebServicePt?wsdl";
    private static final String METHOD_NAME = "consulta";
    private static final String SOAP_ACTION = "consulta";
    private SoapObject request = null;
    private SoapSerializationEnvelope envelope = null;
    private SoapPrimitive resultsRequestSOAP = null;

    public ConsumidorWebService() {}

    public String consultarWs(String qtn) {
        request = new SoapObject(NAMESPACE, METHOD_NAME);
        Log.i("consumidor WebService", "se creo el request");
        request.addProperty("arg0", qtn);
        envelope = new SoapSerializationEnvelope(SoapEnvelope.VER11);
        Log.i("consumidor WebService", "se creo en envelope");
        envelope.dotNet = false;
        envelope.setOutputSoapObject(request);
        Log.i("consumidor WebService",
                "se seteo el OutPut Soap Object con el request");
        HttpTransportSE transporte = new HttpTransportSE(URL,30000);
        Log.i("consumidor WebService", "Se creo el transporte HTTP");

        try {
            transporte.call(SOAP_ACTION, envelope);
            Log.i("consumidor WebService", "Transporte.call exitoso");
            resultsRequestSOAP = (SoapPrimitive) envelope.getResponse();
            Log.i("consumidor WebService", "RquestSOAP exitoso");
            return resultsRequestSOAP.toString();
        } catch (IOException e) {
            Log.w("try del transporte Consumidor Web service",
                    "Ocurrio un error en el transporte.call");
            e.printStackTrace();
        } catch (XmlPullParserException e) {
            e.printStackTrace();
        }
        return null;
    }
}
```

Código 10: Clase ConsumidorWebService

```
package com.conexion;

import java.util.ArrayList;
import java.util.Random;
import android.app.Activity;
import android.app.AlertDialog;
import android.app.ProgressDialog;
import android.content.DialogInterface;
import android.os.AsyncTask;
import android.widget.ListView;
import com.adaptador.ConsultaAdapter;
import com.clases.message;
import com.actividades.sst.R;

public class EnvioMensajeWebService extends AsyncTask<Void, String, String> {

    private ProgressDialog dialog;
    private ArrayList<message> messages;
    private ConsultaAdapter adapter;
    static Random rand = new Random();
    private String sender;
    private ListView listView;
    private String question;
    private Activity activity = null;

    @Override
    protected void onPreExecute() {
        this.dialog.setMessage("Consultando");
        this.dialog.setCanceledOnTouchOutside(false);
        this.dialog.show();
    }

    public EnvioMensajeWebService(Activity activity,
            ArrayList<message> messages, ConsultaAdapter adapter,
            ListView listView, String question) {
        super();
        this.messages = messages;
        this.adapter = adapter;
        this.listView = listView;
        this.sender = "Sistema";
        this.activity = activity;
        this.dialog = new ProgressDialog(activity);
        this.question = question;
    }

    @Override
    protected String doInBackground(Void... params) {
        this.publishProgress(String
                .format("%s ,realizando la consulta", sender));
        return new ConsumidorWebService().consultarWS(this.question);
    }
```

```
    @Override
    public void onProgressUpdate(String... v) {

        if (messages.get(messages.size() - 1).isStatusMessage()) {
            messages.get(messages.size() - 1).setMensaje(v[0]);
            adapter.notifyDataSetChanged();
            listView.setSelection(messages.size() - 1);
        } else {
            addNewMessage(new message(true, v[0]));
        }
    }

    @Override
    protected void onPostExecute(String text) {
        if (text != null) {
            finalizaeConsulta(text);
        } else {
            finalizaeConsulta("Error al conectarse con el servidor");
        }
    }

    private void finalizaeConsulta(String text) {
        if (messages.get(messages.size() - 1).isStatusMessage()) {
            messages.remove(messages.size() - 1);
        }

        addNewMessage(new message(R.drawable.info, "Ontology", text, "req"));

        if (this.dialog.isShowing()) {
            this.dialog.dismiss();
        }
    }

    void addNewMessage(message m) {
        messages.add(m);
        adapter.notifyDataSetChanged();
        listView.setSelection(messages.size() - 1);
    }

}
```

Código 11: Claser EnvioMensajeWebService

```java
package com.clases;

public class message {

    public message(int imagen, String nombre, String mensaje, String tipo) {
        super();
        this.imagen = imagen;
        this.nombre = nombre;
        this.mensaje = mensaje;
        this.tipo = tipo;
        this.isStatusMessage = false;
    }

    public message(boolean status, String message) {
        super();
        this.mensaje = message;
        this.tipo = "req";
        this.isStatusMessage = status;
    }

    private int imagen;
    private String nombre;
    private String mensaje;
    private String tipo;
    private boolean isStatusMessage;

    public int getImagen() {
        return imagen;
    }

    public void setImagen(int imagen) {
        this.imagen = imagen;
    }

    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    public String getMensaje() {
        return mensaje;
    }

    public void setMensaje(String mensaje) {
        this.mensaje = mensaje;
    }

    public String getTipo() {
        return tipo;
    }

}
```

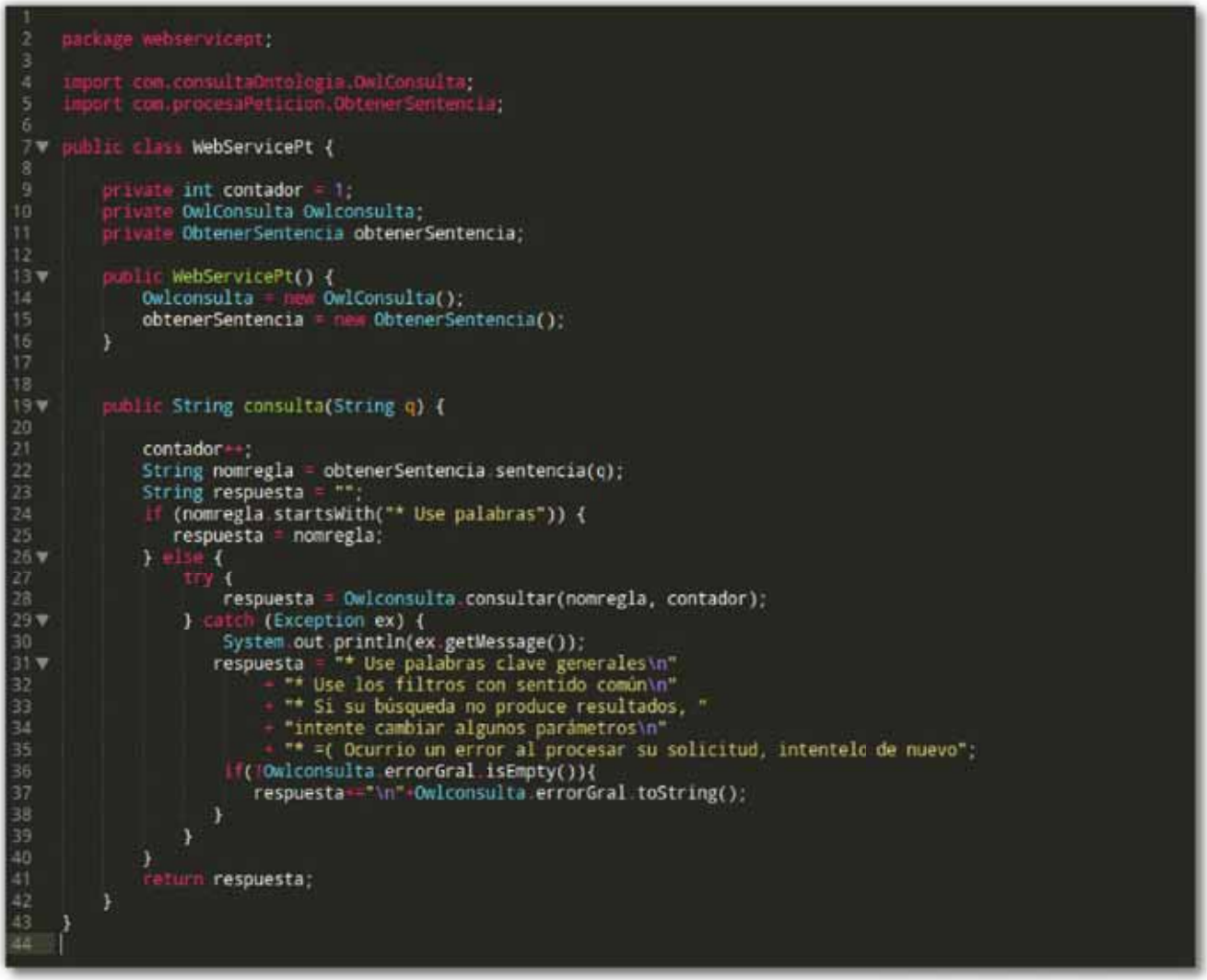
Código 12: Clase message

```
1
2   package com.actividades.ss1;
3
4
5   @SuppressLint("NewApi")
6   public class Consultar extends Activity {
7
8       ArrayList<message> messages;
9       ConsultaAdapter adapter;
10      EditText text;
11      static String sender;
12      private ListView listView;
13
14      @Override
15      protected void onCreate(Bundle savedInstanceState) {
16          super.onCreate(savedInstanceState);
17          this.setContentView(R.layout.activity_consulta);
18          getActionBar().setDisplayHomeAsUpEnabled(true);
19
20          if (!verificaConexion(this)) {
21              Toast.makeText(getBaseContext(),
22                      "Comprueba tu conexión a Internet.", Toast.LENGTH_SHORT)
23                      .show();
24              this.finish();
25          }
26
27          text = (EditText) this.findViewById(R.id.editText1);
28          sender = "Sistema";
29          this.setTitle("Consulta");
30          messages = new ArrayList<message>();
31          this.listView = (ListView) findViewById(R.id.listView);
32          adapter = new ConsultaAdapter(this, messages);
33          this.listView.setAdapter(adapter);
34
35          getWindow().setSoftInputMode(
36                  WindowManager.LayoutParams.SOFT_INPUT_STATE_ALWAYS_HIDDEN);
37
38      }
39
40      public void sendMessage(View v) {
41          String newMessage = text.getText().toString().trim();
42          if (newMessage.length() > 0) {
43              text.setText("");
44              addNewMessage(new message(R.drawable.question, "user", newMessage,
45                      "send"));
46              new EnvioMensajeWebService(this, messages, adapter, listView,
47                      newMessage).execute();
48          }
49      }
50
```

Código 13: Clase Consultar una extensión de la clase nativa de Android Activity parte 1

```
    void addNewMessage(message m) {
        messages.add(m);
        adapter.notifyDataSetChanged();
        listView.setSelection(messages.size() - 1);
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        switch (item.getItemId()) {
        case android.R.id.home:
            NavUtils.navigateUpFromSameTask(this);
            return true;
        case R.id.action_ayuda:
            Intent i = null;
            i = new Intent(this, AyudaListActivity.class);
            startActivity(i);
            Log.i("ActionBar", "ayudaaaaaaaaaa!");
            return true;
        }
        return super.onOptionsItemSelected(item);
    }


    public static boolean verificaConexion(Context ctx) {
        boolean bConectado = false;
        ConnectivityManager connec = (ConnectivityManager) ctx
                .getSystemService(Context.CONNECTIVITY_SERVICE);
        NetworkInfo[] redes = connec.getAllNetworkInfo();
        for (int i = 0; i < 2; i++) {

            if (redes[i].getState() == NetworkInfo.State.CONNECTED) {
                bConectado = true;
            }
        }
        return bConectado;
    }

    public void guardarTodo() {
        new filemanager().writeFile(messages);
    }

    @Override
    protected void onDestroy() {
        guardarTodo();
        super.onDestroy();
    }


    public boolean onCreateOptionsMenu(android.view.Menu menu) {
        MenuInflater inflater = getMenuInflater();
        inflater.inflate(R.menu.main_activity_actions, menu);
        return super.onCreateOptionsMenu(menu);
    }
}
```

Código 14: Clase Consultar una extensión de la clase nativa de Android Activity parte 2



```java

package webservicept;

import com.consultaOntologia.OwlConsulta;
import com.procesaPeticion.ObtenerSentencia;

public class WebServicePt {

    private int contador = 1;
    private OwlConsulta Owlconsulta;
    private ObtenerSentencia obtenerSentencia;

    public WebServicePt() {
        Owlconsulta = new OwlConsulta();
        obtenerSentencia = new ObtenerSentencia();
    }


    public String consulta(String q) {

        contador++;
        String nomregla = obtenerSentencia.sentencia(q);
        String respuesta = "";
        if (nomregla.startsWith("* Use palabras")) {
            respuesta = nomregla;
        } else {
            try {
                respuesta = Owlconsulta.consultar(nomregla, contador);
            } catch (Exception ex) {
                System.out.println(ex.getMessage());
                respuesta = "* Use palabras clave generales\n"
                        + "* Use los filtros con sentido común\n"
                        + "* Si su búsqueda no produce resultados, "
                        + "intente cambiar algunos parámetros\n"
                        + "* =( Ocurrio un error al procesar su solicitud, intentelo de nuevo";
                if(!Owlconsulta.errorGral.isEmpty()){
                    respuesta+="\n"+Owlconsulta.errorGral.toString();
                }
            }
        }
        return respuesta;
    }
}

```

Código 15: Clase WebServicePt del servicio web

```java
package com.consultaOntologia;
/**
 *
 * @author Morpheus
 */
public final class OwlConsulta {

    private SWRLRuleEngine RuleEngine;
    private SQWRLQueryEngine queryEngine;
    private OWLModel owlModel;
    private OWLOntology owlOntology;
    private Result result;
    private String ONTOLOGY_URL_1;
    private Properties defaultProps;
    public Collection errorGral;

    public OwlConsulta() {
        try {
            crarficherodeprueba();
            prepararOntos();
        } catch (Exception ex) {
        }
    }

    public void prepararOntos() {
        errorGral = new ArrayList();
        Properties defaultProps;
        defaultProps = new Properties();

        try {
            defaultProps.load(new FileInputStream("webServicePtConfig.properties"));
        } catch (IOException ex) {
            Logger.getLogger(OwlConsulta.class.getName()).log(Level.SEVERE, null, ex);
            errorGral.add(ex.getMessage());
        }


        ArrayList<String> listURL;
        Scanner s = null;
        try {
            s = new Scanner(new File("repository"));
        } catch (FileNotFoundException ex) {
            Logger.getLogger(OwlConsulta.class.getName()).log(Level.SEVERE, null, ex);
             errorGral.add(ex.getMessage());
        }
```

Código 16: Clase OwlConsulta del servicio web parte 1

```java
        listURL = new ArrayList();
        while (s.hasNext()) {
            listURL.add(s.next());
        }

        ONTOLOGY_URL_1 = defaultProps.getProperty("ONTOLOGY_URL_1");
        System.out.println("Ontologia: " + ONTOLOGY_URL_1);
        for (String uri : listURL) {
            System.out.println("Repositorio :" + uri);
        }

        System.out.println(ONTOLOGY_URL_1);

        Collection errors = new ArrayList();
        OwlProjectFromUriCreator creator = new OwlProjectFromUriCreator();

        if (!listURL.isEmpty()) {
            for (String uri : listURL) {
                try {
                    creator.addRepository(
                            new HTTPRepository(
                            new URL(uri)));
                } catch (MalformedURLException ex) {
                    Logger.getLogger(OwlConsulta.class.getName()).log(Level.SEVERE, null, ex);
                     errorGral.add(ex.getMessage());
                }
            }

        }

        creator.setOntologyUri(ONTOLOGY_URL_1);
        try {
            creator.create(errors);
        } catch (OntologyLoadException ex) {
            Logger.getLogger(OwlConsulta.class.getName()).log(Level.SEVERE, null, ex);
             errorGral.add(ex.getMessage());
        }

        owlModel = creator.getOwlModel();
```

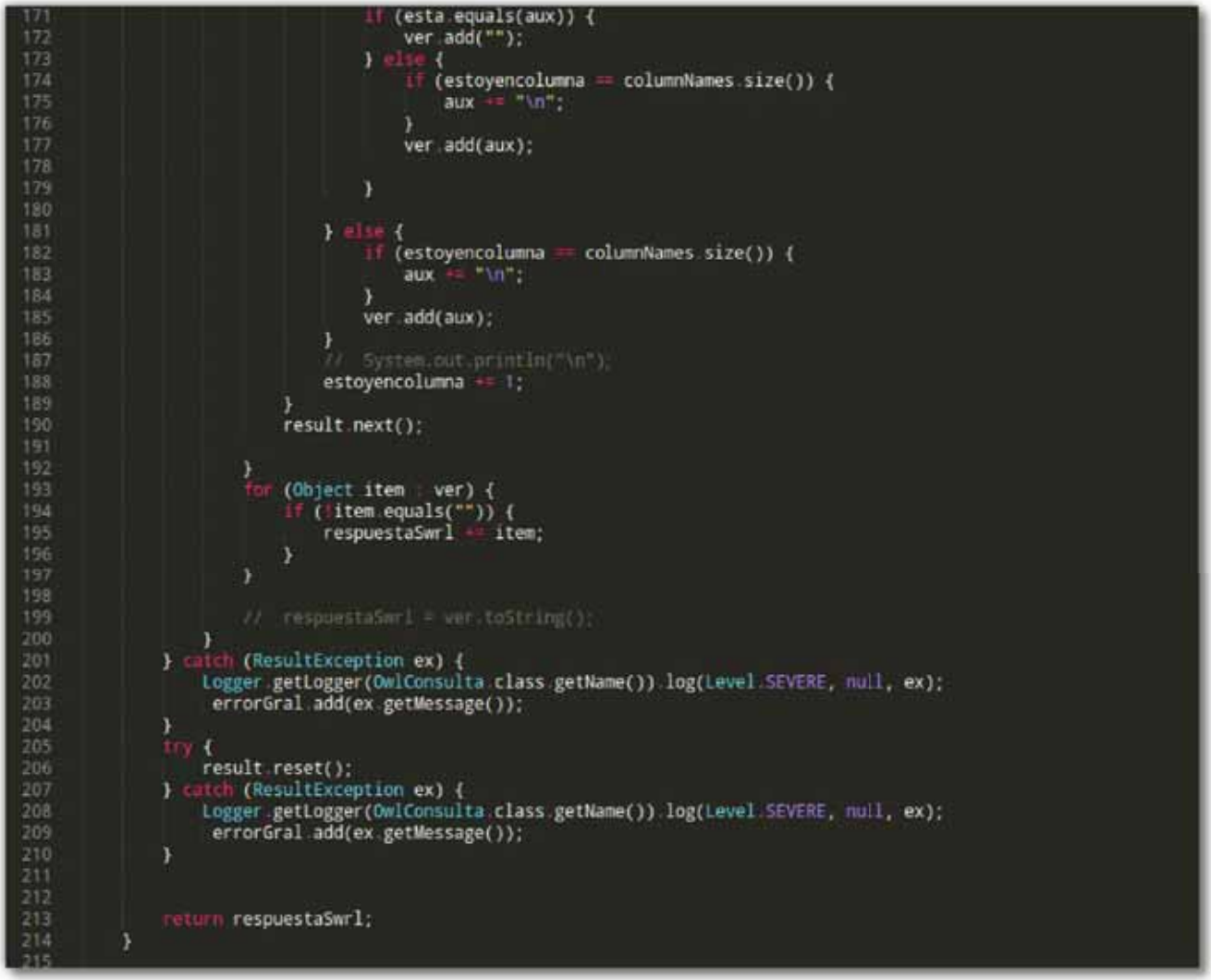
Código 17: Clase OwlConsulta del servicio web parte 2

```
        if (!errors.isEmpty()) {
            System.err.println("Error loading imports.");

        }
        try {
            RuleEngine = P3SWRLRuleEngineFactory.create("Jess", owlModel);
        } catch (SWRLRuleEngineException ex) {
            Logger.getLogger(OwlConsulta.class.getName()).log(Level.SEVERE, null, ex);
             errorGral.add(ex.getMessage());
        }
        try {
            queryEngine = P3SQWRLQueryEngineFactory.create(owlModel);
        } catch (ResultException ex) {
            Logger.getLogger(OwlConsulta.class.getName()).log(Level.SEVERE, null, ex);
             errorGral.add(ex.getMessage());
        }

    }

    public String consultar(String swrl, int contador)  {

        ((OWLModel) owlModel).resetOntologyCache();
        String respuestaSwrl = "";
        try {
            result = queryEngine.runSQWRLQuery("http://www.owl-ontologies.com/Tramites/regla-" + contador, swrl);
        } catch (ResultException ex) {
            Logger.getLogger(OwlConsulta.class.getName()).log(Level.SEVERE, null, ex);
             errorGral.add(ex.getMessage());
        }

        //result = queryEngine.runSQWRLQuery("reglaa-" + contador, swrl);
        //System.out.println("Son " + result.getNumberOfRows() + " objetos.");
        //respuestaSwrl += "Son " + result.getNumberOfRows() + " objetos \n\n";
        List<String> columnNames = null;
        try {
            columnNames = result.getColumnNames();
        } catch (ResultException ex) {
            Logger.getLogger(OwlConsulta.class.getName()).log(Level.SEVERE, null, ex);
             errorGral.add(ex.getMessage());
        }
```

Código 18: Clase OwlConsulta del servicio web parte 3

```
        try {
            if (result.getNumberOfRows() == 0) {
                respuestaSwrl = "* Use palabras clave generales\n"
                        + "* Use los filtros con sentido común\n"
                        + "* Si su búsqueda no produce resultados, "
                        + "intente cambiar algunos parámetros\n"
                        + "* No se obtuvieron resultados con el parametro ingresado, intentelo de nuevo =)";
            } else {
                //System.out.println("##### columnas nombre : " + columnNames);
                List ver = new ArrayList();

                while (result.hasNext()) {
                    int estoyencolumna = 1;
                    for (String colname : columnNames) {
                        ResultValue valor = result.getValue(colname);
                        String representation = "";

                        try {
                            ResultValue value = (result == null) ? null : result.getValue(colname);
                            if (value instanceof IndividualValue) {
                                IndividualValue objectValue = (IndividualValue) value;
                                representation += objectValue.getPrefixedName();
                            } else if (value instanceof LiteralValue) {
                                LiteralValue datatypeValue = (LiteralValue) value;
                                representation += datatypeValue.toString();
                            } else if (value instanceof ClassValue) {
                                ClassValue classValue = (ClassValue) value;
                                representation += classValue.toString();
                            } else if (value instanceof PropertyValue) {
                                PropertyValue propertyValue = (PropertyValue) value;
                                representation += propertyValue.getPrefixedName();
                            }
                        } catch (ResultException ex) {
                             errorGral.add(ex.getMessage());
                        }

                        String aux = colname.replace("?", "") + ":  " + representation + "\n";

                        //System.out.println("######### aux  " + aux);
                        //    System.out.println("-- " + colname + "  " + representation);
                        if (ver.size() >= columnNames.size()) {
                            //System.out.println("********* " + ver.get(ver.size() - columnNames.size()));
                            String esta = (String) ver.get(ver.size() - columnNames.size());
```

Código 19: Clase OwlConsulta del servicio web parte 4



```
                        if (esta.equals(aux)) {
                            ver.add("");
                        } else {
                            if (estoyencolumna == columnNames.size()) {
                                aux += "\n";
                            }
                            ver.add(aux);

                        }

                    } else {
                        if (estoyencolumna == columnNames.size()) {
                            aux += "\n";
                        }
                        ver.add(aux);
                    }
                    //  System.out.println("\n");
                    estoyencolumna += 1;
                }
                result.next();

            }
            for (Object item : ver) {
                if (!item.equals("")) {
                    respuestaSwrl += item;
                }
            }

            //  respuestaSwrl = ver.toString();
        }
    } catch (ResultException ex) {
        Logger.getLogger(OwlConsulta.class.getName()).log(Level.SEVERE, null, ex);
         errorGral.add(ex.getMessage());
    }
    try {
        result.reset();
    } catch (ResultException ex) {
        Logger.getLogger(OwlConsulta.class.getName()).log(Level.SEVERE, null, ex);
         errorGral.add(ex.getMessage());
    }


    return respuestaSwrl;
}
```

Código 20: Clase OwlConsulta del servicio web parte 5

Bibliografía

[1] S. Solís, La Web Semántica. Lulu Enterprises Incorporated, 2007.
[2] L. Criado-Fernández, Nosotros, los constructores de la Web Semántica.
[3] Guía Breve de Servicios Web. *Guía Breve de Servicios Web.* Disponible en http://www.w3c.es/Divulgacion/GuiasBreves/ServiciosWeb
[4] A. Pescador, Ontología. University of Texas: Losada, 1966.
[5] Horrocks, I.; Patel-Schneider, P. F.; Boley, H.; Tabet, S.; Grosof, B.; Dean, M. & others SWRL: A semantic web rule language combining OWL and RuleML W3C Member submission, 2004, 21, 79
[6] Moreau, J.-J.; Gudgin, M.; Nielsen, H. F.; Hadley, M. & Mendelsohn, N. SOAP Version 1.2 Part 1: Messaging Framework W3C, 2003. Disponible en http://www.w3.org/TR/2003/REC-soap12-part1-20030624/
[7] Orchard, D.; Haas, H.; Newcomer, E.; Ferris, C.; Champion, M.; Booth, D. & McCabe, F. Web Services Architecture W3C, 2004. Disponible en http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/
[8] Box, D., Ehnebuske, D., Kakivaya, G., Layman, A., Mendelsohn, N., Nielsen, H. F., ... & Winer, D. (2000). Simple object access protocol (SOAP) 1.1. Disponible en http://www.immagic.com/eLibrary/ARCHIVES/SUPRSDED/W3C/W000520N.pdf
[9] McGuinness, D. L. & van Harmelen, F. OWL Web Ontology Language Overview 2004. Disponible en http://www.w3.org/TR/2004/REC-owl-features-20040210/
[10] Protégé. Protégé. 2013. Disponible en http://protege.stanford.edu
[11] Protégé OWL API. Protégé OWL API. 2013 Disponible en http://protege.stanford.edu/plugins/owl/api/
[12] Brickley, D. & Guha, R. V. Resource Description Framework (RDF) Schema Specification 1.0: W3C Candidate Recommendation 27 March 2000. Disponible en http://www.w3.org/TR/2000/CR-rdf-schema-20000327/
[13] O'Connor, M. J. & Das, A. K. SQWRL: A Query Language for OWL. OWLED, 2009, 529. Disponible en http://171.67.213.129/file_asset/index.php/1723/BMIR-2009-1395.pdf
[14] JESS. JESS. 2013. Disponible en http://herzberg.ca.sandia.gov/.
[15] kSOAP2, kSOAP2, 2013. Disponible en http://ksoap2.sourceforge.net/
[16] REST, REST. 2013. Disponible en http://rest.elkstein.org/
[17] JSON, Introducing JSON. 2013. Disponible en http://www.json.org/
[18] Apache Tomcat, Apache Tomcat. 2013. Disponible en https://tomcat.apache.org/
[19] Apache Axis2/Java, Apache Axis2/Java. 2013. Disponible en https://axis.apache.org/axis2/java/core/
[20] Christensen, E.; Curbera, F.; Meredith, G.; Weerawarana, S. & others Web services description language (WSDL) 1.1 2001. Disponible en http://www.w3.org/TR/2001/NOTE-wsdl-20010315

Buscador semántico de recursos académicos de la DCBI en una plataforma de cómputo móvil.

Manual del Usuario

VERSION 1.0

MANUAL DEL USUARIO

TABLA DE CONTENIDO

1. INSTALACION DE LA APLICACIÓN MOVIL

1.1. Requerimientos mínimos del dispositivo móvil

Pantalla de 3.5″
Procesador 1GHz
512MB RAM
Android 3.0.X HONEYCOMB o superior
Almacenamiento + micro SD mínimo 128 MB

1.2. ¿Cómo Instalar la aplicación?

1.2.1. Configuración inicial del móvil

Para permitir la instalación de aplicaciones desde fuentes desconocidas:

En la pantalla de inicio, presiona el ícono **Pantalla de aplicaciones**.

Presiona **Ajustes > Seguridad.**

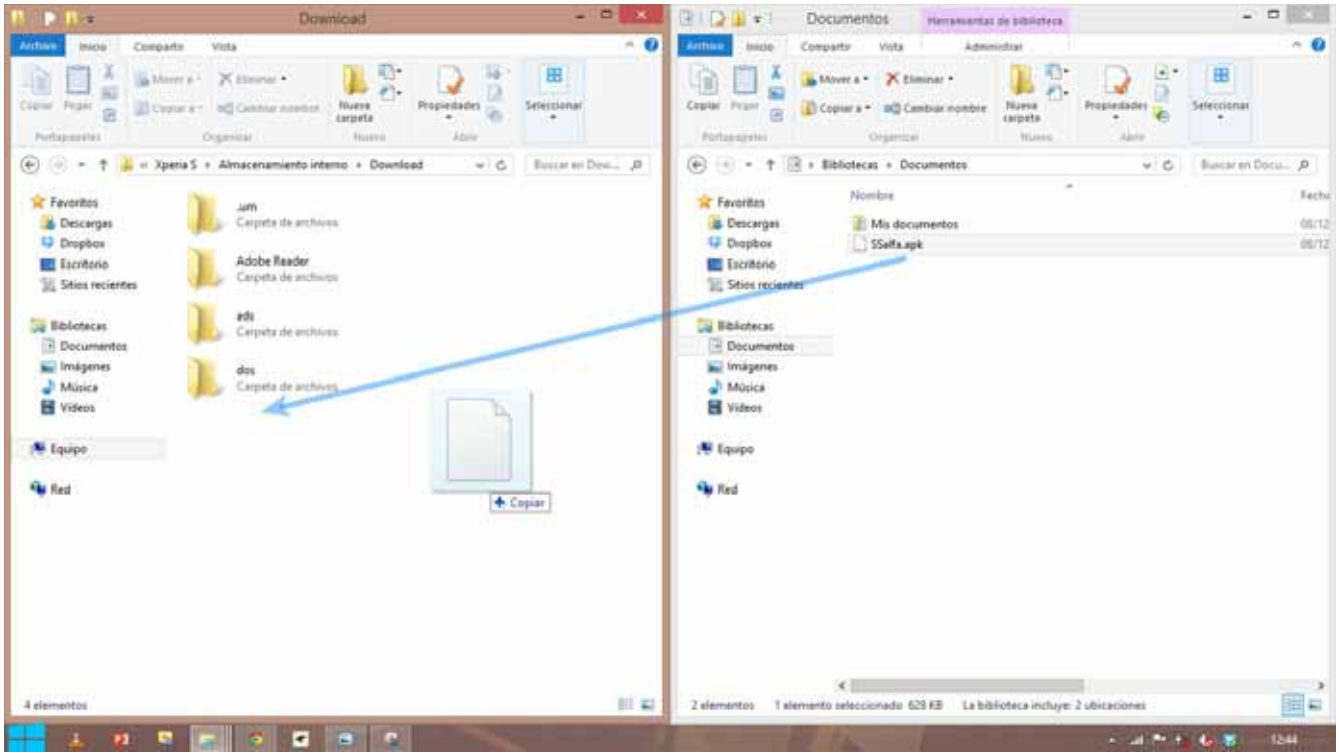
Marca la casilla de **Fuentes desconocidas.**

Si estás de acuerdo, presiona **Aceptar.**

Nota: De forma predeterminada, el teléfono está configurado para bloquear instalaciones desde fuentes desconocidas.

1.2.2. Copiar APK al dispositivo.

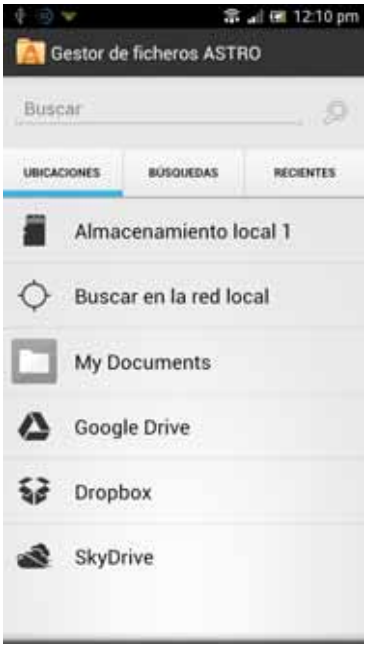
Conectar el móvil a la computadora y copiar el archivo SSalfa.apk a una carpeta de fácil acceso, en el ejemplo se copiara a la carpeta Download del móvil.



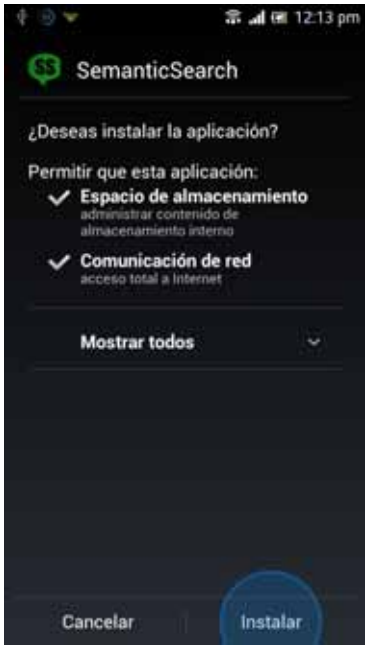
1.2.3. Instalación

Abrir su aplicación para la administración de archivos en este ejemplo se usa el **gestor de ficheros ASTRO**.

Navegar hasta la carpeta donde se copio el archivo SSalfa.apk .



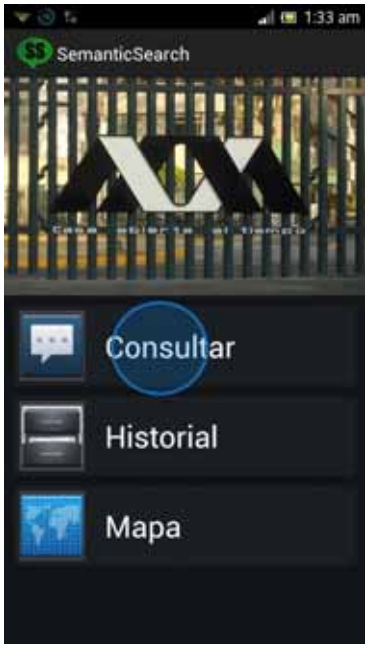
Presionar el **icono de la aplicación > Instalador de paquetes > Instalar > Hecho.**

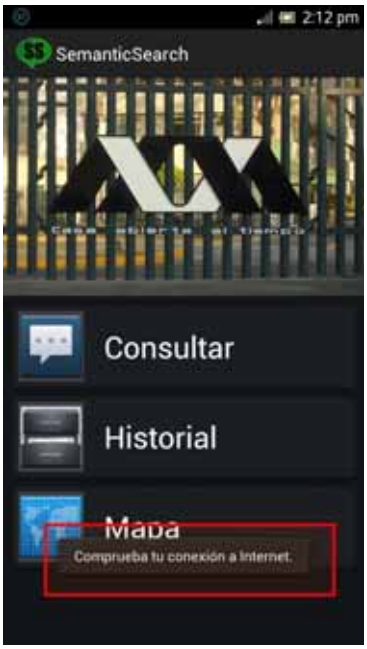


Una ves instalado en su **Pantalla de aplicaciones** aparecerá el icono de la aplicación móvil.

1.3. Inicio de consultas

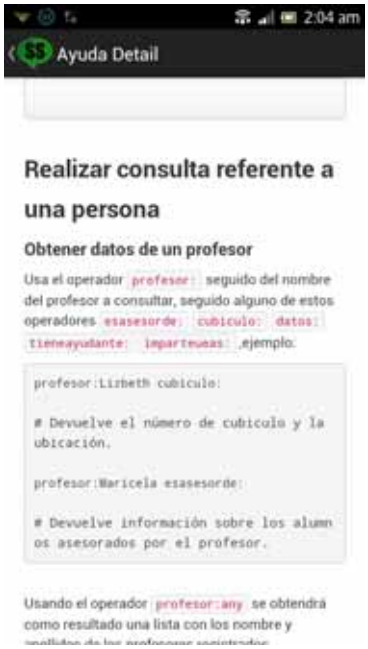
Después de la instalación, toque en el icono “SemanticSearch” para abrir la aplicación, y la pantalla de inicio aparecerá, tenga en cuenta que para realizar una consulta necesita estar conectado a una red.





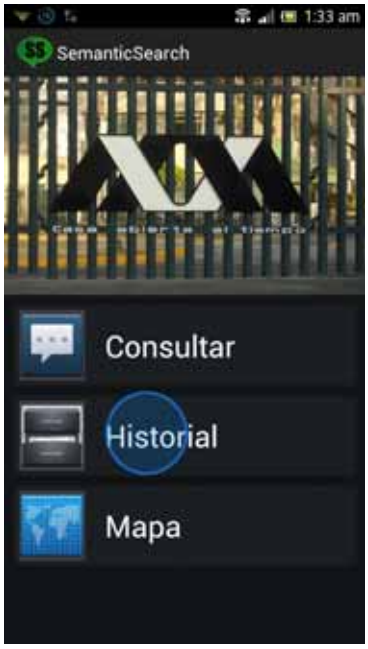
Una vez conectado a una red puede empezar a realizar sus consultas.

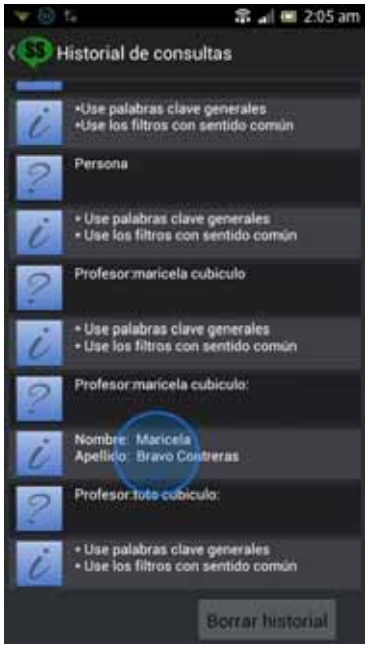
Use el signo de interrogación de la parte superior derecha para ir al manual de consultas.



1.4. Revisar el historial de consultas

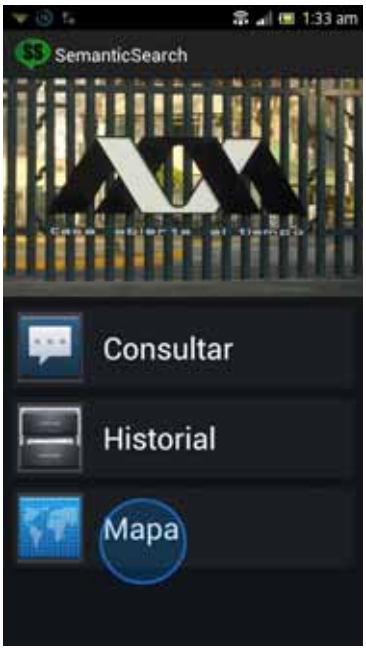
Dentro del menú principal seleccionar **Historial,** seguido puede seleccionar una pregunta o respuesta dada por el servidor para ver mas información.





1.5. Ver mapa general y de la unidad

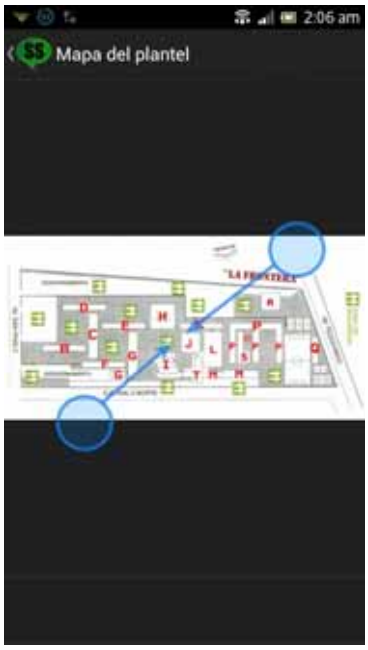
Seleccionar **Mapa** en el menú principal > **Mapa general o Mapa del plantel.**



Para hacer zoom se usan dos dedos.

Zoom +

Zoom -



2. INSTALACIÓN DEL SERVICIO WEB

2.1. Requerimientos mínimos del equipo de computo

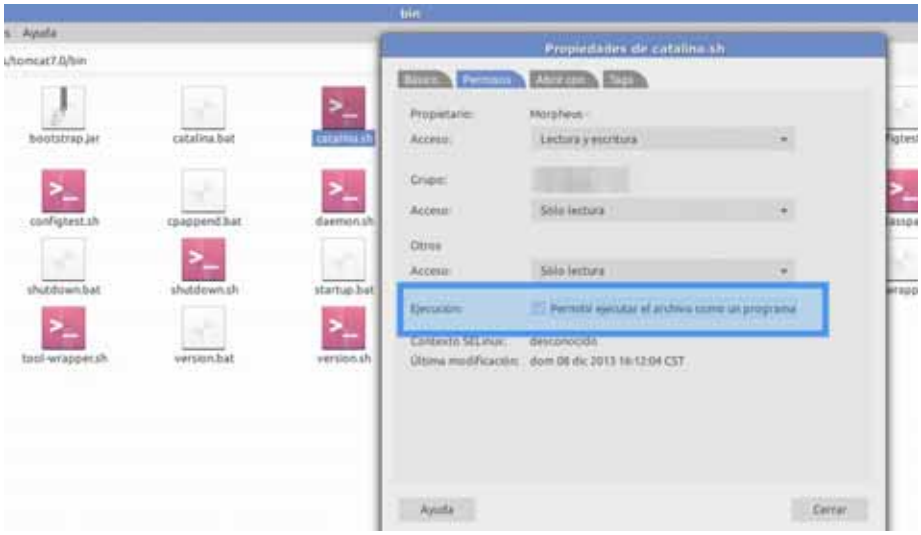
Tanto para Windows y para Linux tener instalado el JDK de java en su versión 7 o superior.
Espacio en disco duro de 250 MB.
2 GB de RAM.

2.2. ¿Cómo Instalar el servicio web?

2.2.1. Configuración de Apache Tomcat

Descargar apache tomcat en su versión 7 o superior.
Recomendado descargar el tomcat Core: zip. De su pagina oficial.
https://tomcat.apache.org/download-70.cgi
Descomprimir el contenido en una carpeta de fácil acceso, en este ejemplo se hará en Linux 12.04 en la carpeta /home/#usuario#/tomcat7.0,

Ir a la carpeta **bin** y seleccionar el archivo **catalina.sh** y cambiarle los permisos para ejecución.



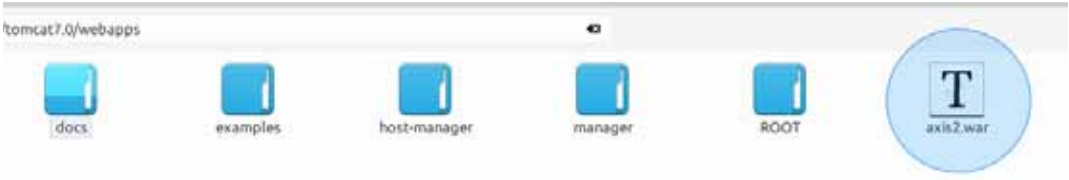
Abrir el archivo con un editor de texto y colocar dos variables:
JAVA_HOME y JRE_HOME
JAVA_HOME es la ruta donde se instalo en JDK de java, en este ejemplo la ruta es
JAVA_HOME="/usr/lib/jvm/java-7-oracle"
y el **JRE_HOME** es la carpeta jre dentro de la carpeta de instalación del JDK quedando asignada:
JRE_HOME="/usr/lib/jvm/java-7-oracle/jre"

```
#!/bin/sh
JAVA_HOME="/usr/lib/jvm/java-7-oracle"
JRE_HOME="/usr/lib/jvm/java-7-oracle/jre"
# Licensed to the Apache Software Foundation (ASF) under one or more
# contributor license agreements.  See the NOTICE file distributed with
# this work for additional information regarding copyright ownership.
# The ASF licenses this file to You under the Apache License, Version 2.0
# (the "License"); you may not use this file except in compliance with
# the License.  You may obtain a copy of the License at
#
```

Cerramos y se ha configurado Apache tomcat.

2.2.2. Configuración de Apache Axis2/Java en Apache tomcat

Descargar de la pagina oficial : https://axis.apache.org/axis2/java/core/download.cgi
Descargar la **version 1.6.2 WAR distribution en formato zip**.
Descomprimir y copiar el archivo axis2.war en la ruta **../tomcat7.0/webapps/**



En este punto podemos ya se puede arrancar Apache tomcat .
Abrir una terminal en la carpeta /tomcat7.0/bin/ y ejecutar el comando
~/tomcat7.0/bin**$ sh catalina.sh run** dando una salida similar a la siguiente:

```
ido hallada en java.library.path: /usr/java/packages/lib/i386:/lib:/usr/lib
dic 08, 2013 4:34:21 PM org.apache.coyote.AbstractProtocol init
Información: Initializing ProtocolHandler ["http-bio-8080"]
dic 08, 2013 4:34:21 PM org.apache.coyote.AbstractProtocol init
Información: Initializing ProtocolHandler ["ajp-bio-8009"]
dic 08, 2013 4:34:21 PM org.apache.catalina.startup.Catalina load
Información: Initialization processed in 474 ms
dic 08, 2013 4:34:21 PM org.apache.catalina.core.StandardService startInternal
Información: Arrancando servicio Catalina
dic 08, 2013 4:34:21 PM org.apache.catalina.core.StandardEngine startInternal
Información: Starting Servlet Engine: Apache Tomcat/7.0.47
dic 08, 2013 4:34:21 PM org.apache.catalina.startup.HostConfig deployWAR
Información: Despliegue del archivo /home/[redacted]/tomcat7.0/webapps/axis2.war de la aplicación web
[INFO] Clustering has been disabled
[INFO] Deploying module: ping-1.6.2 - file:/home/[redacted]/tomcat7.0/webapps/axis2/WEB-INF/modules/ping-1.6.2.mar
[INFO] Deploying module: metadataExchange-1.6.2 - file:/home/[redacted]/tomcat7.0/webapps/axis2/WEB-INF/modules/mex-1.6.2.mar
[INFO] Deploying module: addressing-1.6.2 - file:/home/[redacted]/tomcat7.0/webapps/axis2/WEB-INF/modules/addressing-1.6.2.mar
[INFO] Deploying module: script-1.6.2 - file:/home/[redacted]/tomcat7.0/webapps/axis2/WEB-INF/modules/scripting-1.6.2.mar
[INFO] Deploying module: soapmonitor-1.6.2 - file:/[redacted]/tomcat7.0/webapps/axis2/WEB-INF/modules/soapmonitor-1.6.2.mar
[INFO] Deploying module: jaxws-1.6.2 - file:/home/[redacted]/tomcat7.0/webapps/axis2/WEB-INF/modules/axis2-jaxws-mar-1.6.2.mar
[INFO] Deploying module: ntompolicy-1.6.2 - file:/home/[redacted]/tomcat7.0/webapps/axis2/WEB-INF/modules/ntompolicy-1.6.2.mar
[INFO] Deploying Web service: version-1.6.2.aar - file:/home/[redacted]/tomcat7.0/webapps/axis2/WEB-INF/services/version-1.6.2.aar
dic 08, 2013 4:34:24 PM org.apache.catalina.startup.HostConfig deployDirectory
Información: Despliegue del directorio /home/[redacted]/tomcat7.0/webapps/docs de la aplicación web
dic 08, 2013 4:34:24 PM org.apache.catalina.startup.HostConfig deployDirectory
Información: Despliegue del directorio /home/[redacted]/tomcat7.0/webapps/host-manager de la aplicación web
dic 08, 2013 4:34:24 PM org.apache.catalina.startup.HostConfig deployDirectory
Información: Despliegue del directorio /home/[redacted]/tomcat7.0/webapps/examples de la aplicación web
dic 08, 2013 4:34:25 PM org.apache.catalina.startup.HostConfig deployDirectory
Información: Despliegue del directorio /home/[redacted]/tomcat7.0/webapps/ROOT de la aplicación web
dic 08, 2013 4:34:25 PM org.apache.catalina.startup.HostConfig deployDirectory
Información: Despliegue del directorio /home/[redacted]/tomcat7.0/webapps/manager de la aplicación web
dic 08, 2013 4:34:25 PM org.apache.coyote.AbstractProtocol start
Información: Starting ProtocolHandler ["http-bio-8080"]
dic 08, 2013 4:34:25 PM org.apache.coyote.AbstractProtocol start
Información: Starting ProtocolHandler ["ajp-bio-8009"]
dic 08, 2013 4:34:25 PM org.apache.catalina.startup.Catalina start
Información: Server startup in 4223 ms
```

En el navegador web escribir la siguiente dirección http://localhost:8080/axis2/services/listServices si se muestra una pantalla como la siguiente, todo esta listo para colocar el servicio web.

Copiar en la carpeta ../tomcat7.0/ los archivos **WebServicePtConfig.properties** y **repository**, el primer archivo contiene la URL de la ontología principal en tanto el segundo las URL de las ontologías que se importaran.
Copiar en la carpeta ../tomcat7.0/webapps/axis2/WEB-INF/services/ el archivo con el nombre **WebServicePt.arr** quedando de la siguiente manera:

Abrir una vez mas una terminal en la carpeta /tomcat7.0/bin/ y ejecutar el comando
~/tomcat7.0/bin**$ sh catalina.sh run**
Ir a la dirección http://localhost:8080/axis2/services/listServices y verificar si existe un nuevo servicio llamado **WebServicePt**

En un entorno de Red de área local el servidor deberá tener la dirección **IP 192.168.1.66**