UNIVERSIDAD CARLOS III DE MADRID

Escuela Politécnica Superior

INGENIERÍA DE TELECOMUNICACIÓN

PROYECTO FIN DE CARRERA

"Cálculo de modos y tiempos de desplazamiento en una ciudad usando fuentes públicas"

Autor:

Juan Francisco Ortega Morán

Tutor:

Ángel García Olaya

El Proyecto Fin de Carrera titulado "Cálculo de modos y tiempos de desplazamiento en una ciudad usando fuentes públicas" ha sido defendido por Juan Francisco Ortega Morán a fecha de 19 de Junio de 2009 y calificado con una nota de …… por el tribunal compuesto por:

Presidente **Secretario**

D. Fernando Fernández Rebollo **D. Javier Ortiz Laguna**

Vocal

D. César de Pablo Sánchez

AGRADECIMIENTOS

Es de bien nacido ser agradecido, por eso quiero dar mis más sinceros agradecimientos a las personas que han contribuido a la realización de este proyecto, y también a las que de una manera u otra han contribuido a mi desarrollo personal. Todas estas personas son:

- Mi tutor, Ángel García Olaya, por su ayuda y participación directa en la realización del proyecto.

- Mis compañeros de carrera, y en especial, Carmelo, porque nos hemos apoyado mutuamente desde que nos fuimos juntos a estudiar a Madrid el segundo ciclo de Ingeniería de Telecomunicación.

- Mis compañeros de piso, Mauri y Román, porque ha sido un placer convivir con ellos y me han ayudado a sobrellevar el vivir lejos de mi familia, de mi novia y de mi tierra.

- Mi novia Conchi, porque, a pesar de estar la mayor parte del tiempo a distancia, ha estado a mi lado y me ha apoyado en todo momento. Es un pilar fundamental de mi vida.

- Y sobre todo, a mis padres y mi hermano, que son lo más grande que se puede tener en la vida, que es la familia. Ellos me han permitido tener los estudios que tengo, me han apoyado en todo momento y han contribuido especialmente a mi desarrollo personal. En definitiva, gracias a ellos soy quien soy.

A todas estas personas, mis más sinceros agradecimientos.

INTRODUCCIÓN

En la actualidad existen en Internet distintos servicios gratuitos que proporcionan itinerarios para desplazarse entre dos puntos de una ciudad. La idea del Proyecto Fin de Carrera es extraer automáticamente información de estos servicios para calcular el tiempo que se tarda en desplazarse entre dos puntos de una ciudad usando diversos medios de transporte.

La aplicación que se implementa en este proyecto se encuentra enmarcada en un proyecto más grande, formado por varios módulos, que consiste en un sistema de planificación de visitas turísticas a una ciudad, y que está destinada, por ejemplo, para PDA's. El módulo que implementa este proyecto es el de extracción de la información solicitada de Internet, por lo tanto, es un módulo de gran importancia para el funcionamiento del sistema. El resto de módulos se encargan de otros aspectos de la aplicación, como es la interfaz de usuarios, la comunicación entre módulos, la generación de mapas de la zona, la extracción automática de información de puntos de interés…

Como se mencionó anteriormente, es necesaria la extracción automática de información de Internet para llevar a cabo el proyecto. Esto está muy relacionado con la Web Semántica, la Minería Web y los Wrappers. La Web Semántica consiste en dotar de significado y conocimiento a la Web y la Minería Web consiste en la extracción de información de la Web, para lo cuál se sirve de los Wrappers. En nuestro caso, aplicamos las técnicas de Minería Web para la realización del proyecto, para lo cual necesitamos un Wrapper que procese las páginas Web y extraiga la información solicitada. Un paso previo es analizar páginas Web que faciliten la información de tiempo solicitada entre dos puntos de una ciudad, de las cuales hay que seleccionar las más idóneas para nuestro proyecto. Para ello son necesarios los motores de búsqueda, que son los principales instrumentos para buscar información en Internet. Sin embargo, éstos presentan dificultades a la hora de encontrar la información necesaria, derivadas principalmente del lenguaje HTML con el que se crean los documentos Web. Este lenguaje no dota de significado o semántica a los documentos, por lo tanto, es complicado encontrar páginas Web concretas con la información requerida. A esto trata de dar solución la Web Semántica dotando de significado y conocimiento la Web.

Antes de este proyecto no había tenido contacto con la Web Semántica ni con la Minería Web, y como me pareció un tema bastante interesante, decidí involucrarme en este proyecto, ya que es el futuro de la Web y cada vez se están implementando más aplicaciones relacionadas con estos temas. Además, mi tutor me ofreció facilidades a la hora de implementar la aplicación, pudiéndola realizar en el lenguaje de programación que yo quisiera, acorde a mis conocimientos.

ÍNDICE

Pág.

1. OBJETIVOS

OBJETIVOS

A continuación mostramos los objetivos que tratamos de cubrir en este proyecto:

- El principal objetivo es la implementación de un programa que realice la extracción automática de información de Internet. Esa información en concreto es el tiempo que se tarda entre dos puntos de una ciudad usando fuentes públicas. Éste es el objetivo básico, pero no es el único objetivo del proyecto. Vamos a concretar los objetivos que nos marcamos en este proyecto.

- El primer objetivo es investigar y estudiar todo lo relacionado con la Web Semántica y la Minería Web, ya que estos temas no los he tratado nunca y no tengo mucho conocimiento de ellos. Por lo tanto, lo primero es hacer un estudio teórico previo de Web Semántica y Minería Web para aprender y saber aplicarlo correctamente a mi proyecto.

- Para iniciar el proyecto en sí, lo primero es buscar en Internet mediante un motor de búsqueda páginas Web de donde podamos extraer la información de tiempo solicitada. Como obtendremos listas muy grandes de páginas Web, deberemos seleccionar las que más nos faciliten la labor en nuestro proyecto.

- Una vez seleccionadas las páginas Web, deberemos analizar su código fuente para crear un Wrapper que extraiga de ellas la información solicitada.

- Ya metidos en la programación en sí, debemos implementar un programa que primeramente analice la petición de entrada del usuario para obtener los dos puntos de la ciudad entre los que se quiere calcular el tiempo de desplazamiento entre ellos. Una vez obtenidos origen y destino, hay que programar un Wrapper para que acceda a las páginas Web seleccionadas y extraiga la información de tiempo solicitada. El programa debe finalizar creando una respuesta con los datos de tiempo extraídos.

- Estos son los objetivos del proyecto relacionados con el programa que implemente la aplicación. Un objetivo personal, aparte del de aprender conocimientos sobre Web Semántica y Minería Web, es el de profundizar y mejorar mis conocimientos sobre el lenguaje de programación C, ya que en las asignaturas de la carrera donde lo he utilizado siempre he tenido que seguir un guión sobre su utilización. Ahora es una muy buena ocasión para saber aplicar los conocimientos de C adquiridos, o aprenderlos en caso de necesitarlo, sin necesidad de guión, sino implementando un programa de gran envergadura de principio a fin por mí sólo.

2. INTRODUCCIÓN TEÓRICA

2.1. WEB SEMÁNTICA

2.1.1. Historia de la Web

La aparición de la WWW se puede situar en 1989 [Abrams 1998, Connolly 2000], cuando Tim Berners-Lee presentó su proyecto de "World Wide Web" [Berners-Lee 1989] en el CERN (Suiza), con las características esenciales que perduran en nuestros días. El propio Berners-Lee completó en 1990 el primer servidor Web y el primer cliente, y un año más tarde publicó el primer borrador de las especificaciones de HTML y HTTP. El lanzamiento en 1993 de Mosaic, el primer navegador de dominio público, compatible con Unix, Windows, y Macintosh, por el *National Center for Supercomputing Applications* (NCSA), marca el momento en que la WWW se da a conocer al mundo, extendiéndose primero en universidades y laboratorios, y en cuestión de meses al público en general, iniciando el que sería su vertiginoso crecimiento. Los primeros usuarios acogieron con entusiasmo la facilidad con que se podían integrar texto y gráficos y saltar de un punto a otro del mundo en una misma interfaz, y la extrema sencillez para incorporar contenidos a una Web mundial.

Por estas mismas fechas se define la interfaz CGI para la generación dinámica de páginas Web, con lo que se consigue ofrecer información actualizada en tiempo real, enlazar con bases de datos, o tener en cuenta entradas del usuario, y más aún, servir como punto de acceso y plataforma para la ejecución de aplicaciones distribuidas. En 1994 miembros del equipo que creó Mosaic desarrollan Netscape, un navegador con sensibles mejoras que contribuye a impulsar la propagación de la Web. Este mismo año se celebra el primer congreso internacional de la WWW, y unos meses más tarde se constituye el consorcio W3C, que desde entonces y presidido por Tim Berners-Lee, se ha hecho cargo de estandarizar las principales tecnologías Web. En 1995 Sun lanza oficialmente la primera versión del lenguaje Java, y un año más tarde Netscape presenta JavaScript. Estos lenguajes y otros posteriores permiten que las propias páginas Web contengan programas enteros, dando opción a una mayor autonomía respecto del servidor, mayor eficiencia, capacidad dinámica y capacidad de interacción.

Aunque es sumamente difícil medir el tamaño de la Web, hoy día millones de usuarios utilizan la Web, y ésta contiene una gran cantidad de documentos directamente accesibles por ellos. Esto incluye sólo lo que se ha dado en denominar la *Web superficial*, formada por los documentos estáticos accesibles en la Web. Se ha calculado que la llamada *Web profunda*, constituida por las bases de datos cuyos contenidos, no directamente accesibles, se hacen visibles mediante páginas generadas dinámicamente, puede contener un tamaño de información varios cientos de veces mayor, y de mucha mejor calidad, que la Web superficial, y crece a un ritmo aún mayor que ésta [O'Neill 2003]. Se estima que el tamaño de la Web profunda ha superado ya al volumen total de información impresa existente en todo el planeta.

Hoy casi todo está representado de una u otra forma en la Web, y con la ayuda de un buen buscador, podemos encontrar información sobre casi cualquier cosa que necesitemos. La Web está cerca de convertirse en una enciclopedia universal del conocimiento humano. Por otra parte la Web nos permite realizar diferentes actividades de nuestra vida diaria con una comodidad, economía y eficiencia sin precedentes: sin movernos de casa podemos comprar todo tipo de productos y servicios, gestionar una cuenta bancaria, buscar un restaurante, consultar la cartelera, leer la prensa, localizar a

una persona, matricularnos en la universidad, acceder a un callejero, o trabajar desde nuestro domicilio.

No obstante, en este panorama tan favorable hay espacio para mejoras. Por ejemplo, el enorme tamaño que ha alcanzado la Web, a la vez que es una de las claves de su éxito, hace que algunas tareas (por ejemplo encontrar la planificación óptima con transporte, alojamiento, etc., entre todas las posibles para un viaje bajo ciertas condiciones), requieran un tiempo excesivo para una persona o resulten sencillamente inabarcables. Desarrollar programas que realicen estas tareas en nuestro lugar es enormemente complicado, ya que es muy difícil reproducir, y más costoso aún mantener, en una máquina la capacidad de una persona para comprender los contenidos de la Web tal y como están codificados actualmente.

La asombrosa eficacia de los buscadores actuales tiene también sus límites. Por ejemplo, si queremos conocer la historia de Netscape, los resultados de una consulta como "Netscape history" nos informan sobre las herramientas de históricos de este navegador, pero no nos dicen nada sobre el origen y evolución de Netscape. Igualmente, para averiguar qué organismo se ocupa de estandarizar CGI, o en qué fecha apareció la primera versión de Java, necesitaremos realizar varias consultas y leer varios documentos y artículos hasta llegar indirectamente a la respuesta buscada. Si buscamos un "artículo sobre García Márquez", encontraremos decenas de artículos de García Márquez, pero ninguno que trate sobre este autor. Si preguntamos sobre estándares XML para la enseñanza ("XML education"), la mayor parte de los resultados se referirán a la enseñanza de XML.

Todos estos ejemplos son el síntoma de una causa común: la falta de capacidad de las representaciones en que se basa la Web actual para expresar significados. Los contenidos y servicios en la Web se presentan en formatos (p.e. HTML) e interfaces (p.e. formularios) comprensibles por personas, pero no por máquinas.

En estas condiciones es poco viable automatizar tareas mediante software en sustitución del humano. Un programa puede llevar al usuario hasta lugares en la Web, generar, transportar, transformar y ofrecer la información a las personas, pero la máquina sencillamente no sabe lo que esta información significa, y por tanto su capacidad de actuación autónoma es muy limitada. Esta misma limitación expresiva hace que la noción de semántica que manejan los buscadores Web se limite a palabras clave con pesos, pero planas e inconexas, lo que no permite reconocer ni solicitar significados más elaborados.

2.1.2. La Web Semántica

La Web semántica [Berners-Lee 2001] propone superar las limitaciones de la Web actual mediante la introducción de descripciones explícitas del significado, la estructura interna y la estructura global de los contenidos y servicios disponibles en la WWW. Frente a la semántica implícita, el crecimiento caótico de recursos, y la ausencia de una organización clara de la Web actual, la Web semántica aboga por clasificar, dotar de estructura y anotar los recursos con semántica explícita procesable por máquinas.

La Web semántica (del inglés *semantic Web*) se basa en la idea de añadir metadatos semánticos a la WWW. Esas informaciones adicionales —que describen el contenido, el significado y la relación de los datos— se deben proporcionar de manera formal, para que así sea posible evaluarlas automáticamente por máquinas de procesamiento. El objetivo es mejorar Internet ampliando la interoperabilidad entre los sistemas informáticos y reducir la necesaria mediación de operadores humanos.

Se trata de una corriente, promovida por el propio inventor de la Web y presidente del consorcio W3C, cuyo último fin es lograr que las máquinas puedan entender, y por tanto utilizar, lo que la Web contiene. Esta nueva Web estaría poblada por agentes o representantes software capaces de navegar y realizar operaciones por nosotros para ahorrarnos trabajo y optimizar los resultados. Para conseguir esta meta, la Web semántica propone describir los recursos de la Web con representaciones procesables (es decir, entendibles) no sólo por personas, sino por programas que puedan asistir, representar, o reemplazar a las personas en tareas rutinarias o inabarcables para un humano. Las tecnologías de la Web semántica buscan desarrollar una Web más cohesionada, donde sea aún más fácil localizar, compartir e integrar información y servicios, para sacar un partido todavía mayor de los recursos disponibles en la Web. Tim Berners-Lee, el creador de la idea, la expresó de la siguiente manera [Berners-Lee 2001]:

"Mi sueño es una Web en la que las máquinas sean capaces de analizar todos los datos -contenido, enlaces y transacciones entre la gente y los ordenadores-. La 'Web Semántica', que haría esto posible, está todavía por llegar, pero cuando llegue, la rutina de nuestras compras, burocracia y vida diaria será gestionada por máquinas hablando con máquinas. Los 'Agentes Inteligentes' que han sido anunciados durante décadas se harán por fin realidad".

La idea es que los datos puedan ser utilizados y "comprendidos" por los ordenadores sin necesidad de supervisión humana, de forma que los agentes Web puedan diseñarse para tratar la información situada en las páginas Web de manera semiautomática. Se trata de convertir la información en conocimiento, referenciando datos dentro de las páginas Web a metadatos con un esquema común consensuado sobre algún dominio. Los metadatos no sólo especificarán el esquema de datos que debe aparecer en cada instancia, sino que además podrán tener información adicional de cómo hacer deducciones con ellos, es decir, axiomas que podrán aplicarse en los diferentes dominios que trate el conocimiento almacenado. Con ello, se mejorará la búsqueda de información y se potenciará el desarrollo de aplicaciones de comercio electrónico, ya que las anotaciones de información seguirán un esquema común, y los buscadores Web compartirán con las anotaciones Web los mismos esquemas. Empresas que traten con clientes y proveedores, podrán intercambiar sus datos de productos siguiendo estos esquemas comunes consensuados. Los agentes Web no sólo encontrarán la información de forma precisa, sino que podrán realizar inferencias automáticamente buscando información relacionada con la que se encuentra situada en las páginas, y con los requerimientos de la consulta indicada por el usuario.

2.1.3. Capas de la Web Semántica

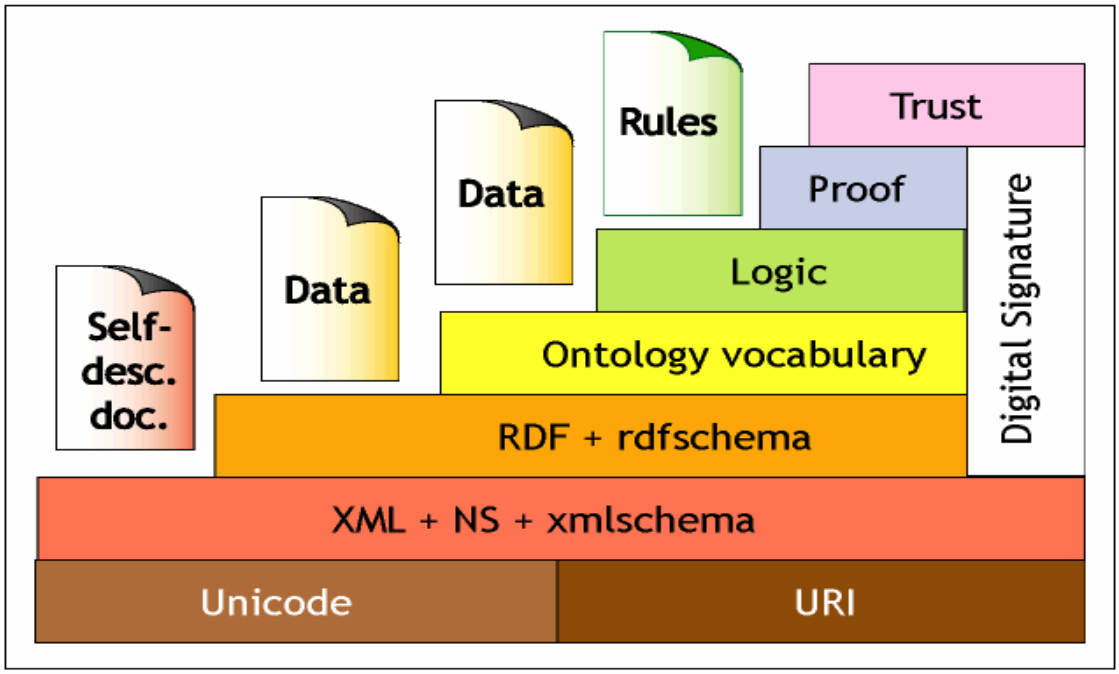


Figura 1. Capas de la Web Semántica

Unicode:

Es un estándar cuyo objetivo es proporcionar el medio por el cual un texto en cualquier forma e idioma pueda ser codificado para el uso informático. El mismo nos permite mostrar información en cualquier idioma y con la certeza de que no aparezcan símbolos extraños.

URI:

Las URI (Uniform Resource Identifier / Identificador Uniforme de Recursos) no son más que cadenas de caracteres usadas para identificar o nombrar un recurso, que puede ser documentos, páginas o dirección de correo electrónico, entre otros.

El propósito principal de esta identificación es permitir la interacción con las representaciones del recurso sobre una red, típicamente el WWW, usando protocolos específicos. Las URIs se definen en los esquemas definiendo una sintaxis específica y protocolos asociados.

En la Web Semántica las URIs son las encargadas de identificar objetos. Todos los objetos pueden ser identificados mediante una URI. Si dos objetos cuentan con la misma URI pueden existir colisiones.

La sintaxis de las URIs es cuidadosamente controlada por el IETF, que ha publicado el RFC 2396 como la especificación general para las URI.

XML+NS+xmlschema:

Esta es la capa más técnica de la Web Semántica. En ella se encuentran agrupadas las diferentes tecnologías que posibilitan la comunicación entre agentes.

El XML (eXtensible Markup Language) nos ofrece un formato común para el intercambio de documentos.

NameSpaces (NS) proporciona un método para cualificar elementos y atributos de nombres usados en documentos XML asociándolos con espacios de nombre identificados por referencias URIs.

XML Schema es un lenguaje que permite describir la estructura y restringir el contenido de documentos XML.

RDF+rdfschema:

Está basada en la capa anterior, define el lenguaje universal con el que podemos expresar diferentes ideas en la Web Semántica.

RDF es un lenguaje que define un modelo de datos para describir recursos mediante tripletas sujeto-predicado-objeto. Los dos primeros serán URIs y el tercero puede ser URI o un valor literal.

RDF Schema es un vocabulario RDF que nos permite describir recursos mediante una orientación a objetos. Esta capa no sólo ofrece una descripción de los datos, sino también cierta información semántica.

Ontology (Ontologías):

Como se verá en el apartado 2.6, nos permite clasificar la información. Esta capa permite extender la funcionalidad de la Web Semántica agregando nuevas clases y propiedades para describir los recursos.

Logic (Lógica):

Además de ontologías se precisan reglas de inferencia. Para que la Web Semántica resulte suficientemente expresiva para ayudarnos en una amplia variedad de situaciones, será necesario construir un lenguaje lógico potente para hacer inferencias.

Proof (Pruebas):

Se intercambiarán "pruebas" escritas en el lenguaje unificador de la Web Semántica. Este lenguaje posibilita las inferencias lógicas realizadas a través del uso de reglas de inferencia.

Trust (Confianza):

Hasta que no se hayan comprobado de forma exhaustiva las fuentes de información, los agentes deberían ser muy escépticos acerca de lo que leen en la Web Semántica.

Digital Signature (Firma digital):

Utilizada por los ordenadores y agentes para verificar que la información ha sido ofrecida por una fuente de confianza. Las firmas digitales son simplemente pequeños bits de código que uno puede usar para verificar que alguien escribió cierto documento.

2.1.4. Componentes de la Web Semántica

Los principales componentes de la Web Semántica son los metalenguajes y los estándares de representación **XML**, **XML Schema**, **RDF**, **RDF Schema** y **OWL**.

El primer lenguaje para la construcción de la Web Semántica fue SHOE, creado por Jim Hendler en la Universidad de Maryland en 1997. Desde entonces se han definido otros lenguajes y estándares con finalidad similar, como XML, RDF, DAML+OIL, y más recientemente OWL, por citar los más importantes.

2.1.4.1. XML

XML (eXtensible Markup Language) es un lenguaje de marcado de propósito general. Aporta la sintaxis de los documentos estructurados, a la que habrá que añadir restricciones semánticas propias de las distintas aplicaciones.

La versión 1.0 del lenguaje XML es una recomendación del W3C desde Febrero de 1998, pero se ha trabajado en ella desde un par de años antes. Está basado en el anterior estándar SGML (Standard Generalized Markup Language, ISO 8879), que data de 1986, pero que empezó a gestarse desde principios de los años 70, y a su vez basado en el GML creado por IBM en 1969.

SGML proporciona un modo consistente y preciso de aplicar etiquetas para describir las partes que componen un documento, permitiendo además el intercambio de documentos entre diferentes plataformas. Sin embargo, el problema que se atribuye a SGML es su excesiva dificultad. Así que, manteniendo su misma filosofía, de él se derivó XML como subconjunto simplificado, eliminando las partes más engorrosas y menos útiles. Como su padre, XML es un metalenguaje: es un lenguaje para definir lenguajes. Los elementos que lo componen pueden dar información sobre lo que contienen, no necesariamente sobre su estructura física o presentación, como en HTML.

Usando SGML se definió precisamente el HTML. En una primera aproximación se puede decir que mediante XML también podríamos definir el HTML, ya que HTML es simplemente un lenguaje, mientras que XML, como se ha dicho, es un metalenguaje, esto es, un lenguaje para definir lenguajes.

2.1.4.1.1. Conceptos de XML

Hay dos tipos de documentos XML: **válidos y bien formados**:

- **Bien formados:** son todos los que cumplen las especificaciones del lenguaje respecto a unas reglas sintácticas, sin estar sujetos a unos elementos fijados en un DTD, que, como veremos más adelante, son definiciones de los elementos que puede incluir un documento XML. De hecho los documentos XML deben tener una estructura jerárquica muy estricta, y los documentos bien formados deben cumplirla.
- **Válidos:** Además de estar bien formados, siguen una estructura y una semántica determinada por un DTD: sus elementos y sobre todo la estructura jerárquica que define el DTD, además de los atributos, deben ajustarse a lo que el DTD dicte.

Con la línea <?xml version="1.0" encoding="UTF-8" standalone="yes"?> deben empezar todos los documentos XML, ya que es la que indica que lo que la sigue es XML. Aunque es opcional, es más que recomendable incluirla siempre. Puede tener varios atributos (los campos que van dentro de la declaración), algunos obligatorios y otros no:

- *version*: indica la versión de XML usada en el documento. Es obligatorio ponerlo, a no ser que sea un documento externo a otro que ya lo incluía.

- *encoding*: la forma en que se ha codificado el documento. Por defecto es UTF-8, aunque podrían ponerse otras, como UTF-16, US-ASCII, ISO-8859-1, etc. No es obligatorio salvo que sea un documento externo a otro principal.
- *standalone*: indica si el documento va acompañado de un DTD ("no"), o no lo necesita ("yes"); en principio no hay por qué ponerlo, porque luego se indica el DTD si se necesita.

En cuanto a la sintaxis del documento, algunos detalles son:

- Los documentos XML son sensibles a mayúsculas, esto es, en ellos se diferencian las mayúsculas de las minúsculas.
- Los espacios y retornos de carro se tienen en cuenta.
- Hay algunos caracteres especiales reservados, que forman parte de la sintaxis de XML: <, >, &, " y '. En su lugar, cuando queramos representarlos deberemos usar las entidades <, >, &, " y ' respectivamente.
- Los valores de los atributos de todas las etiquetas deben ir siempre entrecomillados. Son válidas las dobles comillas (") y la comilla simple (').

Pasando al contenido en sí, es importante diferenciar entre elementos y etiquetas: los elementos son las entidades en sí, lo que tiene contenido, mientras que las etiquetas sólo describen a los elementos. Un documento XML está compuesto por elementos, y en su sintaxis éstos se nombran mediante etiquetas.

Hay dos tipos de elementos: los vacíos y los no vacíos:

- Los elementos vacíos son aquellos que no tienen contenido dentro del documento. La sintaxis correcta para estos elementos implica que la etiqueta tenga siempre esta forma: <etiqueta/>.
- Toda etiqueta no vacía debe tener una etiqueta de cerrado.
- Todos los elementos deben estar perfectamente anidados.

Todos los documentos XML requieren un elemento raíz para ser bien formados. El elemento raíz *(o etiqueta del documento)* debe estar precedido por el prólogo (declaración XML + DTD si lo hubiera), incluir el documento por completo y emparejarse con la declaración del DTD (para ser válido).

Las etiquetas deben cumplir:

- Deben comenzar por letra o guión bajo (_).
- Seguido de alguna combinación de las anteriores y/o números, puntos (.), dos puntos (:) o guiones (-).
- Sin espacios en blanco.
- Sin el prefijo xml/XML/XmL, ... , que está reservado.
- Sin longitud máxima.

Las etiquetas pueden contener atributos, que deben cumplir:

- Definidos en la etiqueta de comienzo del elemento.
- Especifican parejas *nombre="valor"*.
- No deben estar duplicados en la misma etiqueta.

- *Nombre:* con mismas reglas que para el nombre de las etiquetas.
- *Valor:* con comillas obligatorias (en HTML/SGML son optativas).

Las secciones *CDATA*:

```
<![CDATA[
        Aqui puedo poner lo que quiera.
] ]>
```

empiezan con los caracteres "<![CDATA[" y termina con "]]>". Dentro de ellos podemos colocar cualquier cosa ya que no va a ser interpretado, con la salvedad de la cadena que indica el final de CDATA, "]]>", ya que el procesador al encontrársela entendería que la sección CDATA ya ha terminado.

Se permiten *comentarios* a modo informativo para el programador que han de ser ignorados por el procesador. Los comentarios en XML tienen el siguiente formato: <!--- Esto es un comentario --->, igual que en HTML. Los comentarios no pueden estar dentro de una etiqueta, no se pueden anidar, no pueden formar parte de las declaraciones de las entidades y no pueden preceder a la declaración de XML.

Otros elementos de XML son las **Entidades** definidas en los DTD, que son macros o alias de trozos de texto. Pueden ser:

- Entidades generales *(general entities)*
  - Se usan en los documentos XML con un ampersand &
  - &entidad;
- Entidades de parámetros *(parameter entities)*
  - Se usan en los DTD con un tanto por ciento %
  - %entidad;

Por último, detallamos más en profundidad los DTDs. Los **DTD** no son más que definiciones de los elementos que puede incluir un documento XML, de la forma en que deben hacerlo (qué elementos van dentro de otros) y los atributos que se les puede dar.

Hay varios modos de referenciar un DTD en un documento XML:

- Incluir dentro del documento una referencia al DTD en forma de URI:

  <!DOCTYPE ficha SYSTEM "http://www..../ficha.dtd">

  La palabra SYSTEM indica que el DTD se obtendrá a partir de un elemento externo al documento e indicado por el URI que lo sigue entrecomillado.

- O bien incluir dentro del propio documento el DTD directamente, añadiendo la declaración <!DOCTYPE y después del nombre del tipo de documento, en vez de la URI del DTD, el propio DTD entre los símbolos '[' y ']'.

En cuanto a la definición de los elementos, es bastante intuitiva: después de la cláusula <!ELEMENT se incluye el nombre del elemento (el que luego se indicará en la etiqueta), y después diferentes cosas en función del elemento:

- entre paréntesis, si el elemento es no vacío, se indica el contenido que puede tener el elemento: la lista de elementos hijos o que descienden de él si los tiene, separados por comas; o el tipo de contenido, normalmente #PCDATA, que indica datos de tipo texto, que son los más habituales.
- si es un elemento vacío, se indica con la palabra EMPTY.

A la hora de indicar los elementos descendientes (los que están entre paréntesis) vemos que van seguidos de unos caracteres especiales: '+', '*', '?' y '|':

- + : uso obligatorio y múltiple; permite uno o más elementos de ese tipo dentro del elemento padre, pero como mínimo uno.
- * : opcional y múltiple; puede no haber ninguna ocurrencia, una o varias.
- ? : opcional pero singular; puede no haber ninguno o como mucho uno.
- | : equivale a un OR, es decir, da la opción de usar un elemento de entre los que forman la expresión, y sólo uno.

Para la definición de los *atributos* se usa la declaración <!ATTLIST seguida de:

- el *nombre de elemento* del que estamos declarando los atributos;
- el *nombre del atributo*;
- los posibles *valores del atributo*, entre paréntesis y separados por el carácter |, que al igual que para los elementos, significa que el atributo puede tener uno y sólo uno de los valores incluidos entre paréntesis. O bien, si no hay valores definidos, se escribe CDATA para indicar que puede ser cualquier valor. También podemos indicar con la declaración ID que el valor alfanumérico que se le dé será único en el documento, y se podrá referenciar ese elemento a través de ese atributo y valor;
- de forma opcional y entrecomillado, un valor por defecto del atributo si no se incluye otro en la declaración;
- por último, si es obligatorio cada vez que se usa el elemento en cuestión declarar este atributo, es necesario declararlo con la cláusula #REQUIRED; si no lo es, se debe poner #IMPLIED, o bien #FIXED si el valor de dicho atributo se debe mantener fijo a lo largo de todo el documento para todos los elementos del mismo tipo.

Por último, podemos ver un ejemplo de un documento XML muy sencillo:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ficha>
<nombre>Angel</nombre>
<apellido>Barbero</apellido>
<direccion>c/Ulises, 36</direccion>
</ficha>
```

Y un ejemplo de su DTD asociado:

```
<!ELEMENT ficha (nombre+, apellido+, direccion+, foto?)>
<!ELEMENT nombre (#PCDATA)>
<!ATTLIST nombre sexo (masculino|femenino) #IMPLIED>
<!ELEMENT apellido (#PCDATA)>
```

```
<!ELEMENT direccion (#PCDATA)>
<!ELEMENT foto EMPTY>
```

2.1.4.2. XML Schemas

Como ya se explicó en un apartado anterior, XML Schema es un lenguaje para definir la estructura de los documentos XML. Un esquema es un conjunto de reglas que un documento XML debe cumplir para poder ser considerado válido de acuerdo con ese esquema.

Un ejemplo de un XML Schema puede ser:

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.w3schools.com"
xmlns="http://www.w3schools.com"
elementFormDefault="qualified">

<xs:element name="note">
   <xs:complexType>
     <xs:sequence>
         <xs:element name="to" type="xs:string"/>
         <xs:element name="from" type="xs:string"/>
         <xs:element name="heading" type="xs:string"/>
         <xs:element name="body" type="xs:string"/>
     </xs:sequence>
   </xs:complexType>
</xs:element>

</xs:schema>
```

2.1.4.3. RDF

En 1999 se publicó la primera versión de RDF (Resource Description Framework - Infraestructura para la Descripción de Recursos), un lenguaje para la definición de ontologías y metadatos en la Web. RDF es el estándar más popular y extendido en la comunidad de la Web semántica. Es un modelo de datos para los recursos en la Web. Aporta una semántica básica para este modelo de datos que puede representarse mediante XML. Su sintaxis, diseñada para ser comprensible por máquinas, está basada en expresiones de la forma sujeto-predicado-objeto.

El modelo RDF de datos básico consiste en tres tipos de objetos:

Recursos	Todas las cosas descritas por expresiones RDF se denominan *recursos*. Un recurso puede ser una página Web completa; tal como el documento HTML http://www.w3.org/Overview.html, por ejemplo. Un recurso puede ser una parte de una página Web; p.ej. un elemento HTML o XML específico dentro del documento fuente. Un recurso puede ser también una colección completa de páginas; p.ej. un sitio Web completo. Un recurso puede ser también un objeto que no sea directamente accesible vía Web, p.ej. un libro impreso. Los recursos se designan siempre por URIs más identificadores opcionales. Cualquier cosa puede tener un URI; la extensibilidad de URIs permite la introducción de identificadores para cualquier entidad imaginable.
Propiedades	Una *propiedad* es un aspecto específico, característica, atributo, o relación utilizado para describir un recurso. Cada propiedad tiene un significado específico, define sus valores permitidos, los tipos de recursos que puede describir, y sus relaciones con otras propiedades.
Sentencias [declaraciones, enunciados]	Un recurso específico junto con una propiedad denominada, más el valor de dicha propiedad para ese recurso es una *sentencia RDF*. Estas tres partes individuales de una sentencia se denominan, respectivamente, sujeto, predicado y objeto. El objeto de una sentencia (es decir, el valor de la propiedad) puede ser otro recurso o puede ser un literal; es decir, un recurso (especificado por un URI) o una cadena simple de caracteres u otros tipos de datos primitivos definidos por XML. En términos RDF, un *literal* puede comprender en su contenido marcado XML pero ya no puede valorarse más por un procesador RDF.

Tabla 1. Objetos del modelo RDF de datos básicos

Este modelo se basa en la idea de convertir las declaraciones de los recursos en expresiones con la forma sujeto-predicado-objeto (conocidas en términos RDF como tripletes). El *sujeto* es el recurso, es decir, es lo que se está describiendo. El *predicado* es la propiedad o relación que se desea establecer acerca del recurso. Por último, el *objeto* es el valor de la propiedad o el otro recurso con el que se establece la relación.

RDF necesita una sintaxis para crear e intercambiar metadatos. Utiliza XML como su sintaxis de intercambio. Se definen dos sintaxis XML para codificar una instancia de modelo de datos. La *sintaxis serializada* expresa las capacidades totales del modelo de datos de una forma muy regular. La *sintaxis abreviada* incluye términos adicionales que proporcionan una forma más compacta para representar un subconjunto del modelo de datos.

La combinación de RDF con otras herramientas como RDF Schema y OWL permite añadir significado a las páginas, y es una de las tecnologías esenciales de la Web semántica.

Un ejemplo de RDF es:

```
<rdf:RDF>
  <rdf:Description about="http://www.w3.org/Home/Lassila">
    <s:Creator>Ora Lassila</s:Creator>
  </rdf:Description>
</rdf:RDF>
```

2.1.4.4. RDF Schemas

RDF Schema es un vocabulario para describir las propiedades y las clases de los recursos RDF, con una semántica para establecer jerarquías de generalización entre dichas propiedades y clases.

Con RDF Schema (RDFS) se pueden definir jerarquías de clases de recursos, especificando las propiedades y relaciones que se admiten entre ellas.

El modelo proporciona una sintaxis que opera con unos objetos genéricos vacíos de semántica. El modelo no proporciona ningún mecanismo para declarar las propiedades ni definir las relaciones entre propiedades y recursos. El esquema es el que se encarga de ello. El esquema RDF proporciona información sobre la interpretación de las sentencias de RDF. Es una colección de recursos que se pueden utilizar para describir propiedades de otros recursos que definen vocabularios específicos de una aplicación. El esquema define una jerarquía de clases, propiedades, con restricciones sobre las propiedades como dominio, rango de aplicación, etc.

2.1.4.5. OWL

A RDF le siguieron OIL (Ontology Inference Language), desarrollado en Europa, y DAML (DARPA Agent Markup Language), en EE.UU., dos lenguajes muy similares que de hecho se terminaron fundiendo en DAML+OIL. A partir de esta unión se definió el lenguaje OWL (Web Ontology Language), con el propósito de reunir todas las ventajas de DAML+OIL y resolver los problemas de este lenguaje. OWL se puede formular en RDF, por lo que se suele considerar una extensión de éste. OWL incluye toda la capacidad expresiva de RDF y la extiende con la posibilidad de utilizar expresiones lógicas.

El Lenguaje de Ontologías Web (OWL) está diseñado para ser usado en aplicaciones que necesitan procesar el contenido de la información en lugar de únicamente representar información para los humanos. OWL facilita un mejor mecanismo de interpretabilidad de contenido Web que los mecanismos admitidos por XML, RDF, y esquema RDF (RDF-S) proporcionando vocabulario adicional junto con una semántica formal. OWL puede ser usado para representar explícitamente el significado de términos en vocabularios y las relaciones entre esos términos, es decir, una ontología.

Su extensión con respecto a RDF es:

• Capacidad para limitar las propiedades de clases con respecto a número y tipo.
• Capacidad para inferir qué elementos que tienen varias propiedades son miembros de una clase en particular.
• Capacidad para determinar si todos los miembros de una clase tendrán una propiedad en particular, o si puede ser que sólo algunos la tengan.
• Capacidad para distinguir entre relaciones uno a-uno, varios-a-uno o uno-a-varios, permitiendo que las "claves externas" de las bases de datos puedan representarse en una ontología.
• Capacidad para expresar relaciones entre clases definidas en diferentes documentos en la Web.
• Capacidad para construir nuevas clases a partir de uniones, intersecciones y complementos de otras.
• Capacidad para restringir rangos y dominios para especificar combinaciones de clases y propiedades.

OWL proporciona tres lenguajes, cada uno con nivel de expresividad mayor que el anterior:

- ***OWL Lite*** está diseñado para aquellos usuarios que necesitan principalmente una clasificación jerárquica y restricciones simples. OWL Lite tiene una menor complejidad formal que OWL DL.

- **OWL DL** está diseñado para aquellos usuarios que quieren la máxima expresividad conservando completitud computacional (se garantiza que todas las conclusiones sean computables), y resolubilidad (todos los cálculos se resolverán en un tiempo finito). OWL DL incluye todas las construcciones del lenguaje de OWL, pero sólo pueden ser usadas bajo ciertas restricciones. OWL DL es denominado de esta forma debido a su correspondencia con la *lógica de descripción* (**D**escription **L**ogics, en inglés), un campo de investigación que estudia la lógica que compone la base formal de OWL.

- ***OWL Full*** está dirigido a usuarios que quieren máxima expresividad y libertad sintáctica de RDF sin garantías computacionales. OWL Full permite una ontología para aumentar el significado del vocabulario preestablecido (RDF o OWL).

Cada uno de estos sublenguajes es una extensión de su predecesor más simple, respecto a lo que puede ser expresado legamente y a la validación de sus conclusiones. El siguiente grupo de relaciones se mantienen, pero las relaciones inversas no se mantienen:

- Cada ontología legal de OWL Lite es una ontología legal de OWL DL.
- Cada ontología legal de OWL DL es una ontología legal de OWL Full.
- Cada conclusión válida de OWL Lite es una conclusión válida de OWL DL.
- Cada conclusión válida de OWL DL es una conclusión válida de OWL Full.

La elección entre OWL Lite y OWL DL depende de las necesidades de los usuarios sobre la expresividad de las construcciones, proporcionando OWL DL las más

expresivas. La elección entre OWL DL y OWL Full depende principalmente de las necesidades de los usuarios sobre los recursos de metamodelado del esquema RDF. OWL Lite utiliza únicamente algunas de las características del lenguaje OWL y está más limitado en el uso de características que OWL DL y OWL Full.

OWL Full puede ser considerada como una extensión de RDF, mientras que OWL Lite y OWL DL pueden ser consideradas como extensiones de una visión restringida de RDF. Cada documento OWL (Lite, DL, Full) es un documento RDF, y cada documento RDF es un documento de OWL Full, pero sólo algunos documentos RDF serán legalmente documentos OWL Lite o OWL DL. Por este motivo, se ha de tener cuidado cuando un usuario quiera migrar un documento de RDF a OWL.

2.1.5. Mapa Conceptual de la Web Semántica

Lo visto en los puntos anteriores acerca de la Web Semántica lo podríamos resumir en el siguiente mapa conceptual:

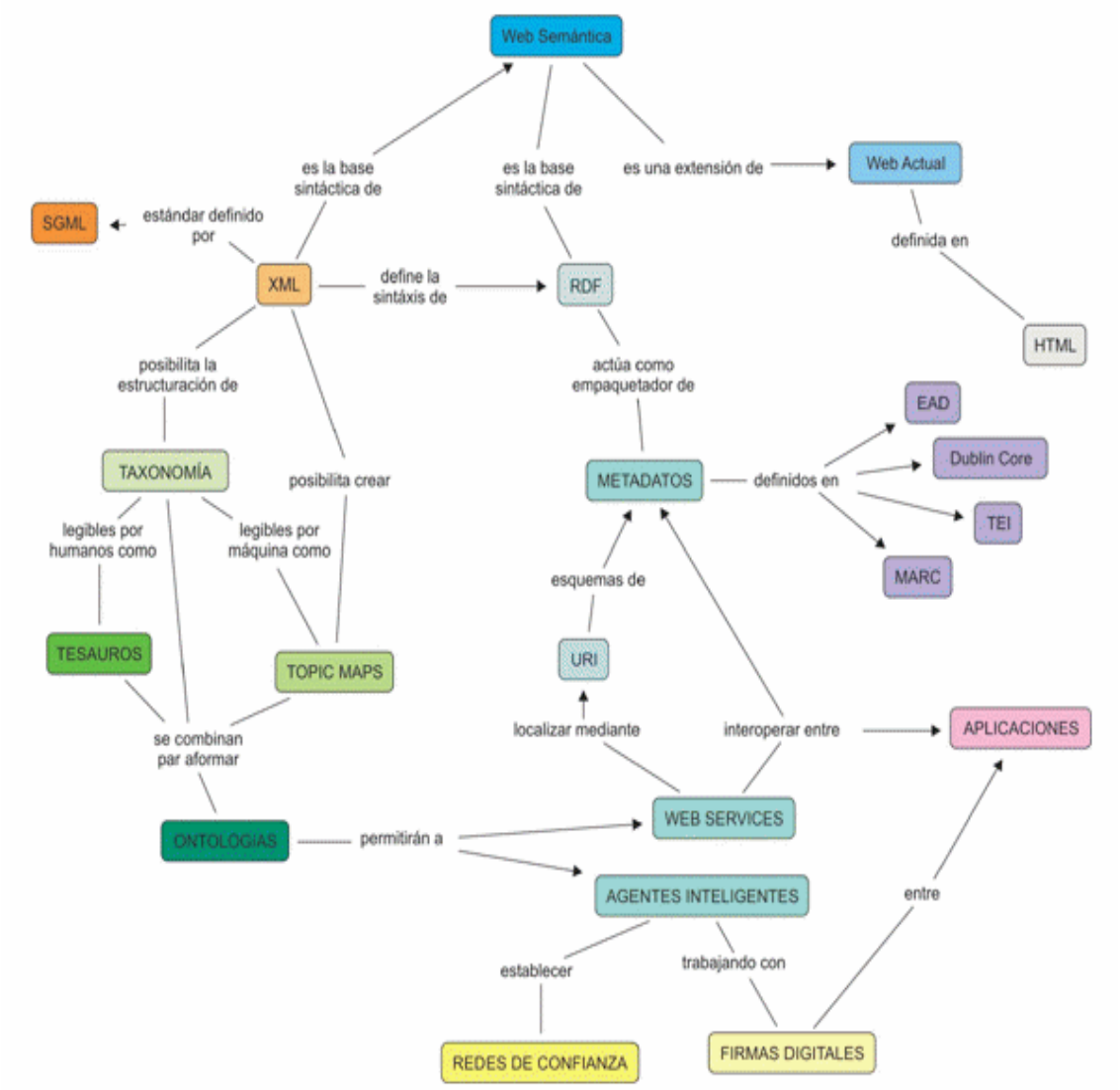


Figura 2. Mapa Conceptual de la Web Semántica

La Web Semántica, basada en XML, es una extensión de la Web Actual, definida en HTML. Mediante XML se pueden construir ontologías y definir el lenguaje RDF, elementos base para la Web Semántica. Las ontologías permiten localizar metadatos que pueden ser empaquetados por RDF. Los metadatos son datos con marcas que le dan el significado a la Web Semántica.

2.1.6. Ontologías

El término ontología proviene de la filosofía; pero en Inteligencia Artificial, tiene diferentes connotaciones. La definición declarativa más consolidada es la propuesta por Gruber [Gruber 1993] y extendida por Studer [Studer 1998] y colegas que la describe como "una especificación explícita y formal sobre una conceptualización compartida". La interpretación de esta definición es que las ontologías definen conceptos y relaciones de algún dominio, de forma compartida y consensuada; y que esta conceptualización debe ser representada de una manera formal, legible y utilizable por los ordenadores.

Las ontologías tienen los siguientes componentes que servirán para representar el conocimiento de algún dominio [Gruber 1993]:

- **Conceptos**: son las ideas básicas que se intentan formalizar. Los conceptos pueden ser clases de objetos, métodos, planes, estrategias, procesos de razonamiento, etc.
- **Relaciones**: representan la interacción y enlace entre los conceptos del dominio. Suelen formar la taxonomía del dominio.
- **Funciones**: son un tipo concreto de relación donde se identifica un elemento mediante el cálculo de una función que considera varios elementos de la ontología.
- **Instancias**: se utilizan para representar objetos determinados de un concepto.
- **Axiomas**: son teoremas que se declaran sobre relaciones que deben cumplir los elementos de la ontología.

Las herramientas de creación de ontologías (editores de ontologías) son las herramientas que permiten la codificación de una determinada ontología utilizando un determinado lenguaje. Este tipo de herramientas permiten definir la estructura sobre la cual se clasificará la información realizada a través de anotaciones. Algunas herramientas de este tipo son *Apollo, LinkFactory, OILEd, OntoEdit, Ontolingua Server, OntoSaurus, OpenKnoME, Protégé-2000, SymOntoX, WebODE, WebOnto*.

2.2. MINERÍA WEB o WEB MINING

Uno de los campos más activos de investigación en la Web Semántica es el de la extracción de información Web. De esto precisamente se encarga la Minería Web o Web Mining.

Algunos autores definen a la Web Mining como el uso de técnicas para descubrir y extraer de forma automática información de los documentos y servicios de la Web. Según M. Scotto [Scotto 2004], la Web Mining es el proceso de descubrir y analizar información "útil" de los documentos de la Web. También se puede definir como el descubrimiento y análisis de información relevante que involucra el uso de

técnicas y aproximaciones basadas en la minería de datos (*Data Mining*) orientados al descubrimiento y extracción automática de información de documentos y servicios de la Web, teniendo en consideración el comportamiento y preferencias del usuario.

2.2.1. Fases de la Minería Web

De acuerdo con Etzioni [Etzioni 1996], el proceso general de la Web Mining se compone de cuatro tareas:

- **Recuperación de la información (RI)** (Descubrimiento de Recursos)

La RI trata de la recuperación automática de todos los documentos relevantes disponibles en la Web. El proceso de RI principalmente incluye la representación de documentos, indexación, y búsqueda de documentos. Existen cuatro clases de indexación de documentos Web, que son indexación humana o manual, indexación automática, indexación inteligente o basada en agentes, y la indexación basada en metadatos. Los motores de búsqueda son programas escritos para consultar y recuperar información guardada en las bases de datos (Estructuradas), páginas HTML (Semiestructuradas), y el texto libre (Sin Estructurar) en la Web.

Las siguientes dificultades pueden encontrarse durante la ejecución de esta tarea:

- Subjetividad, imprecisión e incertidumbre en las consultas del usuario.
- Los motores de búsqueda actuales no tienen la capacidad deductiva.
- Las técnicas de procesamiento de consultas actuales deberían determinar la relevancia de un documento consultado con respecto a una consulta.
- Resulta difícil examinar una lista entera de documentos que retorna el motor de la búsqueda como repuesta a la consulta.
- Se deberían ajustar los documentos según la naturaleza de los usuarios.
- Dinamismo, escalabilidad, y heterogeneidad de los documentos Web.

- **Extracción de la Información (EI)** (Extracción y Pre-procesamiento)

Tiene como objetivo transformar los documentos extraídos en el proceso de recuperación de información en documentos que sean más digeribles, fáciles de leer y de analizar. La extracción de información (EI) se refiere a la tarea de identificar fragmentos específicos de un documento. La EI apunta a extraer nuevo conocimiento de los documentos recuperados centrándose en la estructura del documento y en la representación del documento. Es necesario un sistema de Pre-procesamiento robusto para extraer buena información. Los métodos principales de EI involucran los ”Wrappers”, que se verán en el apartado 4. Los Wrapper son procedimientos para extraer información particular de los recursos de la Web.

Durante la ejecución de esta tarea se puede dar el siguiente problema:

- Un Wrapper es un sistema de EI personalizado para un sitio particular y no es universalmente aplicable.
- No es posible construir sistemas de EI que sean escalables al tamaño y dinamismo de la Web.

- **Generalización**

Reconocimiento de Patrones generales de una página en particular o bien también patrones de diferentes páginas. En esta fase, se usa normalmente reconocimiento de patrones y técnicas de aprendizaje automático para la información extraída.

Las siguientes dificultades pueden darse durante la ejecución de esta tarea:

- Existen datos abundantes en la Web sin etiquetar.
- La comunidad de RI ha explorado la agrupación de documentos, pero la agrupación todavía no es utilizada en los principales motores de búsqueda.
- Los datos son multidimensionales y solapados.
- Outliers (datos fuera de rango) de los usuarios y de sus transacciones.
- Reglas de asociación lingüísticas que son más entendibles por los humanos.

- **Análisis**

El análisis es un problema de manipulación de datos que asume que hay suficientes datos disponibles para que la información potencialmente útil pueda extraerse y analizarse. Los humanos juegan un papel importante en el proceso de descubrimiento de conocimiento de la información de la Web, ya que la Web es un medio interactivo. Esto es especialmente importante para la aprobación y/o interpretación de los patrones encontrados, ya que necesitan de la aprobación humana. Una vez se han descubierto los modelos, los analistas necesitan las herramientas apropiadas para entender, visualizar, e interpretar este conocimiento.

El problema que se enfrenta en este punto es el de mostrar y modelar el conocimiento descubierto. El interpretar el conocimiento descubierto de la información disponible en la Web siempre ha sido un desafío para los analistas, como el rendimiento de los algoritmos de minería de datos. Esto es porque los patrones descubiertos están principalmente en forma matemática.

Basándose en las fases mencionadas, la Minería Web puede verse como el uso de técnicas de minería de datos para recuperación automática, extracción y evaluación de la información para descubrimiento de conocimiento en los documentos y servicios de la Web. La evaluación incluye la generalización y el análisis.

2.2.2. Clases de Minería Web

En materia de minería Web, existen tres clases fundamentales: Minería de Contenido Web (WCM), Minería de Estructura Web (WSM), y Minería de Usuario Web (WUM).

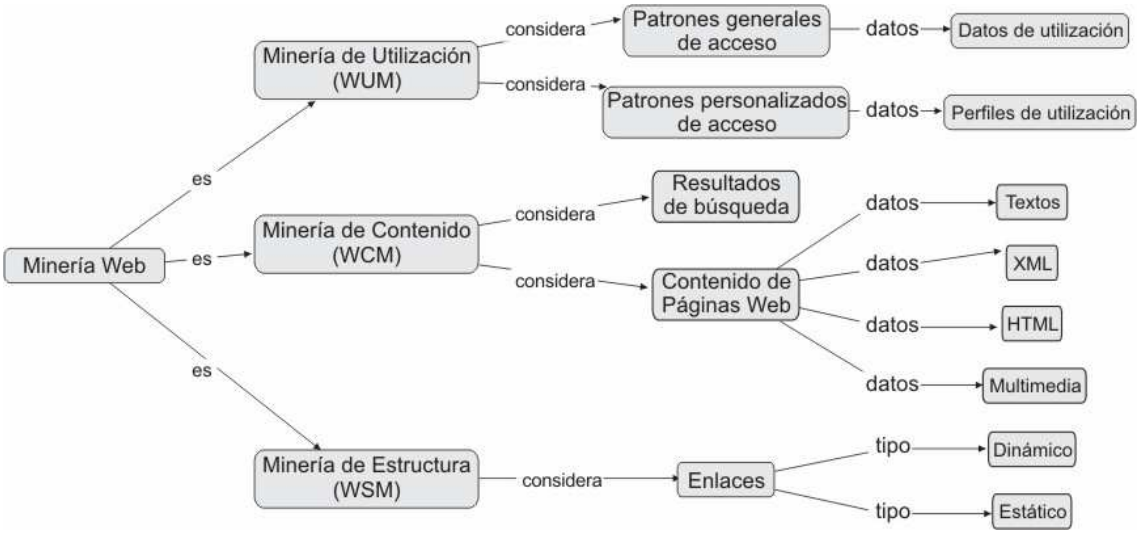


Figura 3. Clases de Minería Web

Minería de Contenido Web (WCM)

Se trata del descubrimiento de información útil de contenido, datos, documentos, servicios de la Web, etc. Sin embargo, los contenidos de la Web no sólo son el texto, éstos pueden abarcar datos como audio, vídeo, símbolos, metadatos e hipervínculos.

Debe notarse que la Minería de Contenido Web y la Recuperación de Información están separadas por una delgada línea. Se puede afirmar que la Recuperación de Información en la Web es un caso de la Minería de Contenido Web.

Hay dos estrategias para WCM: Aquéllos que minan directamente el contenido de los documentos (Web Page Content Mining) y aquéllos que mejoran el contenido de la búsqueda de otras herramientas, como los motores de la búsqueda (Search Result Mining). La tarea de los motores de búsqueda está muy relacionada con WCM.

Minería de Estructura Web (WSM)

La minería de estructura pretende revelar la estructura real de un sitio Web a través de la recogida de datos referentes a su estructura y, principalmente, a su conectividad. Típicamente tiene en cuenta dos tipos de enlaces: estáticos y dinámicos.

Trata de revelar como están relacionados los hipervínculos entre las distintas páginas para generar un informe estructural sobre la página y el sitio Web. La minería de estructura Web, además nos proporciona información acerca de si los usuarios encuentran la información, si la estructura del sitio es demasiado ancha (mucho contenido en una misma página) o demasiado profunda (muchos enlaces que estructuran el mismo contenido), si los elementos están colocados en los lugares adecuados dentro de la página, si la navegación se entiende, cuáles son las secciones menos visitadas y su relación con el lugar que ocupan en la página principal, etc.

La herramienta para realizar la WSM es la utilización de grafos, la cual nos permite reflejar el movimiento entre enlaces al navegar de una página a otra y así tener una mejor visión del conocimiento obtenido.

Minería de Uso Web (WUM)

La minería de uso intenta dar sentido a los datos y comportamientos generados en las sesiones de navegación Web. Es decir, son aquellos datos que describen el uso al cual se ve sometido un sitio, registrado en los logs (registros de sucesos/eventos) de acceso de los servidores Web. A partir de esta información se podría concluir, por ejemplo, qué documento visitado no tiene razón de ser, o si una página no se encuentra en los primeros niveles de jerarquía de un sitio. Analizar los logs de diferentes servidores Web puede ayudar a entender el comportamiento del usuario o la estructura de la Web, permitiendo de este modo mejorar el diseño de esta colección de recursos.

Este tipo de minería tiene dos objetivos principales:

- Extraer patrones generales de uso de un sitio Web de manera que pueda reestructurarse para que sea más fácil de utilizar y mejore el acceso por parte de los usuarios.
- Obtener perfiles de los distintos tipos de usuarios a partir de su comportamiento y navegación, para ofrecer una atención más personalizada.

Ya hemos visto en qué consiste la Minería Web, y para llevarla a cabo necesitamos herramientas que permitan la extracción de la información contenida en las páginas Web de forma automática y con la mayor precisión posible. Esto lo podemos realizar por medio de los Wrappers.

2.3. WRAPPERS

Los Wrappers extraen información de páginas Web (u otras fuentes semiestructuradas) escritas en un formato específico. Un Wrapper sobre fuentes Web es considerado un software que acepta consultas de usuarios de datos en la Web y después de extraer la información relevante retorna los resultados [Araque 2000].

En ocasiones el Wrapper se encarga también de monitorizar las fuentes de datos para detectar posibles cambios en los datos e informar de estos cambios al módulo de integración, siendo este último componente el responsable de integrar los datos.

Algunas de las ventajas de construir Wrappers en Web se pueden resumir en:

- Consultas en las fuentes similares a las realizadas sobre bases de datos.
- Todas las fuentes sobre la que se construyó el Wrapper pueden ser procesadas usando un lenguaje común de consulta.
- Este acceso integrado se puede hacer usando un mediador que integra la información desde las diversas fuentes.

Pero también es necesario reconocer que son imprácticos en casos donde el número de fuentes de interés es muy grande, nuevas fuentes son añadidas frecuentemente o se dan cambios usuales en el formato de fuentes.

2.3.1. Sistema Mediador-Wrapper

Extendiendo la idea de las bases de datos, se puede proponer una arquitectura de mediador-Wrappers. Cada mediador maneja un esquema de datos común capaz de representar los datos ofrecidos por cada fuente de datos. Por cada fuente de datos existe un Wrapper que conoce el esquema de datos de su correspondiente fuente de datos y traduce los datos al esquema de datos existente en el mediador.

Un mediador determina las fuentes a ser consultadas para responder una consulta y las subconsultas a ser enviadas a cada fuente. El Wrapper optimiza las consultas a ser enviadas a una fuente y traduce las enviadas por el mediador hacia su fuente de datos.

Este sistema permite que los Wrapper extraigan la información de fuentes heterogéneas y se integre a través de los mediadores. Los mediadores pueden ser bases de datos convencionales o bases de conocimiento representadas con datos estructurados extraídos desde fuentes semi o no estructurados [Inmon 1993]. Las consultas de los usuarios se aplican sobre estos almacenes de datos, en lugar de ir directamente a la Web.

2.3.2. Construcción de Wrappers

La construcción de un Wrapper incluye:

- Identificar la estructura de las fuentes, identificando secciones y subsecciones de interés.
- Construir un parser o analizador para que trabaje sobre las páginas fuentes.
- Definir la comunicación entre Wrapper, mediador y fuentes Web.

Cada fase se compone de ciertas tareas:

- Identificar la estructura de las fuentes, identificando secciones y subsecciones de interés. La tarea de estructuración puede ser automática o con mínima interacción con el usuario a través de programas que analizan el documento HTML y luego otros que formatean la información con base en tokens (símbolos) y/o jerarquías. A la vez implica dos pasos:
    - Identificar los tokens (símbolos que indican o diferencian secciones) de interés en la página. Se usan analizadores léxicos que permiten obtener expresiones regulares en los documentos.
    - Determinar las jerarquías dentro de las secciones. Generalmente se usan heurísticas basadas en el tamaño de la letra o la indentación de los documentos.

- Construir un parser o analizador para que trabaje sobre las páginas fuentes. A partir de los pasos anteriores y usando un analizador léxico y un generador de compilación, se leen las fuentes y se descubre la estructura.

- Definir la comunicación entre Wrapper, mediador y fuentes Web. Varios elementos se involucran en este paso:

- Determinar las URLs de las páginas de las cuales se va a extraer información, generalmente partiendo de la página inicio (index).
- Garantizar la capacidad de recuperar páginas Internet; generalmente con programas o script en Java, Perl, etc.
- Comunicación entre Wrapper y mediador, usando un lenguaje de comunicación entre agentes como KQML u otro.

2.3.3. Generadores de Wrappers

La clasificación propuesta [Laender 2002] se basa en la tecnología que se utiliza para la obtención de los Wrappers y divide las herramientas en:

• *Lenguajes para el diseño automático*: Lenguajes diseñados de manera específica para realizar tanto la descarga como la extracción de partes de información. Surgieron como alternativa a los de propósito general que ya se utilizaban en este aspecto. Algunos ejemplos son: *Minerva, Tsimmis, WebOQL, WebL*.

• *Generadores basados en la estructura HTML*: Herramientas o proyectos que se basan en el uso de expresiones basadas en la estructura del documento HTML para localizar las partes de interés en cada archivo. Algunos ejemplos son: *W4F, XWRAP, Road Runner, LIXTO, Solvent + Piggy Bank.*

• *Generadores por inducción*: Generan los Wrappers aprendiendo la forma de hacerlo a través de varios ejemplos de entrenamiento. A partir de las reglas obtenidas se infiere la estructura del documento y las partes de interés. Algunos de estos generadores necesitan de reglas especificadas por un usuario mientras que otros pueden conseguirlo de manera automática. Algunos ejemplos son: *ShopBot, WIEN, SoftMealy, STALKER*.

• *Generadores basados en el Modelo*: Herramientas que realizan búsquedas de una estructura de referencia en el interior de cada documento analizado. Cuando se encuentran correspondencias totales o parciales, se extraen los datos, en un proceso inverso a las herramientas de inducción, en las que se infería la estructura a base de ejemplos. Algunos ejemplos son: *NoDoSe, DEByE*.

• *Generadores basados en ontologías*: Herramientas de extracción que, en lugar de basarse en la estructura de presentación del documento HTML, se centra en la estructura de los propios datos, dando menos importancia a etiquetas como <table> o <tr>. Dado un modelo de aplicación, puede utilizarse una ontología para localizar constantes presentes en la página y construir objetos con ella. Algunos ejemplos son: *Brigham Young University Data Extraction Group, RDF Web Scraper, DDC/RDFEditor*.

• *Generadores basados en expresiones regulares*: Generadores que no se basan en la estructura HTML. El principal exponente es Caméléon, que es una de las herramientas que ha sido utilizada con mayor éxito en proyectos de éstas características.

3. IMPLEMENTACIÓN DEL PROYECTO

3.1. Introducción

Explicados los fundamentos teóricos relacionados con la Web Semántica, la Minería Web y los Wrappers, ya poseo los conocimientos necesarios para aplicarlos en mi proyecto. Estos conocimientos teóricos no sólo me sirven a mí para poder realizar mi proyecto y aplicarlos en la implementación del mismo, sino que también ayudan a cualquier persona que lea este proyecto a comprender todo lo realizado, el contexto en el que nos movemos y el programa desarrollado para implementar la aplicación.

En este apartado de implementación del proyecto se explica con detalle el trabajo realizado. Empezando por el marco en el que se ubica el proyecto, se da toda la información necesaria de la aplicación implementada, los módulos que la componen, su programación, los resultados obtenidos, y otros aspectos, como son los problemas encontrados, o las soluciones y decisiones tomadas. Todo esto se hace buscando el equilibrio entre la mayor facilidad y la mejor implementación posible, porque como ya se sabe, en temas de programación es complicado encontrar la mejor implementación con la menor dificultad del programa.

3.2. Marco de ubicación del proyecto

El presente Proyecto Fin de Carrera se encuentra enmarcado dentro de un sistema de planificación de visitas turísticas a una ciudad para una PDA. El sistema está formado por una serie de módulos, de los cuales, este Proyecto Fin de Carrera implementa el *módulo de extracción de la información de tiempos de desplazamiento de Internet* y el *módulo del servidor* de dicho módulo.

Un posible esquema del sistema se puede ver en la figura 4:

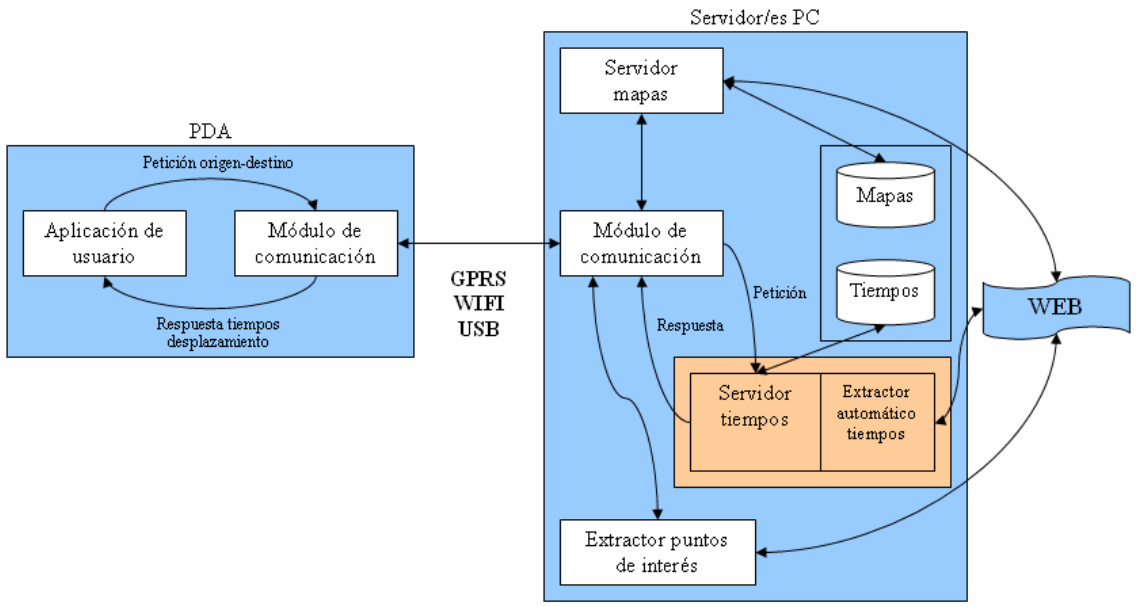


Figura 4. Esquema del sistema

A continuación hacemos una breve descripción de todos los módulos que componen el sistema:

- Aplicación de Usuario: Proporcionará la interfaz de usuario en la PDA y hará uso de la información que solicite y reciba de los demás módulos (tiempos de desplazamiento, mapas, puntos de interés, etc.).

- Módulo de Comunicación: Es el encargado de la transmisión de datos entre la PDA y los servidores de información. Tanto el servidor de mapas, como el servidor-extractor de tiempos de desplazamiento, como el extractor de puntos de interés deberán comunicarse con él para proporcionar la información al módulo de comunicación de la PDA.

- Servidor de Mapas: Es el encargado de gestionar los mapas y generarlos para la PDA. Necesitará acceder a la base de datos de mapas, donde éstos están guardados. Para comunicarse con la PDA deberá contactar primero con el módulo de comunicación, y éste será el que se comunique con el módulo de comunicación de la PDA.

- Servidor/Extractor automático de tiempos: Éste es el módulo desarrollado para este Proyecto Fin de Carrera. Utiliza métodos de extracción automática de información que se irán explicando a lo largo del documento para generar una base de datos de tiempos de desplazamiento entre dos puntos de una ciudad. Una vez conseguida la información y guardada, realizará procesos de comunicación con el módulo correspondiente para la transmisión de los datos.

- Extractor de puntos de interés: Es el encargado de extraer información de los puntos de interés de una ciudad, de manera que posteriormente se puedan extraer los tiempos de desplazamiento entre ellos utilizando el Servidor/Extractor automático de tiempos que se implementa en este Proyecto Fin de Carrera.

3.3. Resumen de la aplicación

Antes de meternos en la programación pura y dura, mostramos en la figura 5 un pequeño resumen de la aplicación que se quiere implementar:

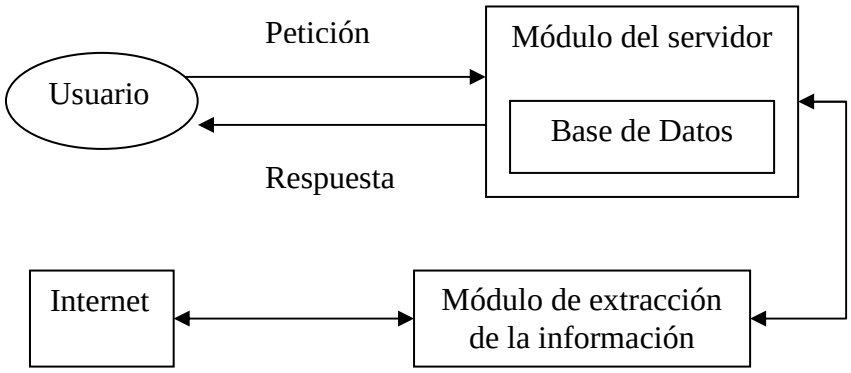


Figura 5. Esquema de la aplicación implementada

Como podemos ver, la aplicación está formada principalmente por dos módulos: Módulo del servidor y Módulo de extracción de la información.

El usuario es quien realiza peticiones a la aplicación, que tras pasar por otros módulos, llega al módulo del servidor. Éste módulo es el que posteriormente creará y enviará la respuesta al usuario con los datos de tiempo solicitados. Tanto la petición como la respuesta son ficheros XML, cuyos formatos se explican con más detalle en el apartado 4.2.

El funcionamiento del sistema consiste en que el usuario solicita información correspondiente al modo y tiempo de desplazamiento entre dos puntos de una ciudad. Esta petición llega al módulo servidor, que es el encargado de procesarla y obtener los datos de los puntos origen y destino entre los que se quiere calcular el tiempo de desplazamiento. El módulo servidor busca primeramente la información solicitada en un fichero que hace las veces de memoria caché o base de datos. Si la encuentra allí, la devolverá en la respuesta, mientra que si no la encuentra, la obtiene de Internet a través del módulo de extracción de información y la almacena en dicho fichero, a la vez que la devuelve en la respuesta. Dicha información es ofrecida por diversas páginas Web de Internet, por lo que el módulo de extracción de información debe extraerla para que el módulo del servidor pueda servirse de ella y ofrecérsela al usuario en forma de respuesta.

Como vimos en el apartado 3.2., existen una serie de módulos que intervienen entre el usuario y nuestra aplicación, pero cuyo funcionamiento no queda detallado en esta memoria debido a que son módulos implementados en los proyectos de otros compañeros.

3.4. Campos de búsqueda

El proyecto consiste en el cálculo del modo y tiempo de desplazamiento entre dos puntos de una ciudad utilizando fuentes públicas. El campo de búsqueda lo reducimos a una ciudad en concreto, ya que si lo hiciésemos para cualquier ciudad, la cantidad de información necesaria que habría que buscar y almacenar sería casi inabarcable. Por lo tanto, este proyecto está centrado en una sola ciudad, en concreto, en Madrid.

Si hacemos caso al título del Proyecto Fin de Carrera, hay que calcular el tiempo de desplazamiento entre dos puntos de una ciudad utilizando solamente fuentes públicas de transporte, como son el autobús, el metro o el cercanías. Sin embargo, para hacer un estudio más amplio y completo, he decidido calcular el tiempo de desplazamiento utilizando no sólo fuentes públicas de transporte, sino también en coche y a pie, ya que Internet te permite obtener toda esa información. Además, el itinerario en coche se podría considerar como si fuese transporte en taxi, con lo que sería también un medio de transporte público.

3.5. Aplicación de Minería Web

El Proyecto Fin de Carrera realizado está basado en la extracción de información de Internet. De ese tema se encarga precisamente la Minería Web. Como bien se explicó en la introducción teórica, la Minería Web consta de cuatro fases, por lo que vamos a identificar cada una de esas fases con el proceso desarrollado en la realización del Proyecto Fin de Carrera.

- **Recuperación de la información (RI)** (Descubrimiento de Recursos)

La RI trata de la recuperación automática de todos los documentos relevantes disponibles en la Web. Los motores de búsqueda son programas escritos para consultar y recuperar información guardada en las bases de datos (Estructuradas), páginas HTML (Semiestructuradas), y el texto libre (Sin Estructurar) en la Web.

De acuerdo a la descripción anterior, podemos decir que la fase de recuperación de la información en el Proyecto Fin de Carrera la asociamos a la selección de páginas Web de las que se realiza la extracción de información.

Utilizamos un motor de búsqueda, en concreto Google, para buscar las páginas Web relacionadas con itinerarios y trayectos entre dos puntos de una ciudad. Es en este punto donde aparecen las dificultades descritas en la introducción teórica relacionadas con la ejecución de esta fase. La subjetividad, imprecisión, e incertidumbre expresada en las consultas que se realizan al motor de búsqueda hacen difícil que éste pueda estimar la importancia de los documentos para el usuario, ya que los motores de búsqueda actuales no tienen capacidad deductiva. Los sistemas de RI existentes ofrecen un modelo muy simple de recuperación que es preferiblemente más rápido que exacto. Los seres humanos encuentran difícil examinar una lista entera de documentos que retorna el motor de la búsqueda como repuesta a la consulta. Debido a todos estos motivos, al realizar en el motor de búsqueda consultas con las palabras “itinerario”, “trayecto”, “callejero”, etc., aparecían una gran cantidad de páginas Web relacionadas, pero la mayoría de ellas no tenían nada que ver con lo que vamos buscando para el Proyecto Fin de Carrera. Debido a ello, de todas las listas de páginas Web retornadas por el motor de búsqueda en cada una de las consultas, hemos tenido que seleccionar manualmente las que más nos podían ayudar a realizar el Proyecto Fin de Carrera. De todas ellas, nos hemos quedado con dos, descartando el resto. Los motivos por los que nos hemos quedado con dos y hemos descartado las demás se exponen más adelante.

Las páginas descartadas son:

http://maps.google.es/

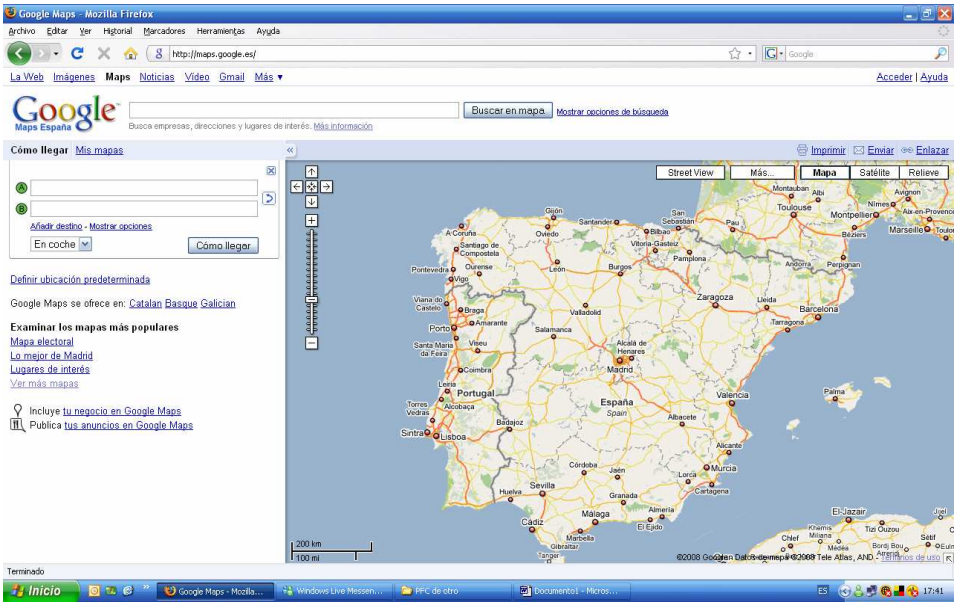


Figura 6. http://maps.google.es/

www.viamichelin.es

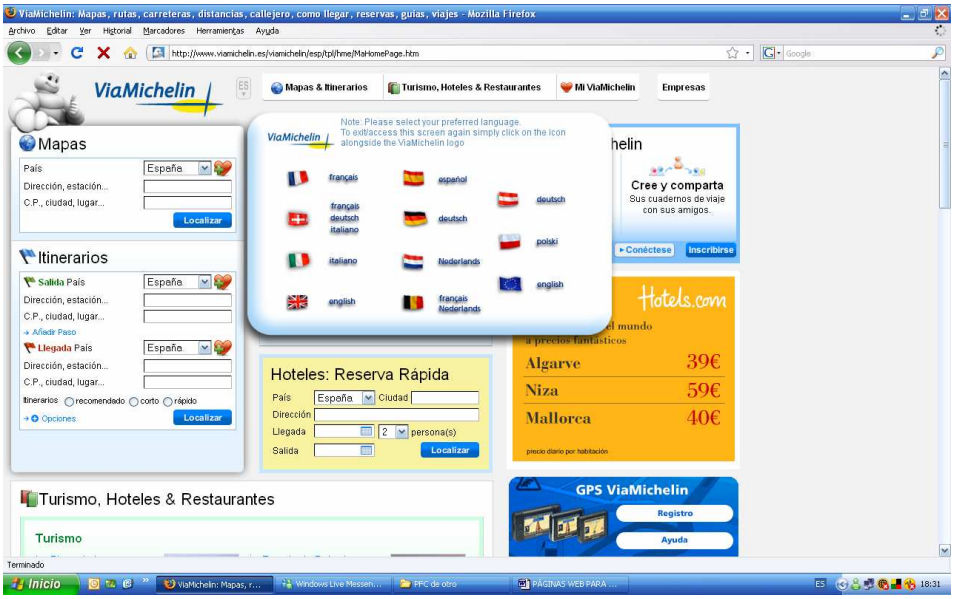


Figura 7. www.viamichelin.es

http://callejero.paginasamarillas.es/home.asp

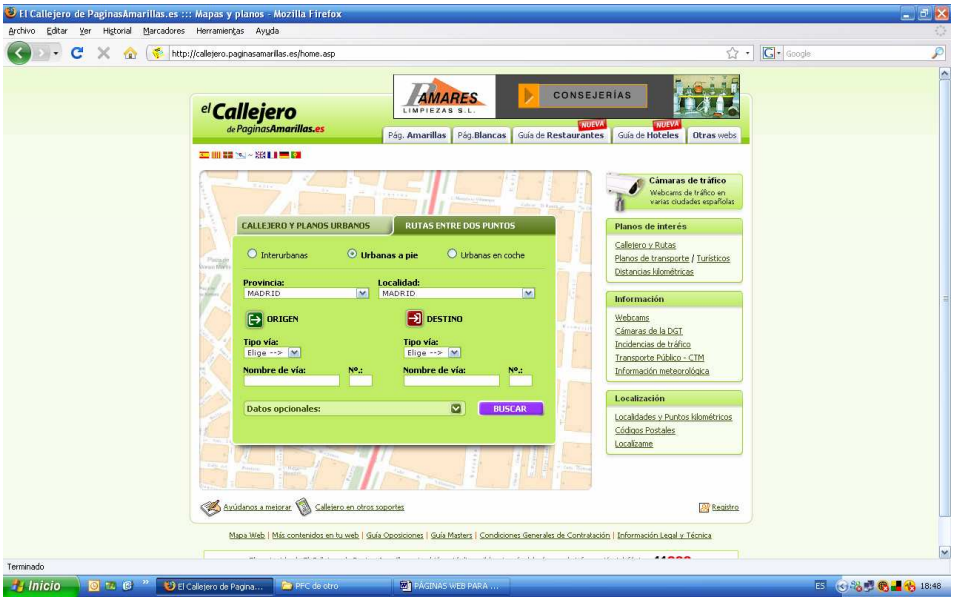


Figura 8. http://callejero.paginasamarillas.es/home.asp

www.guiacampsa.com

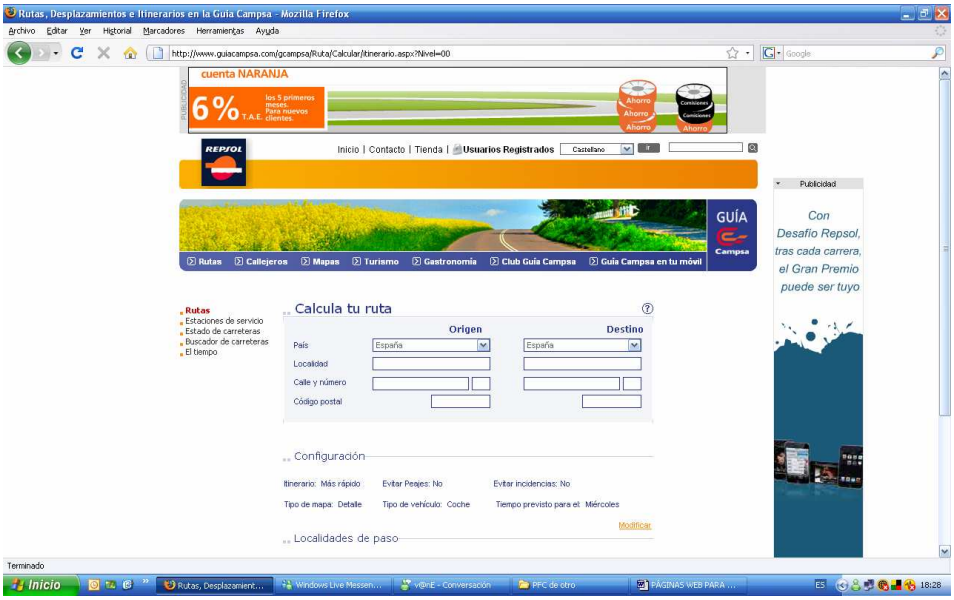


Figura 9. www.guiacampsa.com

Las tres primeras páginas Web se han descartado por la estructura interna de la página, o mejor dicho, su código fuente, ya que no nos aporta la información necesaria para poder programar la aplicación que se pretende realizar. Al buscar el itinerario entre dos puntos de una ciudad en estas páginas Web, éstas nos dan la información del tiempo que se tarda entre esos puntos, tanto en coche como a pie, y resulta paradójico que las descarte, porque esa es la información que necesito para mi Proyecto Fin de Carrera. Sin embargo, la información nos la ofrecen a través de un script de java, con lo que no aparece en el código fuente de la página, por lo tanto, no es válido para la aplicación

implementada, ya que su programación necesita examinar el código fuente de las páginas Web para extraer la información necesaria.

La cuarta página Web sí nos ofrece en su código fuente la información que necesitamos del tiempo que se tarda entre dos puntos de una ciudad. Sin embargo, la he descartado porque sólo obtenemos información de itinerarios en coche, por lo que no la he tenido en cuenta para mi Proyecto Fin de Carrera por considerar que aporta insuficiente información.

Las páginas Web que he seleccionado para la extracción de la información necesaria para la aplicación implementada son:

http://www.ctm-madrid.es/como_ir_a/madrid/como_ir_a_madrid1.jsp

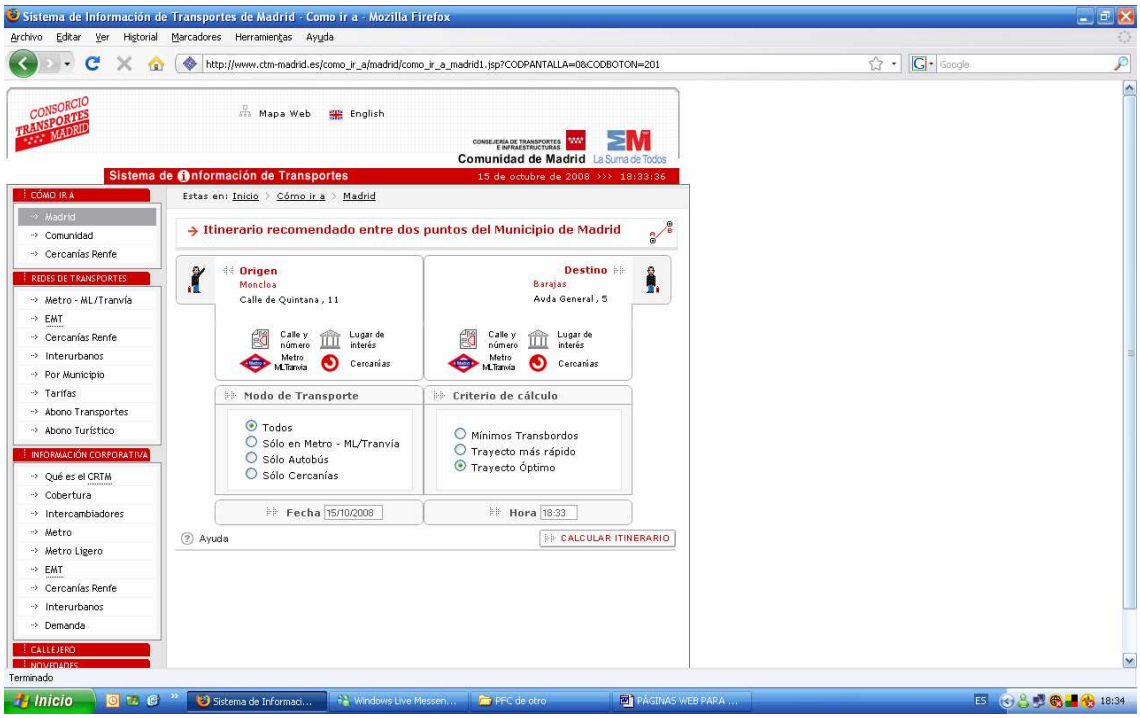


Figura 10. http://www.ctm-madrid.es/como_ir_a/madrid/como_ir_a_madrid1.jsp

Esta página Web es del Sistema de Información de Transportes de Madrid, que te ofrece información del transporte público en la comunidad de Madrid. En particular, se utiliza la sección "Itinerario recomendado entre dos puntos del Municipio de Madrid", que te indica cómo desplazarse entre dos puntos del municipio de Madrid empleando las Redes de Transporte Público, cuya dirección URL es la indicada anteriormente.

He utilizado esta página Web para obtener el tiempo que se tarda entre dos puntos de Madrid utilizando autobús, metro, cercanías y una combinación de los tres.

www.mappy.com

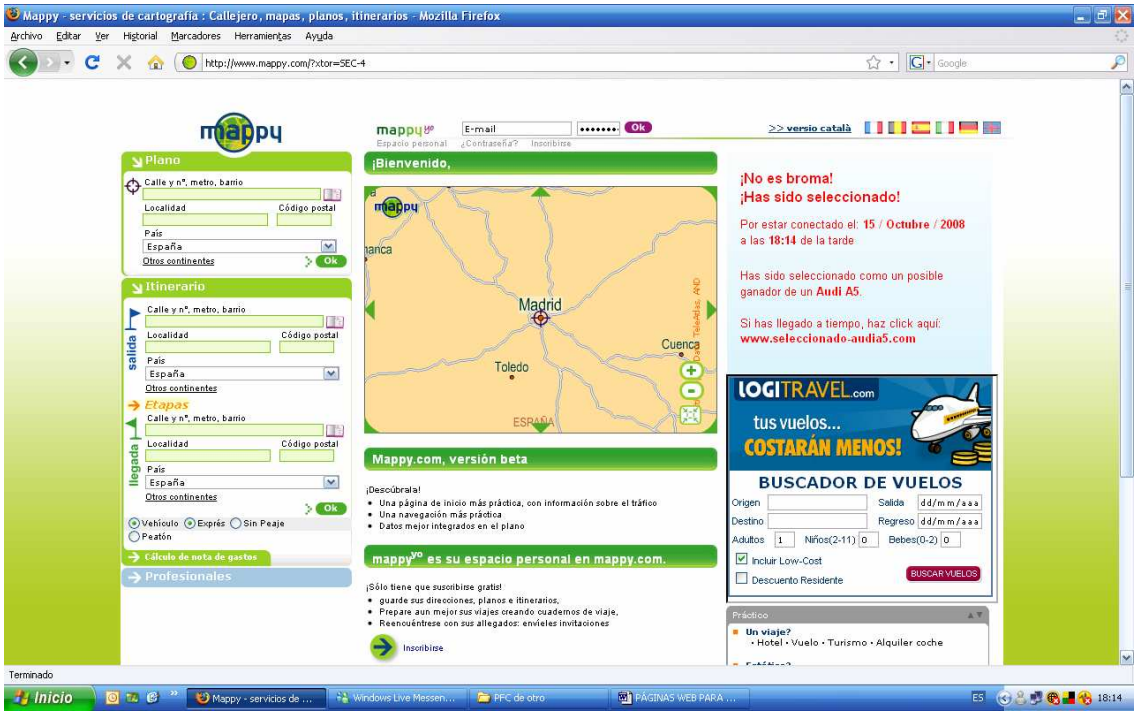


Figura 11. www.mappy.com

Esta página Web ofrece información sobre el itinerario entre dos puntos, tanto para ir en coche como para ir a pie. Por lo tanto, he utilizado esta página Web para obtener el tiempo que se tarda entre dos puntos de Madrid en coche y a pie.

Las páginas Web seleccionadas cumplen los requisitos por los que se descartaron las páginas Web anteriores, es decir, que proporciona información de los itinerarios y el tiempo que se tarda entre dos puntos de una ciudad, y además el código fuente nos facilita la información necesaria para programar la aplicación.

- **Extracción de la Información (EI)** (Extracción y Pre-procesamiento)

Tiene como objetivo transformar los documentos extraídos en el proceso de recuperación de información en documentos que sean más digeribles, fáciles de leer y de analizar. La extracción de información (EI) se refiere a la tarea de identificar fragmentos específicos de un documento. La EI apunta a extraer nuevo conocimiento de los documentos recuperados centrándose en la estructura del documento y en la representación del documento. Los métodos principales de EI involucran los ”Wrappers”. Los Wrapper son procedimientos para extraer información particular de los recursos de la Web.

De acuerdo a la descripción anterior, podemos decir que la fase de extracción de la información en el Proyecto Fin de Carrera la asociamos a la extracción del tiempo que se tarda entre el origen y el destino de la petición utilizando los distintos medios de

transporte. Esta información la extraemos del código fuente de las páginas Web seleccionadas en la fase anterior de recuperación de la información.

Un elemento importante para la realización de esta fase es el Wrapper, que como bien se mencionó anteriormente, se utiliza para la extracción de información de los recursos Web. Como se vio en la introducción teórica, hay dos formas de crear un Wrapper. Una es utilizando un generador de Wrappers y la otra es construirlo sin utilizar ninguna herramienta de generación de Wrappers. Lo más fácil y cómodo es utilizar un generador de Wrapper, ya que te permite crear un Wrapper ahorrándote mucha programación. Son herramientas que te generan un Wrapper simplemente con introducir una serie de parámetros, sin apenas programar. Sin embargo, si no se utiliza una herramienta generadora, hay que programar el Wrapper entero, con la dificultad que la programación conlleva. Por lo tanto, mi primera intención fue utilizar una herramienta generadora de Wrappers para crear el que necesito para mi Proyecto Fin de Carrera.

Como se detalló en la introducción teórica, hay varios tipos de generadores de Wrappers, clasificados según la tecnología utilizada. De todos ellos, unos son generadores comerciales y otros son generadores no comerciales. El principal problema que presentan todos los generadores no comerciales es la dificultad de uso. Muchos se basan en lenguajes de script propietarios para la generación de los Wrappers, además casi todos carecen de interfaz, por lo que es necesario realizar una secuencia de comandos. Otro problema es que al tratarse de código abierto no son estables al 100% y no soportan determinadas características, como páginas con JavaScript. Los generadores comerciales presentan una mayor facilidad de uso, prácticamente todos dan la posibilidad de generar XML, e incluyen una interfaz gráfica más o menos intuitiva. Incluso algunos de ellos permiten la extracción sin tener conocimientos previos de HTML.

Los que nos interesan son los generadores no comerciales, ya que son los que podemos usar sin ningún tipo de costes de adquisición de licencia. En la tabla 2 podemos ver un resumen de los principales generados de Wrappers no comerciales encontrados:

Toolkit	Output Data	Java API	Open Source	Source Code	Web Crawling	GUI	Editor	Scripting Language	Tool Support
Araneus	XML, Text	✓	only API	Java	—	—	—	EDITOR	—
BYU	Text, Tabular	✓	—	Java	—	—	—	ontologies	—
COMMIX	HTML, XML, Text	No explicit API	—	Java	✓	✓	—	—	(✓) Chinese toolkit
DEByE	XML, Text	No explicit API	—	Java	✓	✓	—	—	✓
Info Extractor	DOM type structure	✓ JDBC	—	Java	—	—	—	reg. expr., grammar rules	—
Jedi	XML, Text	✓	—	Java	✓	—	—	JEDI	—
LAPIS	HTML, XHTML, XML, Text	✓	✓	Java	✓	✓	✓	Text Constraints	✓
NoDoSE	XML, Tabular, ODL, OEM	✓	✓	Java	—	✓	—	—	—
Road Runner	XML, HTML, Text	✓	—	Java	✓	—	—	regular expressions	Toolkit not yet available
Scout	HTML, Text	✓	✓	Java	—	—	—	regular expressions	—
SoftMealy	HTML, Text	✓	✓	Java	✓	✓	—	—	—
TSIMMIS	HTML, Text	—	✓	C/C++	—	—	—	—	—
WebL	XML, HTML, Text	✓	✓	Java	✓	—	(✓) third party	WebL	(✓) News-groups
Web Sphinx	HTML, Text	No explicit API	✓	Java	✓	✓	✓	ORO Matcher reg. expr.	✓
WIEN	HTML, Text	No explicit API	✓	Java	—	✓	—	—	—
XWrap	XML (incl. DTD)	✓	(✓) only Wrapper	Java	✓	✓	—	—	✓

Tabla 2. Generadores de Wrappers no comerciales.

El lenguaje de programación que he decidido utilizar para programar la aplicación del Proyecto Fin de Carrera es el lenguaje C, por lo tanto, hay que buscar un generador de Wrappers cuyo código fuente sea el lenguaje C. Si nos fijamos en la tabla 2, la mayoría de los generadores de Wrappers no comerciales son en el lenguaje Java, mientras que el único que tiene código fuente en lenguaje C es TSIMMIS.

TSIMMIS es el nombre de un proyecto que desarrolló herramientas que facilitaran la integración de información desde varias fuentes con estructura o sin ella.

Proporciona una forma de generar Wrappers automáticamente a través de un lenguaje de descripción llamado MSL. En él se especifican una serie de tripletas (variables, fuente, patrón) que, tras ser procesadas, generan un Wrapper que realiza las consultas en el lenguaje nativo de la fuente accedida. El patrón es buscado en el documento fuente y los datos resultantes son almacenados en las variables para su posterior proceso. Este proyecto tiene un hermano desarrollado por IBM llamado Garlic, que se limita a la generación de Wrappers a través de un proceso visual.

Puesto que es el único generador no comercial del que me puedo servir, decido utilizarlo para generar el Wrapper para mi Proyecto Fin de Carrera. Accedo a la página Web donde puedo obtener los archivos y códigos de TSIMMIS y los descargo. Pero es entonces cuando aparecen los problemas mencionados anteriormente de los generadores no comerciales. Aparte de lo complicado que me resultó su instalación, es mucho más complicada aún su utilización. No tiene interfaz, con lo que todas las operaciones necesitan una serie de comandos, dificultando en exceso su uso. Por lo tanto, tras una serie de intentos y no obtener resultado ninguno, decido no utilizar el generador de Wrappers TSIMMIS.

Puesto que no podía disponer de ningún generador de Wrappers no comercial adecuado para realizar mi Proyecto Fin de Carrera, me decidí por la segunda opción de creación de Wrappers, que es construirlo de principio a fin programándolo, y en mi caso, con el lenguaje C.

El sistema utilizado para implementar el Wrapper es un sistema Mediador-Wrapper del estilo que muestra la figura 12:

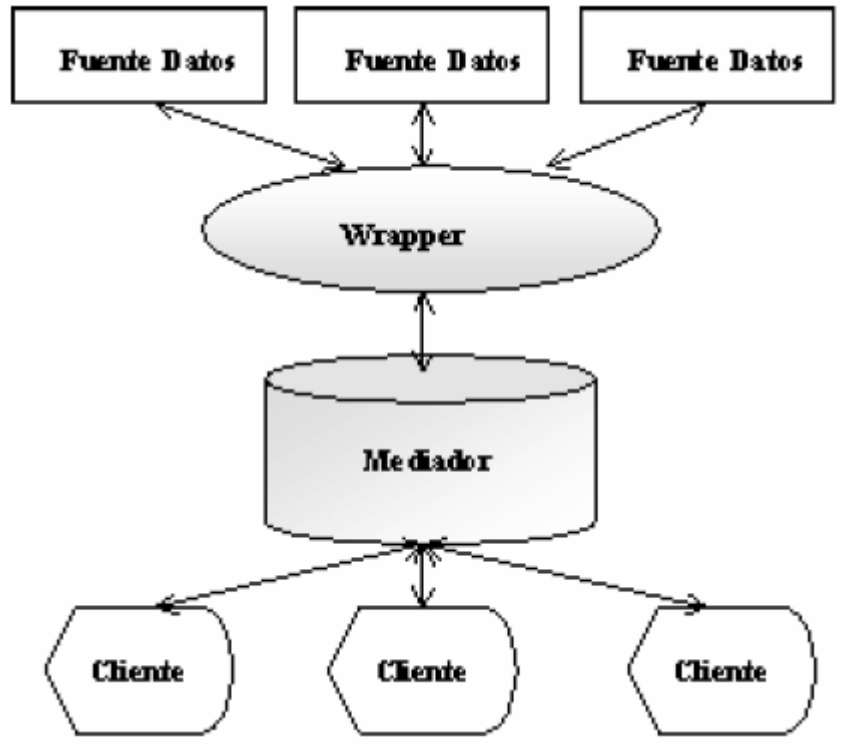


Figura 12. Sistema Mediador-Wrapper

Este sistema permite que los Wrapper extraigan la información de fuentes heterogéneas y se integre a través de los mediadores. Los mediadores pueden ser bases de datos convencionales o bases de conocimiento representadas con datos estructurados extraídos desde fuentes semi o no estructurados. Las consultas de los usuarios se aplican sobre estos almacenes de datos, en lugar de ir directamente a la Web.

En nuestro caso, el Wrapper extrae la información solicitada de las dos páginas Web seleccionadas en la fase anterior y el mediador la almacena. Dicho mediador no es una base de datos en sí, sino simplemente un fichero, ya que la información a almacenar no es muy grande. El mediador maneja un esquema de datos común capaz de representar los datos ofrecidos por cada fuente de datos. El Wrapper conoce el esquema de datos de su correspondiente fuente de datos y traduce los datos al esquema de datos existente en el mediador. De esta forma, las consultas del usuario van directamente sobre el fichero que hace de base de datos en el mediador. Si encuentra allí la información solicitada, la obtiene y la devuelve en la respuesta. Si no la encuentra allí, la consulta se direcciona al Wrapper para extraer la información requerida de las dos páginas Web. Una vez extraída, se almacena en el fichero base de datos y se devuelve la información en la respuesta. El esquema del sistema Mediador-Wrapper de mi Proyecto Fin de Carrera se muestra en la figura 13:

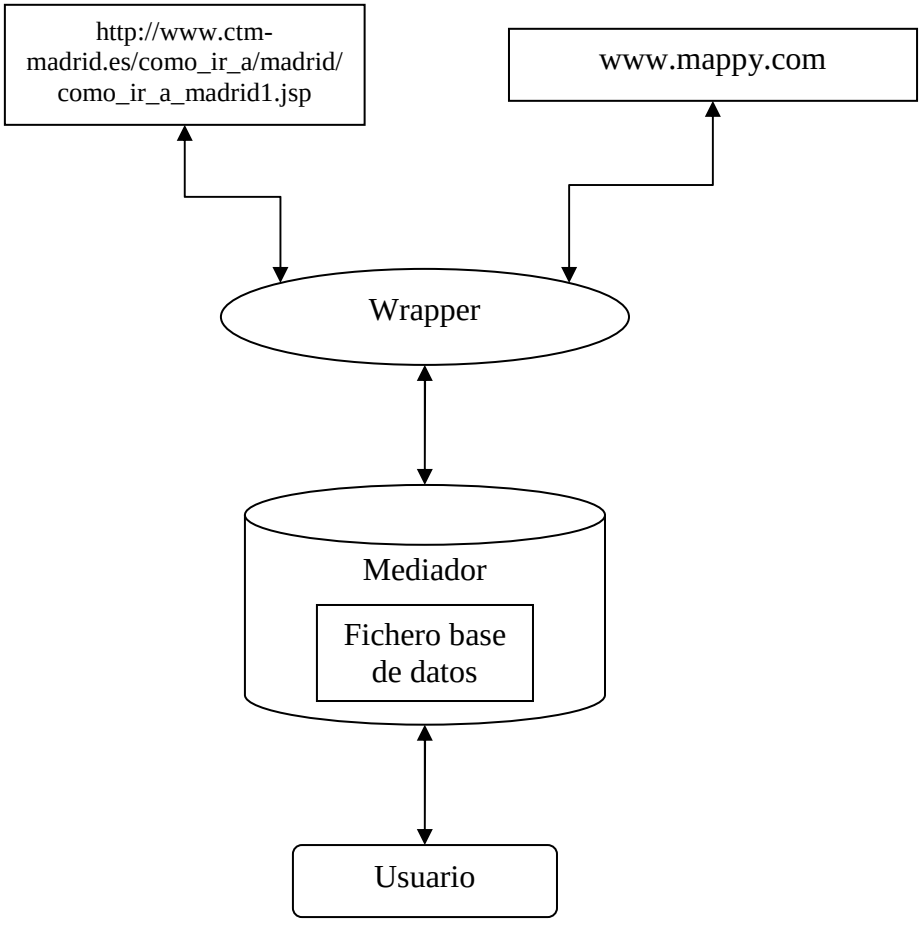


Figura 13. Sistema Mediador-Wrapper del Proyecto Fin de Carrera

En el apartado 3.6 se detalla más concretamente cómo se ha programado el sistema Mediador-Wrapper de la aplicación del Proyecto Fin de Carrera.

Los Wrapper son buenos procedimientos para extraer información particular de los recursos de la Web, sin embargo, su mayor limitación es que cada Wrapper es un sistema de extracción de la información personalizado para un sitio particular y no es universalmente aplicable. El Wrapper creado para este Proyecto Fin de Carrera no se puede aplicar a otro sitio, ya que está basado en extraer información específica de tiempo requerida para esta aplicación a partir de dos páginas Web específicas.

Para acabar este apartado de extracción de la información, queremos explicar por qué se ha elegido un fichero como sistema de almacenaje de información en el mediador:

Como vemos, para la implementación de la aplicación necesitamos una especie de base de datos donde almacenar la información extraída de Internet, de manera que el módulo del servidor pueda servirse de ella para crear las respuestas que satisfagan las peticiones de entrada. En un principio se pensó en crear una ontología, quedando así toda la información de un dominio organizada y relacionada entre sí. Para ello necesitamos una herramienta de creación de ontologías. Como se vio en la introducción teórica, existen varias, de las cuales se podría utilizar la herramienta *Protégé-2000*.

Protégé-2000, desarrollada por el *Stanford Medical Informatics* de la Universidad de Stanford, es la herramienta de construcción de ontologías que más usuarios tiene. Con ella se pueden fácilmente crear clases y jerarquías, declarar propiedades para las clases, crear instancias e introducir valores, todo ello en un entorno de menús, botones, cuadros de diálogo y representaciones gráficas fáciles de usar.

Protégé-2000 es una herramienta integrada de software para desarrollar sistemas basados en el conocimiento. Las aplicaciones desarrolladas con *Protégé-2000* se utilizan para la solución de problemas y la toma de decisiones en un dominio particular. Esta herramienta facilita trabajar simultáneamente con las clases y las instancias.

Protégé-2000 permite:

- Modelar una ontología de las clases que describen un tema particular.
- Creación de una herramienta de adquisición de conocimiento para almacenar conocimiento.
- La introducción de instancias específicas de datos y la creación de una base de conocimiento.
- La ejecución de aplicaciones.

La ontología define el sistema de conceptos y sus relaciones. Esta herramienta de adquisición del conocimiento se diseña para ser específica del dominio, permitiendo a expertos del dominio incorporar fácil y naturalmente su conocimiento del área. La base de conocimiento que resulta se puede entonces utilizar para contestar a preguntas y para solucionar problemas con respecto al dominio.

La ontología creada con Protégé-2000 para este proyecto podría ser la siguiente:

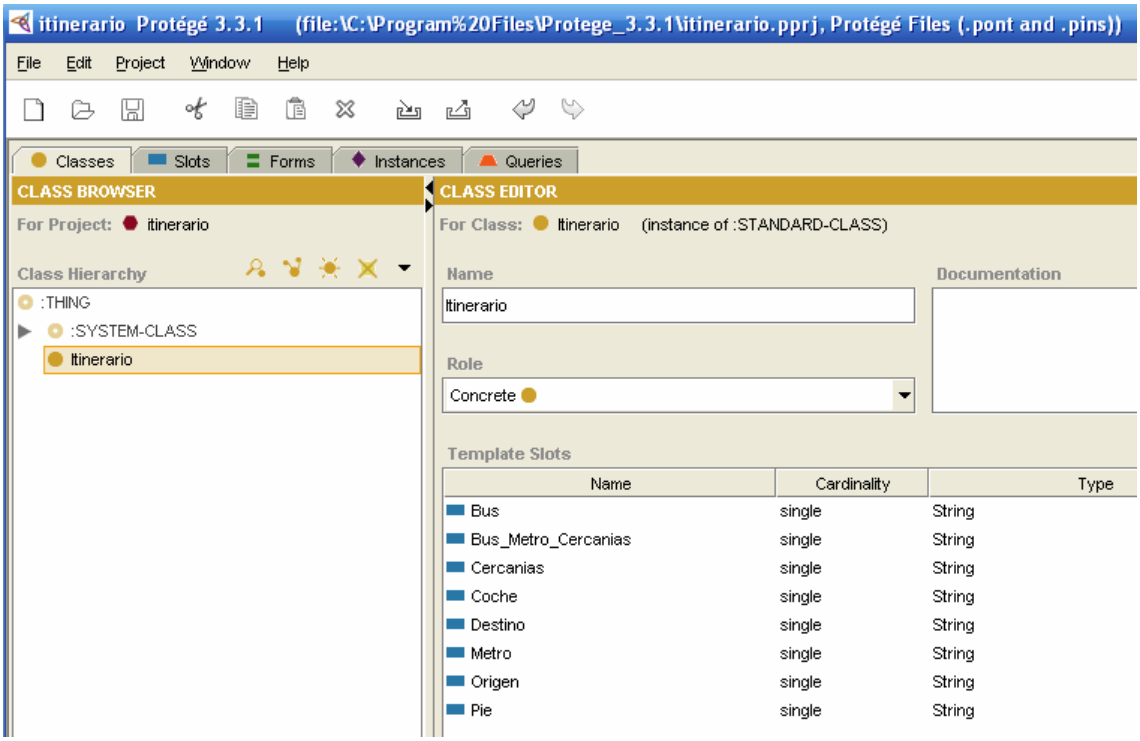


Figura 14. Ontología creada con Protégé-2000

La ontología creada se guarda en el archivo itinerario.pprj. Se ha creado una sola clase, llamada itinerario, cuyos atributos son cada uno de los campos que se necesitan para construir la respuesta XML que el módulo del servidor tiene que devolver. Esta ontología almacena la información extraída por el módulo de extracción de información. Se almacenan tantas instancias de la clase itinerario como peticiones posibles se pueden realizar con las direcciones de origen y destino elegidas para este proyecto. Cada instancia almacena información de origen, destino y duración del itinerario que se solicita utilizando cada uno de los medios de transporte.

Como vemos, una ontología es un buen medio para almacenar la información y que quede organizada. Sin embargo, para este Proyecto Fin de Carrera he decidido no utilizar una ontología como medio de almacenamiento de información, ya que la ontología es muy válida cuando la información a almacenar o devolver al usuario es muy variada y muy extensa. En este caso, podemos observar cómo la información a devolver o almacenar no es muy variada y utiliza muy pocos campos, con lo que una ontología sería innecesaria.

Se podría implementar una pequeña base de datos, o incluso una memoria caché, para almacenar la información de la que pueda servirse el módulo del servidor. Sin embargo, como la información a devolver no es extensa, he decidido almacenarla en un archivo, de manera que el módulo del servidor simplemente busque en ese archivo si contiene la información solicitada.

- **Generalización**

Reconocimiento de Patrones generales de una página en particular o bien también patrones de diferentes páginas. En esta fase, se usa normalmente reconocimiento de patrones y técnicas de aprendizaje automático para la información extraída.

- **Análisis**

El análisis es un problema de manipulación de datos que asume que hay suficientes datos disponibles para que la información potencialmente útil pueda extraerse y analizarse. Los humanos juegan un papel importante en el proceso de descubrimiento de conocimiento de la información de la Web, ya que la Web es un medio interactivo. Esto es especialmente importante para la aprobación y/o interpretación de los patrones encontrados, ya que necesitan de la aprobación humana. Una vez se han descubierto los modelos, los analistas necesitan las herramientas apropiadas para entender, visualizar, e interpretar este conocimiento.

Las fases de generalización y análisis se suelen conjuntar en una fase de evaluación de la información extraída en la fase anterior de extracción de la información. En este caso, como se ha mencionado, cobra importancia el factor humano, ya que una vez extraída la información solicitada hay que analizar de qué parte de los documentos se ha extraído para poder acceder a ella. En este caso, lo que se ha realizado es analizar los códigos fuente de las páginas Web de donde se extrae la información solicitada. Una vez identificada la parte donde se encuentra la información concreta, se programa la aplicación para que acceda directamente a ella en cada consulta.

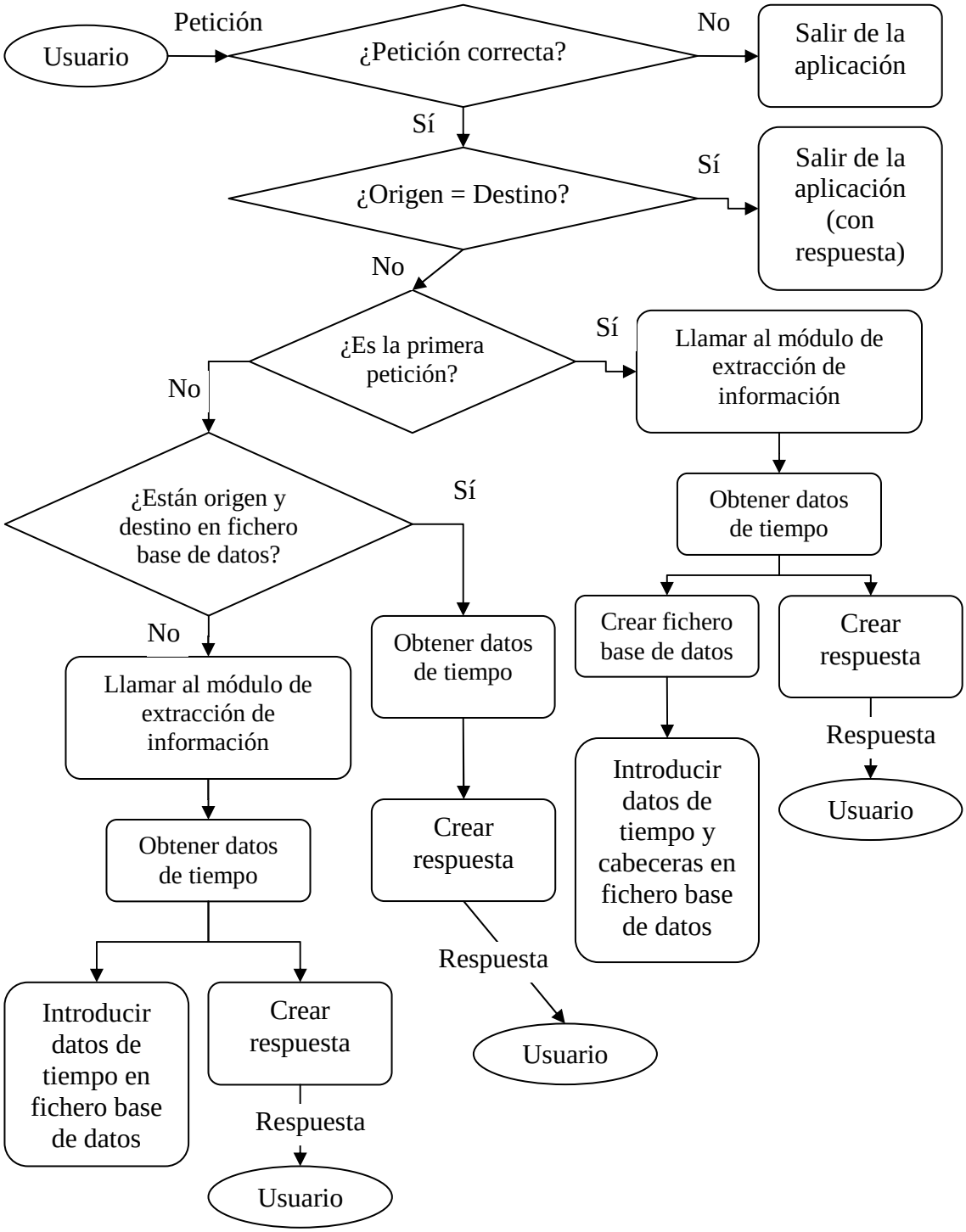
En el apartado 3.6 se detalla más concretamente cómo se han programado las fases de generalización y análisis de la aplicación del Proyecto Fin de Carrera.

3.6. Descripción de los módulos implementados

En este apartado se explica con detalle la implementación de los módulos que componen el Proyecto Fin de Carrera, que son el *módulo de extracción de la información* y el *módulo del servidor* de dicho módulo.

3.6.1. Módulo del servidor

Iniciamos la explicación del módulo del servidor con un diagrama que resume el funcionamiento del mismo y después se explicará con más detalle su implementación.



A continuación se explica detalladamente cómo se han implementado todos los pasos del diagrama anterior.

El usuario, a través de su PDA, envía la petición al sistema solicitando el tiempo que se tarda entre dos calles de Madrid utilizando medios de transporte públicos.

La petición pasa por otros módulos antes de llegar al módulo del servidor. Esos módulos no se detallan en este proyecto porque son implementados por otros alumnos.

Lo primero que se hace en el módulo del servidor es comprobar si el fichero en el que llega la petición es correcto. Si es correcto, se abre y se procesa. Si no es correcto, se sale de la aplicación.

```
if  ((fichero_peticion  =  fopen("peticion.txt",  "r"))  ==  NULL)  {
  printf("Error: No se puede abrir el fichero %s\n", "peticion.txt");
  exit(1);
}
```

Con la función *fopen* procedemos a abrir el fichero *peticion.txt* en modo lectura "*r*", de manera que podamos obtener su contenido. La función *fopen* retorna un puntero al objeto que identifica el fichero. Si el proceso de apertura fallara porque el fichero no existe o no puede ser leído, entonces retorna un puntero nulo. En este último caso, vemos que la aplicación acaba por medio de la función *exit*.

Una vez leído el contenido del fichero petición, procedemos a obtener los datos de los puntos de origen y destino entre los que se quiere calcular el tiempo de desplazamiento. Para ello nos servimos de la función *strstr*:

```
inicio_origen = strstr(array_peticion,"<Origen>");
inicio_destino = strstr(array_peticion,"<Destino>");
```

La función *strstr* localiza la primera aparición de la cadena de caracteres <Origen> o <Destino> en la cadena de caracteres del array_peticion, que es un array en el que hemos almacenado anteriormente todos los caracteres del fichero petición. La función retorna un puntero a la cadena encontrada, o un puntero nulo si no se encontró la cadena. De esta forma, nos situamos en las etiquetas del origen y destino, y posteriormente obtenemos las cadenas de caracteres que se encuentran entre las etiquetas de apertura y de cierre de origen y destino.

Una vez que ya tenemos los puntos de origen y destino de la petición, comprobamos si ambos son iguales, ya que en tal caso, no tiene sentido utilizar la aplicación. Para ello nos servimos de la función *strcmp*:

```
strcmp(peti->origen, peti->destino)
```

La función *strcmp* compara la cadena apuntada por peti->origen con la cadena apuntada por peti->destino, que son dos punteros que apuntan donde están almacenados origen y destino de la petición. La función retorna un número entero mayor, igual, o menor que cero apropiadamente según la cadena apuntada por peti->origen es mayor, igual, o menor que la cadena apuntada por peti->destino.

Si la función *strcmp* devuelve un número distinto de 0, significa que origen y destino no son iguales, con lo que el programa sigue ejecutándose. Sin embargo, si retorna un 0 quiere decir que origen y destino son iguales, con lo que la aplicación termina y se sale del programa con la función *exit*. Además, en este último caso, la aplicación le devuelve al usuario un fichero de respuesta indicándole que el origen y el destino son iguales. Todo esto está implementado de la siguiente manera:

- Comprobamos si origen y destino son iguales:

```
if (strcmp(peti->origen, peti->destino) == 0) {
```

- Si son iguales, lo primero que hacemos es crear el fichero de respuesta con la función *fopen*, cuyos argumentos son el nombre del fichero *respuesta.txt* y la indicación de que el fichero lo creamos para escritura *"w"*:

```
if ((fichero_respuesta = fopen("respuesta.txt", "w")) == NULL) {
    printf("Error: No se puede abrir el fichero %s\n",
           "respuesta.txt");
    exit(1);
}
```

- Inicializamos un array de caracteres llamado *datosrespuesta* para escribir en él los datos de la respuesta, que posteriormente escribiremos en el fichero creado anteriormente. Para ello, primero se reserva memoria con la función *malloc* para un tamaño igual al número de caracteres que componen la respuesta y después se le introducen los datos de la respuesta:

```
int tam = 154+(2*strlen(peti->origen))+(2*strlen(peti->destino));
datosrespuesta = (char *)malloc(sizeof(char)*(tam+1));
strcpy(datosrespuesta, "<?xml version=\"1.0\" encoding=\"UTF-
      8\"?>\n<Respuesta>\n\t<Origen>");
strcat(datosrespuesta, peti->origen);
strcat(datosrespuesta, "</Origen>\n\t<Destino>");
strcat(datosrespuesta, peti->destino);
strcat(datosrespuesta, "</Destino>\n\tOrigen=\"");
strcat(datosrespuesta, peti->origen);
strcat(datosrespuesta, "\" y Destino=\"");
strcat(datosrespuesta, peti->destino);
strcat(datosrespuesta, "\" son iguales\n</Respuesta>");
```

Con las funciones *strcpy* y *strcat* vamos copiando en el primer caso, y concatenando en los siguientes, cadenas de caracteres en el array *datosrespuesta* creado hasta formar los datos de la respuesta, que como vemos, tendrá formato XML. En el primer caso se copian caracteres porque el array está vacío, mientras que en los siguientes se concatenan caracteres a los ya escritos en el array hasta completar todos los datos.

- Escribimos en el fichero creado los datos de la respuesta. Para ello utilizamos la función *fwrite*, que envía desde el array apuntado por *datosrespuesta* los caracteres de la respuesta al fichero apuntado por *fichero_respuesta*:

```
fwrite(datosrespuesta, sizeof(char), tam-15, fichero_respuesta);
```

- Ya por último, salimos de la aplicación con la función *exit*, no sin antes cerrar con la función *fclose* el puntero al fichero utilizado y liberar la memoria reservada para el array de la respuesta con la función *free*:

```
fclose(fichero_respuesta);

free(datosrespuesta);

exit(1);
```

Si origen y destino son iguales, se ejecuta todo lo explicado anteriormente y la aplicación termina, devolviendo al usuario la respuesta creada. Sin embargo, éste es un caso anómalo, y lo normal es que la consulta del usuario no sea con origen y destino iguales, por lo que la aplicación seguiría ejecutándose con normalidad.

Una vez comprobado que origen y destino no son iguales, vemos si es la primera petición que se realiza al sistema. Si es así, sabemos que no existe todavía el fichero base de datos y no tendremos que buscar en él los datos de tiempo solicitados, por lo que hay que ir directamente al módulo de extracción de información para extraerlos de Internet y posteriormente crear el fichero base de datos para introducirlos en él, junto con sus cabeceras. Todo este proceso, que está esquematizado en el diagrama anterior, se implementa de la siguiente manera:

- Comprobamos si es la primera petición mediante la función *fopen*. Con ella intentamos abrir el fichero *basedatos.txt* para lectura, y si nos devuelve NULL quiere decir que el fichero no existe, por lo tanto no se ha hecho antes ninguna petición, ya que cada vez que se extrae un dato de Internet con el módulo de extracción de información se guarda en el fichero base de datos.

```
if (fopen("basedatos.txt", "r") == NULL){
```

- Si la función *fopen* devuelve NULL, resulta que es la primera petición, con lo que pasamos directamente a llamar a la función que implementa el módulo de extracción de información:

```
resp = extractor(peti);
```

La función *extractor* es la función creada que implementa el módulo de extracción de información. Esta función tiene como argumento los datos de la petición y devuelve los datos extraídos por el módulo de extracción de información. En este punto ya obtenemos los datos de tiempo solicitados, por lo tanto, lo siguiente es crear la respuesta con ellos y crear el fichero base de datos para introducirlos, ya que, como dijimos, éste no existe.

- Para crear el fichero base de datos e introducir la información en él, lo implementamos de la misma manera que se explicó anteriormente para crear el fichero de respuesta. Lo primero que hacemos es crear el fichero base de datos con la función *fopen*, cuyos argumentos son el nombre del fichero *basedatos.txt* y la indicación de que el fichero lo creamos para escritura, pero añadiendo sobre lo anterior sin borrarlo *"a+"*:

```
if ((fichero_basedatos = fopen("basedatos.txt", "a+")) == NULL){
      printf("Error: No se puede abrir el fichero %s\n",
             "basedatos.txt");
      exit(1);
}
```

- Inicializamos un array de caracteres llamado *cabecerasydatos* para escribir en él las cabeceras y los datos de tiempo extraídos, que posteriormente escribiremos en el fichero creado anteriormente. Para ello, primero se reserva memoria con la función *malloc* para un tamaño igual al número de caracteres que componen las cabeceras y los datos extraídos y después los introducimos:

```
int      longi      =      (103+strlen(resp->origen)+strlen(resp-
>destino)+strlen(resp->bus)+strlen(resp->metro)+strlen(resp-
>cercanias)+strlen(resp->bus_metro_cercanias)+strlen(resp-
>coche)+strlen(resp->pie));

cabecerasydatos = (char *)malloc(sizeof(char)*(longi+1));

strcpy(cabecerasydatos, "Origen - Destino - Bus - Metro –
Cercanias - Bus_Metro_Cercanias - Coche - Pie\n");
strcat(cabecerasydatos, resp->origen);
strcat(cabecerasydatos, " - ");
strcat(cabecerasydatos, resp->destino);
strcat(cabecerasydatos, " - ");
strcat(cabecerasydatos, resp->bus);
strcat(cabecerasydatos, " - ");
strcat(cabecerasydatos, resp->metro);
strcat(cabecerasydatos, " - ");
strcat(cabecerasydatos, resp->cercanias);
strcat(cabecerasydatos, " - ");
strcat(cabecerasydatos, resp->bus_metro_cercanias);
strcat(cabecerasydatos, " - ");
strcat(cabecerasydatos, resp->coche);
strcat(cabecerasydatos, " - ");
strcat(cabecerasydatos, resp->pie);
strcat(cabecerasydatos, "\n");
```

De nuevo utilizamos las funciones *strcpy* y *strcat*, con las que vamos copiando en el primer caso, y concatenando en los siguientes, cadenas de caracteres en el array *cabecerasydatos* creado hasta formar los datos del fichero base de datos. El formato de las cabeceras nos indica que los datos extraídos se irán almacenando en el orden propuesto, que es:

Origen - Destino - Bus - Metro - Cercanias - Bus_metro_cercanias - Coche - Pie

Que quiere decir que primero se almacena el punto de origen, después el punto de destino, y a continuación se van almacenando los datos de tiempo extraídos correspondientes al tiempo que se tarda en desplazarse entre los puntos de origen y destino utilizando autobús, metro, cercanías, una combinación de los tres, en coche y a pie.

- Escribimos en el fichero creado los datos del fichero base de datos. Para ello utilizamos la función *fwrite*, que envía desde el array apuntado por *cabecerasydatos* los caracteres del fichero base de datos al fichero apuntado por *fichero_basedatos*:

```
fwrite(cabecerasydatos, sizeof(char), longi-2, fichero_basedatos);
```

- Ya por último, cerramos con la función *fclose* el puntero al fichero utilizado y liberar la memoria reservada para el array de la base de datos con la función *free*:

```
fclose(fichero_basedatos);

free(cabecerasydatos);
```

- Hemos creado el fichero base de datos y hemos introducido los datos extraídos en él. Lo que queda por hacer si la aplicación sigue el camino de que es la primera petición sería crear la respuesta para el usuario. El procedimiento para ello es el mismo que ya se explicó anteriormente, es decir, creamos el fichero respuesta, reservamos memoria para un array con los datos de la respuesta y los introducimos en él, para posteriormente escribirlos en el fichero creado. Por último se cierra el puntero al fichero creado y se libera toda la memoria reservada.

Esta ha sido la implementación del sistema cuando es la primera petición que se le hace. Sin embargo, si no es la primera petición, o lo que es lo mismo, si la función `fopen("basedatos.txt", "r")` no devuelve el valor NULL, el fichero base de datos ya existe, con lo que la implementación es la siguiente:

- Lo primero que hacemos es abrir el fichero base de datos para leer su contenido. De nuevo utilizamos la función *fopen* indicándole que queremos abrir el fichero *basedatos.txt* en modo lectura "*r*":

```
if ((fichero_basedatos = fopen("basedatos.txt", "r")) == NULL) {
      printf("Error: No se puede abrir el fichero %s\n",
             "basedatos.txt");
      exit(1);
}
```

- A continuación almacenamos en un array de caracteres los datos de los puntos de origen y destino de la petición para buscarlos en el fichero base de datos:

```
busco = (char *)malloc(sizeof(char)*(strlen(peti-
     >origen)+3+strlen(peti->destino)+1));
strcpy(busco, peti->origen);
strcat(busco, " - ");
strcat(busco, peti->destino);
```

Hemos reservado memoria para ese array con la función *malloc* y hemos almacenado en él los datos de origen y destino con las funciones *strcpy* y *strcat*, que como ya sabemos, son las funciones para copiar y concatenar a lo ya escrito cadenas de caracteres. El formato de la cadena de caracteres que queremos buscar en el fichero base de datos es:

"Punto de Origen - Punto de Destino "

porque como ya especificamos anteriormente, el formato con el que almacenamos en el fichero base de datos los datos extraídos es:

Origen - Destino - Bus - Metro - Cercanias - Bus_metro_cercanias - Coche - Pie

por lo tanto, si buscamos en el fichero base de datos esa cadena de caracteres y existe, sabemos que a continuación están los datos de tiempo solicitados.

- Para comprobar definitivamente si los puntos de origen y destino están en el fichero base de datos utilizamos la función *strstr*, que como sabemos, localiza la primera aparición de la cadena de caracteres "Punto de Origen - Punto de Destino " en la cadena de caracteres del *array_basedatos*, que es un array en el que hemos almacenado anteriormente todos los caracteres del fichero base de datos. La función retorna un puntero a la cadena encontrada, o un puntero nulo si no se encontró la cadena. De esta forma, en caso de que exista, nos situamos sobre los datos de tiempo solicitados para posteriormente obtenerlos.

```
if ((encontrado = strstr(array_basedatos, busco)) == NULL)
```

Como vemos, comprobamos con la función *strstr* si en el *array_basedatos* se encuentra la cadena de caracteres que hemos almacenado en el array *busco*. Si la función devuelve un puntero NULL quiere decir que no ha encontrado la cadena buscada en el array base de datos, por lo tanto, no se encuentran almacenados los datos solicitados en el fichero base de datos. En este caso, la aplicación sigue el siguiente camino:

- Pasamos directamente a llamar a la función que implementa el módulo de extracción de información:

```
resp = extractor(peti);
```

Como ya mencionamos anteriormente, la función *extractor* es la función creada que implementa el módulo de extracción de información. Esta función tiene como argumento los datos de la petición y devuelve los datos extraídos por el módulo de extracción de información. En este punto ya obtenemos los datos de tiempo solicitados, por lo tanto, lo siguiente es crear la respuesta con ellos e introducirlos en el fichero base de datos.

- Para introducir la información en el fichero base de datos, lo implementamos de la misma manera que se explicó anteriormente. Lo primero que hacemos es abrir el fichero base de datos con la función *fopen*, cuyos argumentos son el nombre del fichero *basedatos.txt* y la indicación de que el fichero lo abrimos para escritura, pero añadiendo sobre lo ya escrito sin borrarlo *"a+"*:

```
if ((fichero_basedatos = fopen("basedatos.txt", "a+")) == NULL){
      printf("Error: No se puede abrir el fichero %s\n",
             "basedatos.txt");
      exit(1);
}
```

- Inicializamos un array de caracteres llamado *cabecerasydatos* para escribir en él las cabeceras y los datos de tiempo extraídos, que posteriormente escribiremos en el fichero creado anteriormente. Para ello, primero se reserva memoria con la función *malloc* para un tamaño igual al número de caracteres que componen las cabeceras y los datos extraídos y después los introducimos con las funciones *strcpy* y *strcat*:

```
int longit = (23+strlen(resp->origen)+strlen(resp-
>destino)+strlen(resp->bus)+strlen(resp->metro)+strlen(resp-
>cercanias)+strlen(resp->bus_metro_cercanias)+strlen(resp-
>coche)+strlen(resp->pie));

datos = (char *)malloc(sizeof(char)*(longit+1));

strcpy(datos, resp->origen);
strcat(datos, " - ");
strcat(datos, resp->destino);
strcat(datos, " - ");
strcat(datos, resp->bus);
strcat(datos, " - ");
strcat(datos, resp->metro);
strcat(datos, " - ");
strcat(datos, resp->cercanias);
strcat(datos, " - ");
strcat(datos, resp->bus_metro_cercanias);
strcat(datos, " - ");
strcat(datos, resp->coche);
strcat(datos, " - ");
strcat(datos, resp->pie);
strcat(datos, "\n");
```

Como vemos, se introducen los datos con el formato que describimos a la hora de crear el fichero base de datos:

Origen - Destino - Bus - Metro - Cercanias - Bus_metro_cercanias - Coche - Pie

Estos datos se escriben a continuación de los que ya están almacenados en el fichero.

Ya sólo quedaría escribir con la función *fwrite* en el fichero base de datos abierto los datos extraídos, cerrar el puntero al fichero base de datos con la función *fclose* y liberar la memoria reservada con la función *free*:

```
fwrite(datos, sizeof(char), longit-1, fichero_basedatos);

fclose(fichero_basedatos);

free(datos);
```

- Hemos introducido los datos extraídos en el fichero base de datos. Lo que queda por hacer si la aplicación sigue el camino de que los puntos de origen y destino están en el fichero base de datos es crear la respuesta para el usuario. El procedimiento para ello es el mismo que ya se explicó anteriormente, es decir, creamos el fichero respuesta, reservamos memoria para un array con los datos de la respuesta y los introducimos en él, para posteriormente escribirlos en el fichero creado. Por último se cierra el puntero al fichero creado y se libera toda la memoria reservada.

Este ha sido el camino que seguiría la aplicación si no se encuentran los puntos de origen y destino en el fichero base de datos. Sin embargo, si cuando con la función siguiente:

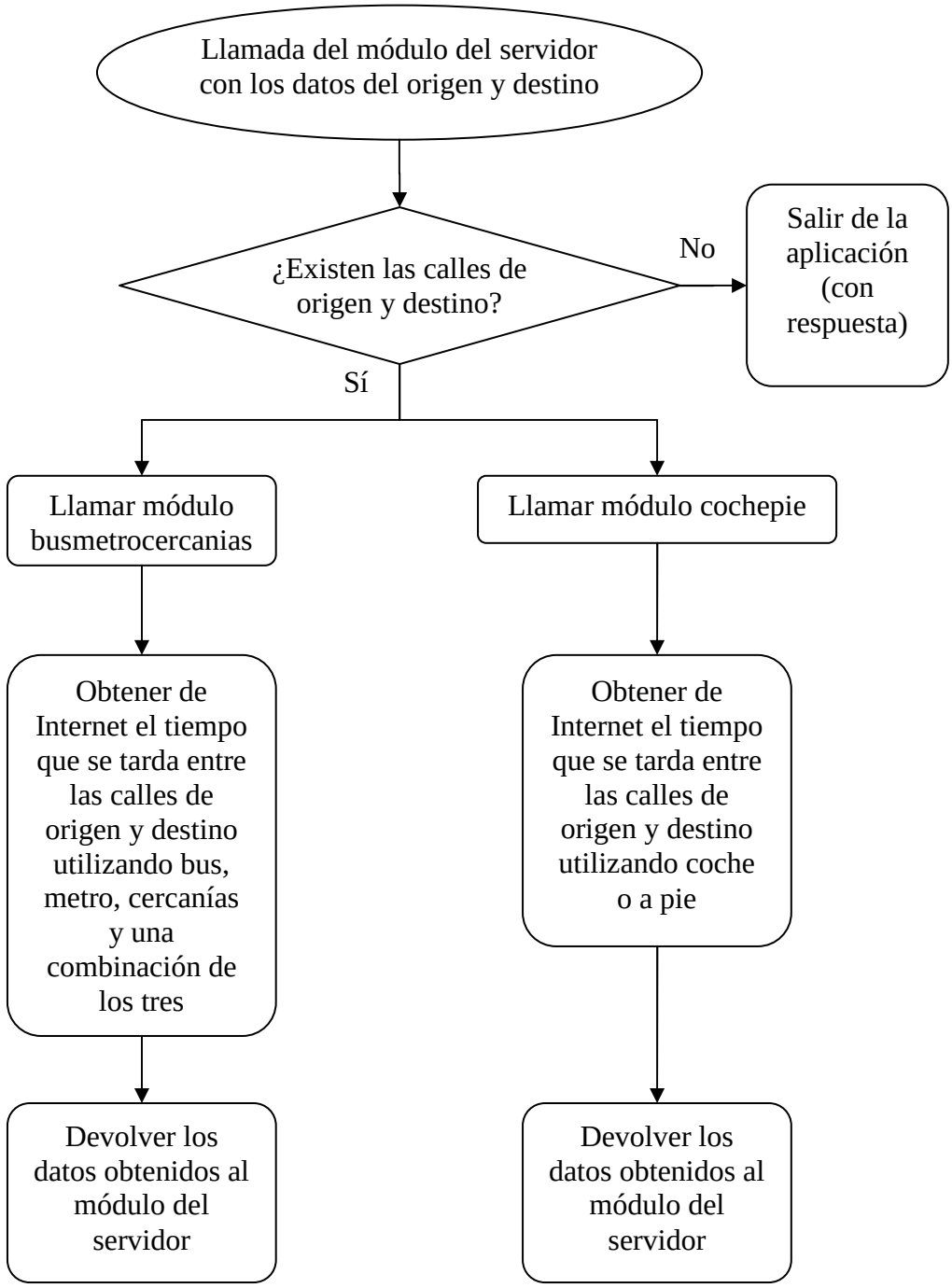
```
if ((encontrado = strstr(array_basedatos, busco)) == NULL)
```

obtenemos un puntero que no es NULL, significa que los datos de tiempo solicitados se encuentran almacenados en el fichero base de datos, con lo que los podemos obtener del fichero sin necesidad de llamar al módulo de extracción de información para obtenerlos de Internet. En este caso, la aplicación sigue el siguiente camino más sencillo, que es obtener los datos de tiempo solicitados que están almacenados en el fichero base de datos y crear la respuesta con ellos de la manera que ya hemos explicado varias veces, es decir, que creamos el fichero respuesta, reservamos memoria para un array con los datos de la respuesta y los introducimos en él, para posteriormente escribirlos en el fichero creado. Por último se cierra el puntero al fichero creado y se libera toda la memoria reservada.

Esta ha sido la explicación del método que se ha seguido para implementar el módulo del servidor, explicando detalladamente el diagrama que se mostró al principio de este apartado. En el siguiente apartado se explica con detalle la implementación que se ha hecho del módulo de extracción de la información.

3.6.2. Módulo de extracción de la información

Iniciamos la explicación del módulo de extracción de la información con un diagrama que resume el funcionamiento del mismo y después se explicará con más detalle su implementación.



A continuación se explica detalladamente cómo se han implementado todos los pasos del diagrama anterior. Es en la implementación del módulo de extracción de información donde más problemas he encontrado para programar la aplicación. Estos problemas y sus soluciones se especifican más adelante.

La llamada que hace el módulo del servidor al módulo de extracción de información es:

```
resp = extractor(peti);
```

por lo tanto, esta función recibe los datos de la petición correspondientes a los puntos de origen y destino entre los que se quiere calcular el tiempo de desplazamiento y devuelve dichos datos de tiempo extraídos de Internet.

Lo primero que hacemos con los datos recibidos es comprobar si existen las calles de origen y destino, ya que si no existen resultaría inútil seguir con la aplicación. Para ello nos servimos de un fichero llamado *calles* en el que tenemos almacenadas todas las calles de Madrid. Esta lista de las calles la hemos obtenido de una de las páginas Web de las que se realiza la extracción de información, http://www.ctm-madrid.es/como_ir_a/madrid/como_ir_a_madrid1.jsp. Hemos basado nuestra aplicación en las calles que tiene almacenadas esta página Web porque el tiempo de desplazamiento lo va a calcular solamente entre ellas. Puede que haya calles o puntos de Madrid que no estén en esta lista y que realmente existan en Madrid, pero como no se encuentran entre las calles almacenadas en esa lista, la página Web las ignoraría y nuestra aplicación no podría funcionar correctamente. Por lo tanto, hemos restringido la lista de calles a las que hemos obtenido de esa página Web, mientras que las no estén entre ellas, las trataremos como que no existen. De esta forma, para comprobar en nuestra aplicación si existen las calles de origen y destino, las buscamos en el fichero *calles*. En concreto, tenemos almacenadas 8962 calles en dicho fichero.

Abrimos el fichero *calles* con la función *fopen* en modo lectura "*r*":

```
if ((fichero_calles = fopen("calles", "r")) == NULL) {
    printf("Error: No se puede abrir el fichero %s\n",
            "calles");
    exit(1);
}
```

Para comprobar si las calles de origen y destino se encuentran en el fichero, utilizamos la función *strstr*:

```
if ((calle_origen = strstr(array_calles, cercaniasmetrobus-
>origen)) == NULL)

if ((calle_destino = strstr(array_calles, cercaniasmetrobus-
>destino)) == NULL) {
```

De esta forma, con la función *strstr* obtenemos un puntero a la primera aparición de las calles origen y destino en el fichero *calles*. Si la función devuelve un puntero NULL quiere decir que no las ha encontrado, por lo tanto la aplicación entiende que esas calles no existen. En ese caso, creamos la respuesta indicando que la calle en cuestión no existe y nos salimos de la aplicación. Esto lo implementamos de la misma forma que hemos hecho en el módulo del servidor. Creamos el fichero *respuesta.txt* en modo escritura "*w*", reservamos memoria para un array con los datos de la respuesta y los introducimos en él, para posteriormente escribirlos en el fichero creado. Por último se cierra el puntero al fichero creado y se libera toda la memoria reservada. El último

paso sería salir de la aplicación con la función *exit*, que implementa la salida del sistema directamente sin volver al módulo del servidor.

En el caso de que existan las calles de origen y destino, la aplicación sigue su camino. Lo siguiente sería ya extraer de Internet la información solicitada. Como tenemos dos páginas Web distintas de las que extraemos la información, hemos decidido implementar dos módulos independientes:

- Módulo busmetrocercanias: Este módulo servirá para extraer la información de tiempo de desplazamiento entre origen y destino utilizando bus, metro, cercanías y una combinación de los tres, para lo que nos serviremos de la página Web http://www.ctm-madrid.es/como_ir_a/madrid/como_ir_a_madrid1.jsp.

- Módulo cochepie: Este módulo servirá para extraer la información de tiempo de desplazamiento entre origen y destino utilizando coche o a pie, para lo que nos serviremos de la página Web www.mappy.com.

De esta forma, al ser módulos independientes, si hay que modificar algo de uno de ellos, no hay que "tocar" el otro. Además, si necesitamos ampliar la aplicación teniendo que acceder a otra página Web para buscar información distinta, o para utilizar otros medios de transporte, o cualquier asunto independiente de los módulos ya programados, se podría hacer sin tener que modificar nada de estos dos módulos.

En los apartados 3.6.2.1 y 3.6.2.2 se explica con detalle, y de manera independiente, cómo hemos implementado dichos módulos.

Estos dos módulos devuelven los datos de tiempo extraídos de Internet, por lo tanto, lo que queda por implementar en este módulo de extracción de información es almacenar esos datos extraídos en una estructura llamada *extraidos* y devolverlos a la función que implementa el módulo del servidor con la función *return*:

```
return(extraidos);
```

Con esto ya queda implementado el módulo de extracción de la información. Lo que queda por detallar es la implementación de los dos módulos independientes que lo forman.

3.6.2.1. Módulo busmetrocercanias

El objetivo de este módulo es extraer de Internet el tiempo que se tarda entre las calles de origen y destino utilizando bus, metro, cercanías y una combinación de los tres.

La llamada que se realiza a la función que implementa este módulo es la siguiente:

```
iti_bmc = busmetrocercanias(consulta);
```

La función la hemos llamado igual que el módulo *busmetrocercanías*, y tiene como argumento los datos de los puntos de origen y destino de la petición de entrada, mientras que devuelve los datos de tiempo extraídos de Internet correspondientes al tiempo que se tarda entre origen y destino utilizando bus, metro, cercanías y una combinación de los tres.

Lo que hay que implementar en esta función es simplemente acceder a Internet, extraer los datos de tiempo solicitados y devolverlos a la función principal del módulo de extracción de información, para que éste a su vez los retorne a la función del módulo del servidor.

Es en este punto donde hemos encontrado la mayor dificultad a la hora de programar la aplicación, ya que después de buscar y buscar no he encontrado ninguna función en el lenguaje de programación C para acceder a Internet u obtener el código fuente de una página Web. Por lo tanto, hay que buscar una solución alternativa.

Lo primero que encontré fue el lenguaje de programación Python, que es un lenguaje extensible y puede interactuar con programas en C. De esta forma, lo que pensé fue escribir un pequeño programa en Python que permitiese acceder a Internet y obtener la página Web en cuestión, y después llamar a ese programa desde el programa C principal.

Existen una serie de módulos en Python que permiten interactuar con Internet, como son *urllib* o *urllib2*. Estos módulos disponen de una serie de funciones que nos podrían servir, como son *urlopen*, *urlretrieve*, *URLopener*, que nos permiten abrir un objeto de red denotado por una URL para lectura o para copiarlo a un fichero. Esta parecía ser una solución que podría valerme. Sin embargo, una vez que me puse con ello, encontré muchas dificultades a la hora de aprender el nuevo lenguaje Python. No fui capaz de aprender a utilizarlo bien, y lo que es más importante, no sabía cómo hacerlo interactuar con mi programa en C, con lo que me puse a buscar otra solución. Seguí buscando algún lenguaje de programación que pudiese realizar algo parecido a Python, pero no lo encontré.

De nuevo buscando alguna función del lenguaje C que me pudiese servir, encontré una que no tiene nada que ver con interactuar con Internet, pero que me podría ser de utilidad, que es la función *system*. Esta función pasa una cadena de caracteres al entorno local para ser ejecutada por el "procesador de comandos" - también denominado "intérprete de comandos" - de una forma definida según la implementación. El proyecto se realiza en C sobre Linux, con lo que hay que utilizar la línea de comandos de Linux para ejecutar el programa. Por lo tanto, si existe algún comando de Linux que permita interactuar con Internet, mediante la función *system* podemos utilizar la línea de comandos para ejecutar dicho comando. De esta forma, lo siguiente es encontrar algún comando de Linux que permita acceder a Internet o extraer el código fuente de páginas Web.

Encontramos un comando llamado *wget*. **GNU Wget** es una herramienta de Software Libre que permite la descarga de contenidos desde servidores Web de una forma simple. Su nombre deriva de World Wide Web (w), y de «obtener» (en inglés *get*), esto quiere decir: *obtener desde WWW*. Soporta descargas mediante los protocolos HTTP, HTTPS y FTP. Permite la conversión de enlaces para la visualización de

contenidos HTML localmente. Es un programa utilizado a través de línea de comandos, principalmente en sistemas de UNIX/Linux, que está escrito en el lenguaje de programación C. El uso típico de **GNU Wget** consiste en invocar desde la línea de comandos una o más URLs como argumentos: *wget http://www.ejemplo.com/*.

De esta forma, con el comando *wget* podríamos obtener el código fuente de la página Web de la que queremos extraer el tiempo de desplazamiento entre origen y destino utilizando bus, metro, cercanías y una combinación de los tres. Como se explicó en apartados anteriores, esta página Web es http://www.ctm-madrid.es/como_ir_a/madrid/como_ir_a_madrid1.jsp:

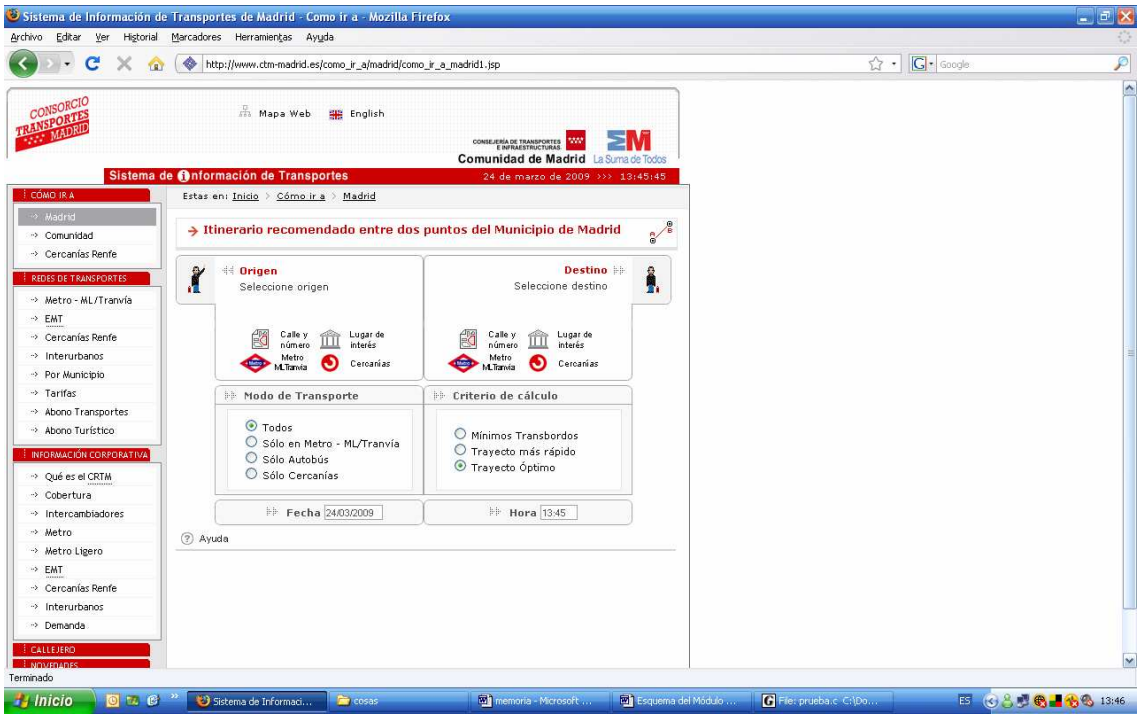


Figura 15. http://www.ctm-madrid.es/como_ir_a/madrid/como_ir_a_madrid1.jsp

Si nos fijamos bien en el cuadro del centro:

Figura 16. http://www.ctm-madrid.es/como_ir_a/madrid/como_ir_a_madrid1.jsp

en esta página tenemos que seleccionar el punto de origen, el punto de destino y el medio de transporte que queremos utilizar. Por lo tanto, para llegar a la página Web de la que queremos extraer el tiempo solicitado tenemos que navegar por varias páginas intermedias para seleccionar todos los datos de búsqueda anteriores.

Una vez que tenemos todos los datos de búsqueda seleccionados:

Figura 17. http://www.ctm-madrid.es/como_ir_a/madrid/como_ir_a_madrid1.jsp

aparece abajo y a la derecha un botón que es un enlace que nos lleva a la página Web definitiva donde se calcula el tiempo de desplazamiento basándose en los datos de búsqueda seleccionados, cuya URL en este caso de ejemplo sería: http://www.ctm-madrid.es/servlet/CalcItinerarioServlet?xh_ACCION=4&xh_TIPO=1&xh_URL=%2Fcomo_ir_a%2Fmadrid%2Fcomo_ir_a_madrid1.jsp&CODPANTALLA=4&xh_MEDIOS=3&xh_CRITERIO=2&xh_FECHA=24%2F03%2F2009&xh_HORA=13%3A54&x=48&y=11.

A la hora de utilizar el comando *wget*, ésta tendría que ser la URL que habría que introducirle como argumento para acceder a la página Web de la que extraer los datos de tiempo solicitados. Sin embargo, si nos fijamos en dicha URL, no aparece ningún parámetro indicando el punto de origen y el punto de destino seleccionados para llegar a esta página Web. Sí aparece el medio de transporte seleccionado en la variable xh_MEDIOS=3, que se corresponde con el autobús, pero no aparece ninguna variable con el origen o el destino. Por este motivo, no podemos utilizar el comando *wget*, ya que la dirección de la página Web que queremos no nos permite introducirle el origen y el destino en el argumento del comando *wget*.

Puesto que no podemos utilizar el comando *wget*, optamos por seguir buscando otro comando, y encontramos otro, llamado *lynx*. **Lynx** es un navegador Web en modo texto y que se puede utilizar en línea de comandos. Es un navegador típico de sistemas Linux o Unix, originalmente creado para este último sistema operativo, aunque también

está disponible para otros sistemas como Windows. Lynx sólo permite ver texto, lo que resulta una gran desventaja. No obstante las imágenes, que no se muestran se indican con texto, poniendo el nombre del archivo o el texto que haya en el atributo alt de la imagen, si es que se había definido un texto alternativo. Para navegar con Lynx se puede seleccionar un enlace con las teclas de dirección o, activando una opción para numerar los enlaces, ingresando el número de cada enlace. El uso típico de **Lynx** consiste en invocar desde la línea de comandos una URL como argumentos: *lynx http://www.ejemplo.com/*, apareciendo en la línea de comandos la página Web invocada en modo texto.

Con el comando *lynx* podríamos obtener el código fuente de la página Web de la que queremos extraer el tiempo de desplazamiento entre origen y destino utilizando bus, metro, cercanías y una combinación de los tres. Como se explicó en apartados anteriores, esta página Web es http://www.ctm-madrid.es/como_ir_a/madrid/como_ir_a_madrid1.jsp, que invocada con el comando *lynx* tiene el siguiente aspecto en la línea de comandos:

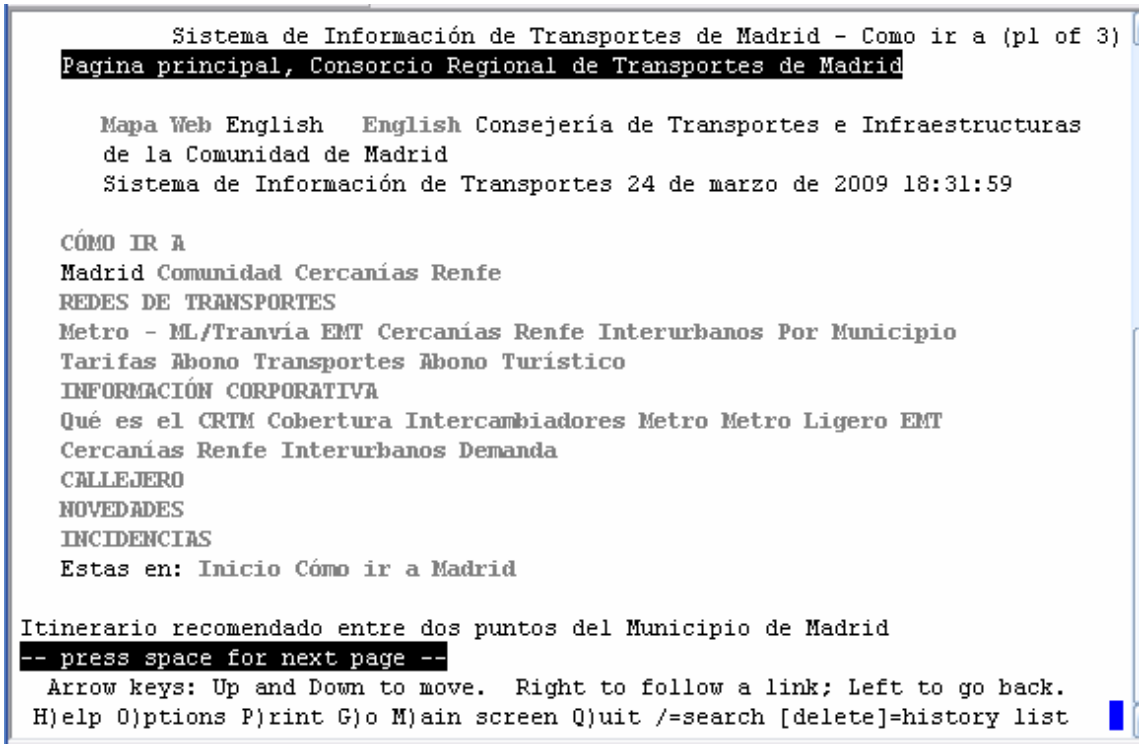


Figura 18. http://www.ctm-madrid.es/como_ir_a/madrid/como_ir_a_madrid1.jsp

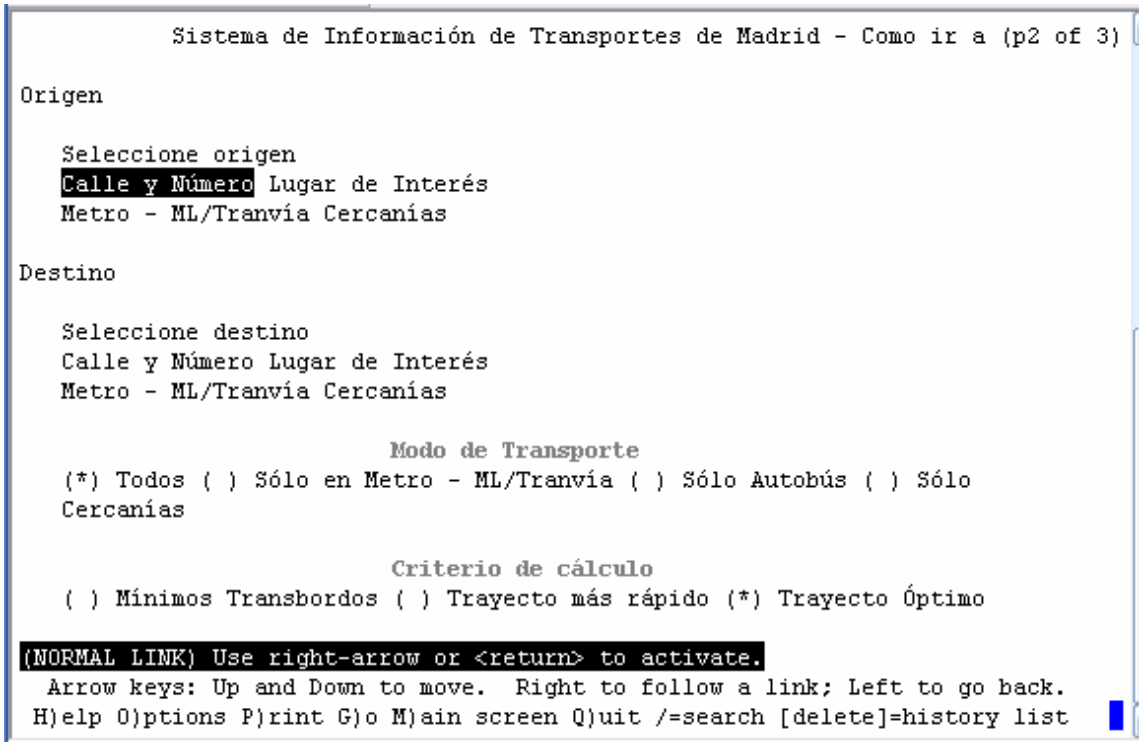


Figura 19. http://www.ctm-madrid.es/como_ir_a/madrid/como_ir_a_madrid1.jsp

Se puede comprobar que la página Web está en modo texto, de manera que para navegar por ella no se puede utilizar el ratón, sino que hay que utilizar las teclas de dirección. Para movernos hacia arriba y hacia abajo en la misma página, lo hacemos con las teclas de dirección arriba y abajo. Para acceder a un enlace, lo hacemos con la tecla de dirección derecha o la tecla Enter, y para volver de un enlace utilizamos la tecla de dirección izquierda.

Como vimos con el comando *wget*, para ir a la página Web final de la que hay que extraer los datos de tiempo solicitado hay que navegar para seleccionar el punto de origen, el punto de destino y el medio de transporte utilizado, y eso lo tenemos que hacer con un solo comando. Existe una opción del comando *lynx* que nos permite navegar y llegar a la página Web final de la que hay que extraer el tiempo solicitado. Esta opción es *-cmd_script=”nombre_fichero”*, que lee los comandos de teclado desde el fichero especificado. De esta forma, cada línea del fichero especificado contiene un comando correspondiente a pulsar una tecla del teclado y que tiene una misión a la hora de navegar con *lynx* por Internet. El comando *lynx* va leyendo uno por uno estos comandos y va ejecutando la acción que representan. Por lo tanto, lo que hacemos es crear un fichero donde cada línea se corresponde con el comando que representa la tecla que tenemos que pulsar para navegar desde la página Web de inicio hasta la página Web final de la que extraeremos la información de tiempo solicitada. El fichero creado se llama *comandos* y lo compone una combinación de los siguientes comandos:

Comando	Acción que realiza
Key Down Arrow	Ir hacia abajo en la misma página (tecla dirección abajo)
key Up Arrow	Ir hacia arriba en la misma página (tecla dirección arriba)
key Right Arrow	Acceder a un enlace (tecla dirección derecha)
key Left Arrow	Volver de un enlace (tecla dirección izquierda)
key q	Abandonar (tecla q)

key y	Aceptar (tecla y)
key ^J	Acceder a un enlace o Aceptar (tecla Enter)
key <delete>	Borrar (tecla borrar)
key "*letra*"	Escribir la letra "*letra*" (tecla letra "*letra*")

Tabla 3. Comandos que componen el fichero *comandos*

El fichero *comandos* está compuesto por una combinación de los comandos de la tabla 3. *lynx* los va leyendo uno a uno y va ejecutando la acción que representan para ir navegando desde la página Web inicial http://www.ctm-madrid.es/como_ir_a/madrid/como_ir_a_madrid1.jsp hasta la página Web final de la que extraeremos los datos de tiempo solicitados http://www.ctm-madrid.es/servlet/CalcItinerarioServlet?xh_ACCION=4&xh_TIPO=1&xh_URL=%2Fcomo_ir_a%2Fmadrid%2Fcomo_ir_a_madrid1.jsp&CODPANTALLA=4&xh_MEDIOS=3&xh_CRITERIO=2&xh_FECHA=24%2F03%2F2009&xh_HORA=13%3A54&x=48&y=11. Estos comandos también nos permiten guardar dicha página Web final en un fichero para que posteriormente analicemos su código fuente y extraigamos los datos de tiempo solicitados.

La secuencia de acciones que realiza *lynx* leyendo cada línea del fichero *comandos* que hemos creado es:

- Bajar en la página principal y entrar en el enlace de seleccionar origen, exactamente en el de calle y número.
- El origen lo tiene que seleccionar de una lista que contiene todas las calles de Madrid, bajando en dicha lista hasta llegar a la calle en cuestión.
- Una vez seleccionado el origen, se vuelve a la página principal, con lo que hay que volver a bajar y entrar en el enlace de seleccionar destino, exactamente en el de calle y número.
- De nuevo hay que bajar en la lista de todas las calles de Madrid hasta llegar a la calle en cuestión.
- Seleccionado el destino, aparece de nuevo la página principal, con lo que hay que volver a bajar en dicha página hasta seleccionar el medio de transporte. Primero seleccionamos hacer el trayecto con una combinación de autobús, metro y cercanías.
- Seleccionados todos los datos de búsqueda, accedemos al enlace que nos lleva a la página Web final que calcula el tiempo de desplazamiento y guardamos en un fichero llamado *tiempo_todos* el código fuente de dicha página.
- Volvemos hacia atrás, seleccionamos hacer el trayecto en autobús, accedemos al enlace de la página final y guardamos su código fuente en un fichero llamado *tiempo_bus*.
- Volvemos hacia atrás, seleccionamos hacer el trayecto en metro, accedemos al enlace de la página final y guardamos su código fuente en un fichero llamado *tiempo_metro*.
- Volvemos hacia atrás, seleccionamos hacer el trayecto en cercanías, accedemos al enlace de la página final y guardamos su código fuente en un fichero llamado *tiempo_cercanias*.

Como vemos, mediante el comando *lynx* con la opción -c*md_script=”nombre_fichero”* podemos obtener cuatro ficheros que almacenan el código fuente de las páginas Web que contienen el tiempo de desplazamiento entre origen y destino utilizando autobús, metro, cercanías y una combinación de los tres.

Todo este proceso lo hemos implementado de la siguiente manera:

- Creamos con la función *fopen* el fichero *comandos* en modo escritura “*w*”:

```
if ((fichero_comandos = fopen("comandos", "w")) == NULL) {
    printf("Error: No se puede abrir el fichero %s\n",
            "comandos");
    exit(1);
}
```

- Creamos las cadenas de caracteres correspondientes a los comandos que introduciremos en el fichero *comandos*:

```
char abajo[] = "key Down Arrow\n";
char arriba[] = "key Up Arrow\n";
char derecha[] = "key Right Arrow\n";
char izquierda[] = "key Left Arrow\n";
char abandonar[] = "key q\n";
char si[] = "key y\n";
char enter[] = "key ^J\n";
char borrar[] = "key <delete>\n";
char letra_p[] = "key p\n";
char letra_t[] = "key t\n";
char letra_i[] = "key i\n";
char letra_e[] = "key e\n";
char letra_m[] = "key m\n";
char letra_o[] = "key o\n";
char letra__[] = "key _\n";
char letra_d[] = "key d\n";
char letra_s[] = "key s\n";
char letra_r[] = "key r\n";
char letra_b[] = "key b\n";
char letra_u[] = "key u\n";
char letra_c[] = "key c\n";
char letra_a[] = "key a\n";
char letra_n[] = "key n\n";
```

- Escribimos en el fichero creado los comandos, para lo cual utilizamos la función *fwrite* de la siguiente manera:

```
fwrite(“comando”, sizeof(char), strlen(“comando”), fichero_comandos);
```

Con *fwrite* vamos escribiendo en el *fichero_comandos* cada cadena de caracteres correspondiente a cada “comando” necesario para navegar con *lynx*.

- Una vez creado el fichero *comandos*, lo siguiente es utilizar la función *system* para pasar a la línea de comandos el comando *lynx* con las opciones necesarias:

```
system("lynx -cmd_script=comandos -accept_all_cookies
http://www.ctm-madrid.es/como_ir_a/madrid/como_ir_a_madrid1.jsp");
```

Como vemos, el argumento de la función *system* es la cadena de caracteres que queremos invocar con la línea de comandos, que se compone del comando *lynx* y sus opciones, cuyo significado es el siguiente:

- *-cmd_script=comandos*: lee los comandos de teclado desde el fichero *comandos*.
- *-accept_all_cookies*: acepta todas las cookies necesarias para realizar la conexión con la página Web de inicio http://www.ctm-madrid.es/como_ir_a/madrid/como_ir_a_madrid1.jsp.
- *http://www.ctm-madrid.es/como_ir_a/madrid/como_ir_a_madrid1.jsp*: página Web que invoca el comando *lynx* desde la que empieza la navegación.

Hay una cuestión en esta implementación que hay que explicar más detalladamente. Como hemos mencionado anteriormente, cuando estamos navegando para seleccionar el origen o el destino, llegamos a una lista que contiene todas las calles de Madrid y tenemos que bajar por ella hasta seleccionar el origen o el destino en cuestión. Pero claro, hay que saber de antemano cuantas veces tenemos que bajar hasta llegar a ellos. Para ello nos servimos del fichero *calles* que hemos creado con todas las calles de Madrid, y que precisamente lo hemos obtenido de esta lista de calles de las que tenemos que seleccionar origen y destino. El fichero *calles* contiene una lista con todas las calles de Madrid exactamente igual a ésta. La implementación realizada es la siguiente:

- En la comprobación de si existían las calles de origen y destino, se utilizó la función *strstr* que devolvía un puntero a la primera ocurrencia de las calles origen y destino en el fichero *calles*:

```
if ((calle_origen = strstr(array_calles, cercaniasmetrobus-
>origen)) == NULL)

if ((calle_destino = strstr(array_calles, cercaniasmetrobus-
>destino)) == NULL) {
```

Por lo tanto, tenemos dos punteros, *calle_origen* y *calle_destino*, que apuntan a la calle origen y a la calle destino respectivamente en el fichero *calles*. Lo que verdaderamente ocurre es que con la función *strstr* nos quedamos con todos los caracteres del fichero calles que hay desde la calle origen, o desde la calle destino, hasta el final de dicho fichero, eliminando los anteriores, y los almacena en un array de caracteres apuntado por el puntero *calle_origen* o *calle_destino*.

- Lo siguiente es restar al número de caracteres del fichero *calles* el número de caracteres del array *calle_origen* o *calle_destino*. De esta forma, sabemos el número de caracteres que hemos pasado desde el inicio del fichero *calles* hasta llegar a la calle origen o destino, que se corresponden con los caracteres eliminados por la función *strstr*. Una vez hecha la resta, dividimos por 52, y esa sería la posición en que se encuentran la calle origen o la calle destino dentro del fichero calles, y por lo tanto, estas serían las veces que hay que bajar en la lista de las calles de Madrid a la que llegamos con el comando *lynx* para seleccionar la calle origen o la calle destino:

```
posicion_origen = (strlen(array_calles)-strlen(calle_origen))/52;
posicion_destino = (strlen(array_calles)-strlen(calle_destino))/52;
```

El motivo por el que dividimos entre 52 es porque cada línea que contiene el fichero *calles* está formada por 51 caracteres más uno del salto de línea (\n), y como cada línea se corresponde con una calle de esa lista, resulta que cada 52 caracteres tenemos una calle de la lista.

Una vez que ya tenemos los ficheros que contienen los datos de tiempo solicitados, lo siguiente es extraerlos para devolverlos a la función principal del módulo de extracción de información. Lo primero que hay que hacer es analizar el código fuente almacenado en dichos ficheros y encontrar dónde se sitúa el dato del tiempo que buscamos. Un fragmento de dichos ficheros se muestra en la figura 20:

```
Origen

Calle del Abedul, 1

  Caminar hasta Caminar hasta Ver Plano 3 min
  Habana 165 Habana 165
  Línea 14 Plaza Conde de Casal-Avenida de Pio XII Línea 14 Plaza Conde
de Casal-Avenida de Pio XII   32 min
  Recoletos 25 Recoletos 25
  Línea 37 Glorieta de Cuatro Caminos-Puente de Vallecas Línea 37
Glorieta de Cuatro Caminos-Puente de Vallecas   20 min
  Ciudad de Barcelona 144 Ciudad de Barcelona 144
  Caminar hasta Caminar hasta Ver Plano 1 min

Destino

Calle de Abtao, 1 Tiempo estimado del Trayecto Total: 56 min
```

Figura 20. Fragmento de fichero extraído

Como vemos en la última línea que se muestra en la figura, aparece el tiempo del trayecto total. Por lo tanto, lo que hacemos para encontrar dónde se sitúa el dato de tiempo buscado es localizar la palabra “Total” que se encuentra justo antes del dato de tiempo que queremos, ya que esta palabra “Total” es la única vez que aparece en el fichero. Una vez que obtenemos dónde se encuentra la palabra “Total”, sólo habría que desplazarse unos caracteres para obtener el tiempo solicitado.

En el caso de que no se pueda realizar el trayecto solicitado utilizando el medio de transporte seleccionado, el código almacenado en el fichero nos muestra la figura 21:

```
AVISO

   No hay camino posible con los datos introducidos
   Volver a la pantalla anterior Ayuda Nuevo itinerario
```

Figura 21. Fragmento de fichero extraído

En este caso, al no poder obtener el tiempo de desplazamiento entre origen y destino utilizando este medio de transporte, lo indicamos devolviendo un NO. Como observamos, en el código fuente obtenido aquí no aparece la palabra “Total”, por lo que para implementar este caso utilizamos este detalle.

La implementación de la extracción del tiempo solicitado de los ficheros obtenidos se realiza de la misma forma para los cuatro casos, correspondientes a utilizar autobús, metro, cercanías y una combinación de los tres. Por lo tanto, explicamos el caso de utilizar autobús y ya quedarían explicados los otros tres.

- Abrimos con la función *fopen* el fichero *tiempo_bus* en modo lectura *“r”*:

```
if ((fichero_bus = fopen("tiempo_bus", "r")) == NULL) {
    printf("Error: No se puede abrir el fichero %s\n",
            "tiempo_bus");
    exit(1);
}
```

- Obtenemos con la función *strstr* un puntero a la primera ocurrencia de la cadena "Total" en el array bus, ya que es donde nos indica la información de tiempo solicitada. Si devuelve un puntero NULL es porque no ha encontrado la cadena "Total", lo que significa que con ese medio de transporte no es posible hacer el trayecto solicitado. Almacenamos los datos solicitados que devolveremos a la función principal del módulo de extracción de información.

```
char *puntero_tiempo_bus;
if ((puntero_tiempo_bus = strstr(array_bus,"Total")) == NULL){
    datos_bmc->bus=(char *)malloc(sizeof(char)*2);
    strcpy(datos_bmc->bus,"NO");
}else{
    int h=0;
    do {
       h++;
       }
    while(puntero_tiempo_bus[h+7]!='n');
    if(h>6){
       datos_bmc->bus=(char *)malloc(sizeof(char)*5);
       strcpy(datos_bmc->bus,"9 min");
    }else{
       datos_bmc->bus = (char *)calloc(h+1,sizeof(char));
       h = 0;
       do {
          datos_bmc->bus[h]=puntero_tiempo_bus[h+7];
          h++;
          }
       while(puntero_tiempo_bus[h+7]!='n');
       datos_bmc->bus[h]=puntero_tiempo_bus[h+7];
}
```

Si nos fijamos en el código, tenemos implementado un caso extraño, en el que devolvemos como dato de tiempo 9 min. Este caso lo explicamos a partir de la figura 22:

```
Origen

Paseo del Prado, 1

  Caminar hasta Caminar hasta Ver Plano 2 min

Destino

Plaza de Cibeles, 1 Tiempo estimado del Trayecto Total:
```

Figura 22. Fragmento de fichero extraído

Como vemos, al estar origen y destino cerca, la aplicación muestra el tiempo que se tardaría entre ellos caminando a pie, con lo que al haber sólo una etapa entre las calles de origen y destino, no hay que sumar el tiempo empleado en las distintas etapas para obtener el tiempo total del trayecto, y ya no mostraría el tiempo que se tarda en el trayecto a continuación de la palabra "Total". Como nuestra aplicación se basa en la palabra "Total" para obtener el tiempo total del trayecto, en este caso no devolvería nada al no encontrar nada después de la palabra "Total". Lo que hemos hecho para implementar esto es que en estos casos, devuelva como tiempo de trayecto 9 min, ya que hemos hecho muchas pruebas con estos casos y la media de tiempos obtenidos es de 9 min.

Liberando la memoria utilizada en el módulo con la función *free* y devolviendo los datos de tiempo extraídos con la función *return*, finaliza la implementación del módulo *busmetrocercanias* correspondiente a obtener el tiempo que se tarda entre origen y destino utilizando autobús, metro, cercanías y una combinación de los tres:

```
free(array_calles);
free(array_bus);
free(array_metro);
free(array_cercanias);
free(array_todos);

return(datos_bmc);
```

3.6.2.2. Módulo cochepie

El objetivo de este módulo es extraer de Internet el tiempo que se tarda entre las calles de origen y destino utilizando coche y a pie.

La llamada que se realiza a la función que implementa este módulo es la siguiente:

```
iti_cp = cochepie(consulta);
```

La función la hemos llamado igual que el módulo coche, y tiene como argumento los datos de los puntos de origen y destino de la petición de entrada, mientras que devuelve los datos de tiempo extraídos de Internet correspondientes al tiempo que se tarda entre origen y destino utilizando coche y a pie.

Lo que hay que implementar en esta función es simplemente acceder a Internet, extraer los datos de tiempo solicitados y devolverlos a la función principal del módulo de extracción de información, para que éste a su vez los retorne a la función del módulo del servidor.

La página Web de la que queremos extraer el tiempo de desplazamiento entre origen y destino utilizando coche o a pie, como se explicó en apartados anteriores, es www.mappy.com:

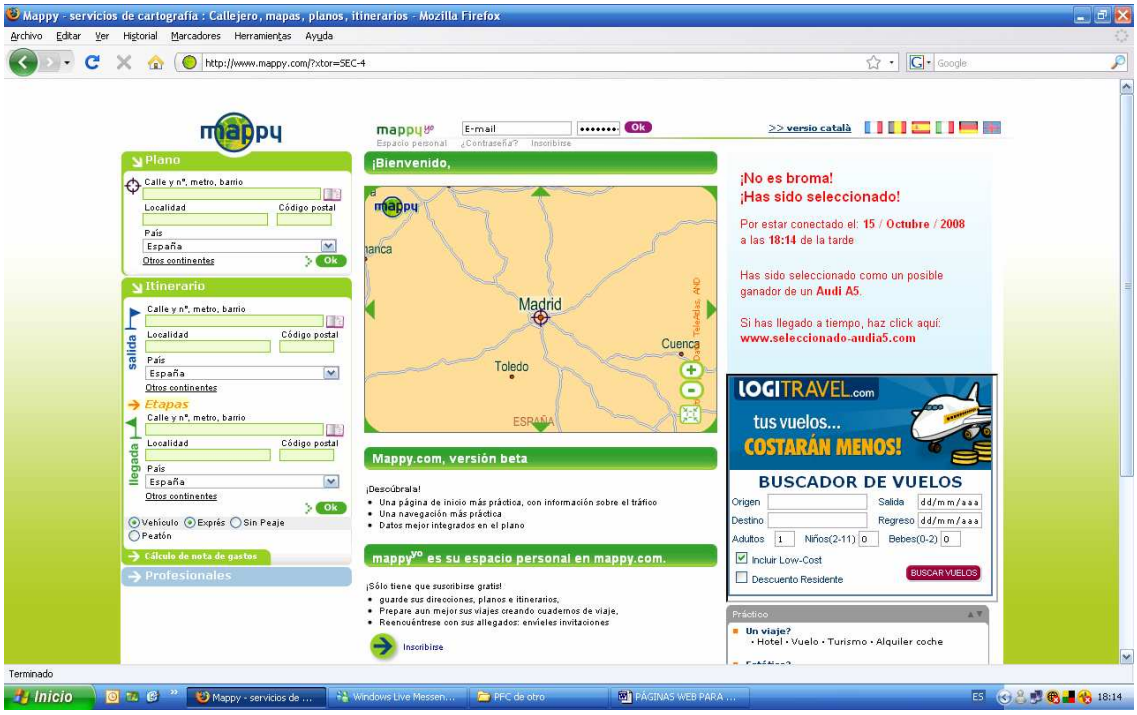


Figura 23. www.mappy.com

Si nos fijamos bien en el cuadro de la izquierda:

Figura 24. www.mappy.com

en esta página tenemos que seleccionar la calle de origen, la calle de destino y si realizamos el trayecto en vehículo o a pie. Por lo tanto, para llegar a la página Web de la que queremos extraer el tiempo solicitado tenemos que rellenar estos datos y acceder a ella pulsando el botón OK verde que aparece a la derecha y abajo del cuadro de la figura 24. Una vez realizado esto, llegamos a una página Web cuya URL en un caso de ejemplo sería:

http://www33.mappy.com/sid1NacfsjkBrbMn21w/CFGT?anchor=0&option=0&csl=i0%2Ci3%2Cv1%2Cv2%2Cv3%2Cv4%2Cv5%2Ci4&fsl=i0%2Ci3%2Cv1%2Cv2%2Cv3%2Cv4%2Cv5%2Ci4&gsl=i0%2Ci3%2Cv1%2Cv2%2Cv3%2Cv4%2Cv5%2Ci4&isl=i0&ivsl1=v1&ivsl2=v2&ivsl3=v3&ivsl4=v4&ivsl5=v5&iveh=car&isveh=sedcar&xsl=2&out=2&issl=i3&iesl=i4&show_poi=0&sveh=car&force_radius=&wci3=&noi3=&tci3=&a10i3=&xi3=&yi3=&wni3=Calle+de+Atocha&tni3=madrid&pci3=&sci3=&cci3=724&coi3=EU&force_radius=&wcv1=&nov1=&tcv1=&a10v1=&xv1=&yv1=&wnv1=&tnv1=&pcv1=&scv1=&ccv1=724&cov1=EU&force_radius=&wcv2=&nov2=&tcv2=&a10v2=&xv2=&yv2=&wnv2=&tnv2=&pcv2=&scv2=&ccv2=724&cov2=EU&force_radius=&wcv3=&nov3=&tcv3=&a10v3=&xv3=&yv3=&wnv3=&tnv3=&pcv3=&scv3=&ccv3=724&cov3=EU&force_radius=&wcv4=&nov4=&tcv4=&a10v4=&xv4=&yv4=&wnv4=&tnv4=&pcv4=&scv4=&ccv4=724&cov4=EU&force_radius=&wcv5=&nov5=&tcv5=&a10v5=&xv5=&yv5=&wnv5=&tnv5=&pcv5=&scv5=&ccv5=724&cov5=EU&order=0&force_radius=&wci4=&noi4=&tci4=&a10i4=&xi4=&yi4=&wni4=Calle+de+Serrano&tni4=madrid&pci4=&sci4=&cci4=724&coi4=EU&x=36&y=5&itimode=0&imode=0&iveh=car&isveh.car=sedcar&icvn=0&isveh.van=van&itra.van=3ax&isveh.tru=lt3.5&itra.tru=3ax&ipass=1&idist=km&ifkind=petrol&ifcost=1.12&idev=euro&ireimb=0.

Como explicamos en el apartado 3.6.2.1 en la implementación del módulo busmetrocercanias, el mayor problema nos lo encontramos en la manera de acceder desde el programa a la página Web de donde tenemos que extraer la información de tiempo solicitada. Para implementar el módulo busmetrocercanias utilizamos el comando *lynx* para solucionar este problema, sin embargo, para implementar el módulo cochepie vamos a utilizar el comando *wget*. El motivo por el que utilizamos este comando es que, si nos fijamos en la URL de la página Web final a la que tenemos que acceder para extraer los datos de tiempo solicitados, hay dos parámetros que se corresponden con la calle origen y la calle destino:

Calle origen: wni3=Calle+de+Atocha
Calle destino: wni4=Calle+de+Serrano

Además, tras varias pruebas, comprobamos que en esa URL también existen dos parámetros que indican si el trayecto se quiere realizar en coche o a pie:

En coche: itimode=0&imode=0
A pie: itimode=1&imode=2

Por lo tanto, todos los datos de búsqueda se encuentran en la URL de la página Web final de la que tenemos que extraer el tiempo del trayecto. De esta forma, para cada petición podemos invocar dicha página Web utilizando el comando *wget*, modificando únicamente en cada caso los parámetros de esa URL correspondientes a las

calles origen y destino y los parámetros que indican si el trayecto se hace en coche o a pie.

Otro problema que nos encontramos con esta página Web, es que cada petición la realiza el sistema a un servidor distinto, de manera que no se sobrecargue a uno solo con todas las peticiones. El servidor viene denominado por www33.mappy.com, donde cada servidor está identificado por www seguidas de un número. Pero el verdadero problema no es que haya distintos servidores, ya que podríamos hacer todas las peticiones al mismo servidor poniendo www seguidas del mismo número. El verdadero problema es que la aplicación inicia una sesión para cada petición, de manera que una vez pasado un tiempo, esa sesión se cierra y no la podríamos volver a utilizar. El identificador de sesión en la URL viene determinado por sid1NacfsjkBrbMn21w/, siendo realmente 1NacfsjkBrbMn21w/. Por lo tanto, para realizar distintas peticiones sobre el sistema necesitamos iniciar una sesión en cada petición. De esta forma, no nos importaría que la sesión de cada petición se cierre pasado un tiempo, ya que no la volveremos a utilizar. La solución que le damos a este problema es sencilla. Lo que hacemos es obtener primeramente el código fuente de la página Web www.mappy.com inicial, ya que, si nos fijamos en un fragmento de dicho código mostrado en la figura 25:

```
<title>Mappy - Road Guide</title>
<base href="http://www33.mappy.com/sid1NacfsjkBrbMn21w/">
<meta name="description" content="Mappy.com, your road and
<meta name="keywords" content="road guide, route planner, p
<META NAME="ROBOTS" CONTENT="INDEX, FOLLOW"><link rel="SHOF
<link rel="icon" type="image/ico" href="http://x.mappy.net/
<link rel="stylesheet" type="text/css" href="http://x.mappy
<link rel="stylesheet" type="text/css" href="http://x.mappy
<link rel="stylesheet" type="text/css" href="http://x.mappy
<link rel="stylesheet" type="text/css" href="http://x.mappy
<link rel="stylesheet" type="text/css" href="http://x.mappy

        var base_url_static = 'http://x.mappy.net';
        var JAVA_FLAG=0;
        var JAVA_FLAG_NORTON_TRAD=0;
        var JAVA_FLAG_NORTON_GUIDE=0;
        var mappy_server = 'http://www33.mappy.com';
        var mappy_sid = '1NacfsjkBrbMn21w/';
```

Figura 25. Fragmento de código fuente de www.mappy.com

Hemos marcado con un recuadro rojo las zonas donde se identifica el servidor utilizado y la sesión abierta para una nueva petición. De esta forma, lo que hacemos en cada petición al obtener primeramente el código fuente de la página Web www.mappy.com con *wget* es extraer de él el servidor y el identificador de la sesión que vamos a utilizar para crear la URL final de la que extraeremos el tiempo solicitado utilizando nuevamente *wget*, ya que esto se hará justo después de iniciar la sesión, sin preocuparnos de que pase tiempo y se cierre.

A continuación se detalla la implementación del módulo:

- Lo primero que hacemos es cambiar los espacios en blanco " " que contienen las cadenas de caracteres de las calles de origen y destino por el carácter "+", ya que, si nos fijamos en los parámetros de la URL que identifican el origen y el destino:

Calle origen: wni3=Calle+de+Atocha
Calle destino: wni4=Calle+de+Serrano

vemos cómo en lugar de espacios en blanco " " aparece el carácter "+". Por lo tanto, para introducir las cadenas de caracteres de las calles de origen y destino en la URL de la que extraeremos la información de tiempo solicitada tienen que tener este formato. Esto lo implementamos así:

```
int v;
for (v=0;v<strlen(piecoche->origen);v++) {
    if (piecoche->origen[v]==' ') piecoche->origen[v]='+';
}
int w;
for (w=0;w<strlen(piecoche->destino);w++) {
    if (piecoche->destino[w]==' ') piecoche->destino[w]='+';
}
```

Lo que hacemos es recorrer todos los caracteres de las calles de origen y destino y si encontramos un carácter en blanco " " lo sustituimos por un carácter "+".

- A continuación obtenemos el código fuente de la página www.mappy.com mediante el comando *wget*, utilizando la función *system* para pasarle el comando *wget* a la línea de comandos. Esto lo hacemos para obtener posteriormente el servidor y el identificador de sesión al que realizaremos la petición con los datos de búsqueda:

```
system("wget www.mappy.com -O intermedio");
```

Como vemos, el argumento de la función *system* es la cadena de caracteres que queremos invocar con la línea de comandos, que se compone del comando *wget* y sus opciones, cuyo significado es el siguiente:

- *www.mappy.com*: URL de la página Web que invoca el comando *lynx*.
- *-O intermedio*: con la opción -O guardamos en un fichero el código fuente de la página Web indicada en la opción anterior. En este caso, lo guardamos en un fichero llamado *intermedio*.

Del código fuente almacenado en el fichero *intermedio* vamos a obtener el servidor y el identificador de sesión al que vamos a realizar la petición.

- Abrimos con la función *fopen* el fichero *intermedio* en modo lectura "*r*":

```
if ((fichero_intermedio = fopen("intermedio", "r")) == NULL) {
    printf("Error: No se puede abrir el fichero %s\n",
            "intermedio");
    exit(1);
}
```

- Si nos fijamos en un fragmento del código almacenado en el fichero *intermedio*:

```
<title>Mappy - Road Guide</title>
<base href="http://www33.mappy.com/sid1NacfsjkBrbMn21w/">
<meta name="description" content="Mappy.com, your road and
```

Figura 26. Fragmento de código fuente almacenado en el fichero *intermedio*

en la etiqueta *base href* se encuentra la información del servidor y el identificador de la sesión que se ha iniciado al acceder a la página www.mappy.com con el comando *wget*, por lo tanto, es la sesión que vamos a utilizar para hacer la petición a dicho servidor. Para ello, tenemos que extraer esta información que aparece en la etiqueta *base href* para crear la URL de la página Web final de la que extraeremos la información de tiempo solicitada. Para implementar esto, nuevamente utilizamos la función *strstr*, de manera que nos devuelva un puntero a la primera ocurrencia de la cadena *base href* en el código fuente almacenado en el fichero *intermedio*:

```
base = strstr(array_intermedio,"base href");
```

A continuación reservamos memoria para un array de caracteres donde iremos almacenado los caracteres de la cadena que forman el servidor y el identificador de la sesión que utilizaremos para realizar la petición, que en este ejemplo serían *http://www33.mappy.com/sid1NacfsjkBrbMn21w/*:

```
int i=0;
while(base[i+11]!='"'){
      i++;
}
char *servidor = (char *)calloc(i,sizeof(char));
i=0;
while(base[i+11]!='"'){
      servidor[i]=base[i+11];
      i++;
}
```

Ya tenemos toda la información para crear la URL de la página Web final a la que realizaremos la petición para extraer la información de tiempo solicitada. Dicha URL tendrá el siguiente aspecto:

“SERVIDOR”CFGT?anchor=0&option=0&csl=i0%2Ci3%2Cv1%2Cv2%2Cv3%2Cv4%2Cv5%2Ci4&fsl=i0%2Ci3%2Cv1%2Cv2%2Cv3%2Cv4%2Cv5%2Ci4&gsl=i0%2Ci3%2Cv1%2Cv2%2Cv3%2Cv4%2Cv5%2Ci4&isl=i0&ivsl1=v1&ivsl2=v2&ivsl3=v3&ivsl4=v4&ivsl5=v5&iveh=car&isveh=sedcar&xsl=2&out=2&issl=i3&iesl=i4&show_poi=0&sveh=car&force_radius=&wci3=&noi3=&tci3=&a10i3=&xi3=&yi3=&wni3=**“ORIGEN”**&tni3=madrid&pci3=&sci3=&cci3=724&coi3=EU&force_radius=&wcv1=&nov1=&tcv1=&a10v1=&xv1=&yv1=&wnv1=&tnv1=&pcv1=&scv1=&ccv1=724&cov1=EU&force_radius=&wcv2=&nov2=&tcv2=&a10v2=&xv2=&yv2=&wnv2=&tnv2=&pcv2=&scv2=&ccv2=724&cov2=EU&force_radius=&wcv3=&nov3=&tcv3=&a10v3=&xv3=&yv3=&wnv3=&tnv3=&pcv3=&scv3=&ccv3=724&cov3=EU&force_radius=&wcv4=&nov4=&tcv4=&a10v4=&xv4=&yv4=&wnv4=&tnv4=&pcv4=&scv4=&ccv4=724&cov4=EU&force_radius=&wcv5=&nov5=&tcv5=&a10v5=&xv5=&yv5=&wnv5=&tnv5=&pcv5=&scv5=&ccv5=724&cov5=EU&order=0&force_radius=&wc

i4=&noi4=&tci4=&a10i4=&xi4=&yi4=&wni4=**"DESTINO"**&tni4=madrid&pci4=&sci4=&cci4=724&coi4=EU&x=36&y=5&itimode=**"0/1(COCHE/PIE)"**&imode=**"0/2(COCHE/PIE)"**&iveh=car&isveh.car=sedcar&icvn=0&isveh.van=van&itra.van=3ax&isveh.tru=lt3.5&itra.tru=3ax&ipass=1&idist=km&ifkind=petrol&ifcost=1.12&idev=euro&ireimb=0.

Lo que está en rojo es lo que varía en cada petición y lo tendremos que introducir en cada caso. La información del servidor la obtenemos como hemos explicado anteriormente, el origen y el destino lo obtenemos de la petición de entrada y el modo de transporte lo elegimos según queramos coche o a pie.

- Lo siguiente es crear la cadena de caracteres llamada *comando* formada por el comando *wget* y sus opciones para después enviarlo a la línea de comandos invocándola con la función *system*. Para ello, reservamos memoria con la función *calloc*[1] para dicha cadena de caracteres y almacenamos los datos en ella con las funciones *strcpy* y *strcat* como en ocasiones anteriores. Este procedimiento lo realizamos dos veces, una para obtener la información de tiempo realizando el trayecto a pie y otra utilizando el coche. En este primer caso, es para el trayecto realizado a pie:

```
char *comando = (char *)calloc(1129+strlen(piecoche-
>origen)+strlen(piecoche-
>destino)+strlen(servidor),sizeof(char));

strcpy(comando, "wget --post-data='anchor=0&option=0&
csl=i0%2Ci3%2Cv1%2Cv2%2Cv3%2Cv4%2Cv5%2Ci4&fsl=i0%2Ci3%2Cv1%2Cv2%
2Cv3%2Cv4%2Cv5%2Ci4&gsl=i0%2Ci3%2Cv1%2Cv2%2Cv3%2Cv4%2Cv5%2Ci4&is
l=i0&ivsl1=v1&ivsl2=v2&ivsl3=v3&ivsl4=v4&ivsl5=v5&iveh=car&isveh
=sedcar&xsl=2&out=2&issl=i3&iesl=i4&show_poi=0&sveh=car&force_ra
dius=&wci3=&noi3=&tci3=&a10i3=&xi3=&yi3=&wni3=");
strcat(comando, piecoche->origen);
strcat(comando,
"&tni3=madrid&pci3=&sci3=&cci3=724&coi3=EU&force_radius=&wcv1=&n
ov1=&tcv1=&a10v1=&xv1=&yv1=&wnv1=&tnv1=&pcv1=&scv1=&ccv1=724&cov
1=EU&force_radius=&wcv2=&nov2=&tcv2=&a10v2=&xv2=&yv2=&wnv2=&tnv2
=&pcv2=&scv2=&ccv2=724&cov2=EU&force_radius=&wcv3=&nov3=&tcv3=&a
10v3=&xv3=&yv3=&wnv3=&tnv3=&pcv3=&scv3=&ccv3=724&cov3=EU&force_r
adius=&wcv4=&nov4=&tcv4=&a10v4=&xv4=&yv4=&wnv4=&tnv4=&pcv4=&scv4
=&ccv4=724&cov4=EU&force_radius=&wcv5=&nov5=&tcv5=&a10v5=&xv5=&y
v5=&wnv5=&tnv5=&pcv5=&scv5=&ccv5=724&cov5=EU&order=0&force_radiu
s=&wci4=&noi4=&tci4=&a10i4=&xi4=&yi4=&wni4=");
strcat(comando, piecoche->destino);
strcat(comando,
"&tni4=madrid&pci4=&sci4=&cci4=724&coi4=EU&x=0&y=0&itimode=1&imo
de=2&iveh=car&isveh.car=sedcar&icvn=0&isveh.van=van&itra.van=3ax
&isveh.tru=lt3.5&itra.tru=3ax&ipass=1&idist=km&ifkind=petrol&ifc
ost=1.531&idev=euro&ireimb=0' ");
strcat(comando, servidor);
strcat(comando, "CFGT -O datospie");
```

Para aclarar lo hecho anteriormente, mostramos la cadena de caracteres que se forma con la implementación anterior:

[1] La función *calloc* también reserva memoria como hicimos con la función *malloc*, pero con la diferencia de que *calloc* inicializa a cero la memoria reservada.

```
      wget --post-data='anchor=0&option=0&
csl=i0%2Ci3%2Cv1%2Cv2%2Cv3%2Cv4%2Cv5%2Ci4&fsl=i0%2Ci3%2Cv1%2Cv2%2Cv3%2
Cv4%2Cv5%2Ci4&gsl=i0%2Ci3%2Cv1%2Cv2%2Cv3%2Cv4%2Cv5%2Ci4&isl=i0&ivsl1=v
1&ivsl2=v2&ivsl3=v3&ivsl4=v4&ivsl5=v5&iveh=car&isveh=sedcar&xsl=2&out=
2&issl=i3&iesl=i4&show_poi=0&sveh=car&force_radius=&wci3=&noi3=&tci3=&
a10i3=&xi3=&yi3=&wni3="ORIGEN"&tni3=madrid&pci3=&sci3=&cci3=724&coi3=E
U&force_radius=&wcv1=&nov1=&tcv1=&a10v1=&xv1=&yv1=&wnv1=&tnv1=&pcv1=&s
cv1=&ccv1=724&cov1=EU&force_radius=&wcv2=&nov2=&tcv2=&a10v2=&xv2=&yv2=
&wnv2=&tnv2=&pcv2=&scv2=&ccv2=724&cov2=EU&force_radius=&wcv3=&nov3=&tc
v3=&a10v3=&xv3=&yv3=&wnv3=&tnv3=&pcv3=&scv3=&ccv3=724&cov3=EU&force_ra
dius=&wcv4=&nov4=&tcv4=&a10v4=&xv4=&yv4=&wnv4=&tnv4=&pcv4=&scv4=&ccv4=
724&cov4=EU&force_radius=&wcv5=&nov5=&tcv5=&a10v5=&xv5=&yv5=&wnv5=&tnv
5=&pcv5=&scv5=&ccv5=724&cov5=EU&order=0&force_radius=&wci4=&noi4=&tci4
=&a10i4=&xi4=&yi4=&wni4="DESTINO"&tni4=madrid&pci4=&sci4=&cci4=724&coi
4=EU&x=0&y=0&itimode=1&imode=2&iveh=car&isveh.car=sedcar&icvn=0&isveh.
van=van&itra.van=3ax&isveh.tru=lt3.5&itra.tru=3ax&ipass=1&idist=km&ifk
ind=petrol&ifcost=1.531&idev=euro&ireimb=0' "SERVIDOR"CFGT -O datospie
```

Ésta es la cadena de caracteres que queremos invocar con la línea de comandos, que se compone del comando *wget* y sus opciones, cuyo significado es el siguiente:

- *--post-data= 'anchor=0&option=0&csl=i0%2Ci3%2Cv1%2Cv2%2Cv3%2Cv4%2Cv5%2Ci4&fsl=i0%2Ci3%2Cv1%2Cv2%2Cv3%2Cv4%2Cv5%2Ci4&gsl=i0%2Ci3%2Cv1%2Cv2%2Cv3%2Cv4%2Cv5%2Ci4&isl=i0&ivsl1=v1&ivsl2=v2&ivsl3=v3&ivsl4=v4&ivsl5=v5&iveh=car&isveh=sedcar&xsl=2&out=2&issl=i3&iesl=i4&show_poi=0&sveh=car&force_radius=&wci3=&noi3=&tci3=&a10i3=&xi3=&yi3=&wni3="ORIGEN"&tni3=madrid&pci3=&sci3=&cci3=724&coi3=EU&force_radius=&wcv1=&nov1=&tcv1=&a10v1=&xv1=&yv1=&wnv1=&tnv1=&pcv1=&scv1=&ccv1=724&cov1=EU&force_radius=&wcv2=&nov2=&tcv2=&a10v2=&xv2=&yv2=&wnv2=&tnv2=&pcv2=&scv2=&ccv2=724&cov2=EU&force_radius=&wcv3=&nov3=&tcv3=&a10v3=&xv3=&yv3=&wnv3=&tnv3=&pcv3=&scv3=&ccv3=724&cov3=EU&force_radius=&wcv4=&nov4=&tcv4=&a10v4=&xv4=&yv4=&wnv4=&tnv4=&pcv4=&scv4=&ccv4=724&cov4=EU&force_radius=&wcv5=&nov5=&tcv5=&a10v5=&xv5=&yv5=&wnv5=&tnv5=&pcv5=&scv5=&ccv5=724&cov5=EU&order=0&force_radius=&wci4=&noi4=&tci4=&a10i4=&xi4=&yi4=&wni4="DESTINO"&tni4=madrid&pci4=&sci4=&cci4=724&coi4=EU&x=0&y=0&itimode=1&imode=2&iveh=car&isveh.car=sedcar&icvn=0&isveh.van=van&itra.van=3ax&isveh.tru=lt3.5&itra.tru=3ax&ipass=1&idist=km&ifkind=petrol&ifcost=1.531&idev=euro&ireimb=0'*: La opción --post-data se utiliza para enviar los datos necesarios para rellenar cuestionarios HTTP que usan el método POST para ello. En la URL completa, estos datos son los que están a continuación del carácter "*?*", que separa la dirección de la página Web de los datos del cuestionario, y es la manera con la que se envían los datos del cuestionario a rellenar utilizando el método POST. Los datos tendrán el formato "*nombre=valor*" unidos todos por el carácter "*&*" sin espacios en blanco. Con la opción --post-data enviamos los datos del origen en el parámetro *wni3*, del destino en el parámetro *wni4* e indicamos que queremos realizar el trayecto a pie con los parámetros *itimode=1&imode=2*.

- *"SERVIDOR"CFGT*: Con esta opción indicamos la dirección de la página Web que invocamos con el comando *wget* y es a la que realizamos la petición para obtener el tiempo solicitado. En la URL completa, esta dirección está antes del carácter *"?"*, que la separa de los datos enviados para rellenar el cuestionario utilizando el método POST de la opción anterior.
- *-O datospie*: con la opción -O guardamos en un fichero el código fuente de la página Web indicada en la opción anterior. En este caso, lo guardamos en un fichero llamado *datospie*.

Una vez creada la cadena de caracteres *comando* que incluye el comando *wget* y sus opciones, la mandamos a la línea de comandos con la función *system* para que la ejecute:

```
system(comando);
```

Esta ha sido la implementación para extraer, y almacenar en el fichero *datospie*, el código fuente de la página Web final de la que extraeremos el tiempo de desplazamiento entre origen y destino realizando el trayecto a pie. Ahora tenemos que hacer lo mismo, pero para el caso de realizar el trayecto en coche. En este caso, la cadena de caracteres que creamos se llama *comando2* y es la siguiente:

```
wget --post-data='anchor=0&option=0&
csl=i0%2Ci3%2Cv1%2Cv2%2Cv3%2Cv4%2Cv5%2Ci4&fsl=i0%2Ci3%2Cv1%2Cv2%
2Cv3%2Cv4%2Cv5%2Ci4&gsl=i0%2Ci3%2Cv1%2Cv2%2Cv3%2Cv4%2Cv5%2Ci4&isl=i0&i
vsl1=v1&ivsl2=v2&ivsl3=v3&ivsl4=v4&ivsl5=v5&iveh=car&isveh=sedcar&xsl=
2&out=2&issl=i3&iesl=i4&show_poi=0&sveh=car&force_radius=&wci3=&noi3=&
tci3=&a10i3=&xi3=&yi3=&wni3="ORIGEN"&tni3=madrid&pci3=&sci3=&cci3=724&
coi3=EU&force_radius=&wcv1=&nov1=&tcv1=&a10v1=&xv1=&yv1=&wnv1=&tnv1=&p
cv1=&scv1=&ccv1=724&cov1=EU&force_radius=&wcv2=&nov2=&tcv2=&a10v2=&xv2
=&yv2=&wnv2=&tnv2=&pcv2=&scv2=&ccv2=724&cov2=EU&force_radius=&wcv3=&no
v3=&tcv3=&a10v3=&xv3=&yv3=&wnv3=&tnv3=&pcv3=&scv3=&ccv3=724&cov3=EU&fo
rce_radius=&wcv4=&nov4=&tcv4=&a10v4=&xv4=&yv4=&wnv4=&tnv4=&pcv4=&scv4=
&ccv4=724&cov4=EU&force_radius=&wcv5=&nov5=&tcv5=&a10v5=&xv5=&yv5=&wnv
5=&tnv5=&pcv5=&scv5=&ccv5=724&cov5=EU&order=0&force_radius=&wci4=&noi4
=&tci4=&a10i4=&xi4=&yi4=&wni4="DESTINO"&tni4=madrid&pci4=&sci4=&cci4=7
24&coi4=EU&x=0&y=0&itimode=0&imode=0&iveh=car&isveh.car=sedcar&icvn=0&
isveh.van=van&itra.van=3ax&isveh.tru=lt3.5&itra.tru=3ax&ipass=1&idist=
km&ifkind=petrol&ifcost=1.531&idev=euro&ireimb=0' "SERVIDOR"CFGT -O
datoscoche
```

Como vemos, está formada por el comando *wget* y sus opciones, que son:

- Con la opción --post-data enviamos los datos del origen en el parámetro *wni3*, del destino en el parámetro *wni4* e indicamos que queremos realizar el trayecto con coche con los parámetros *itimode=0&imode=0*.
- Con la opción *"SERVIDOR"CFGT* indicamos la dirección de la página Web que invocamos con el comando *wget* y es a la que realizamos la petición para obtener el tiempo solicitado.
- *-O datospie*: con la opción -O guardamos en un fichero el código fuente de la página Web indicada en la opción anterior. En este caso, lo guardamos en un fichero llamado *datoscoche*.

Una vez creada la cadena de caracteres *comando2* que incluye el comando *wget* y sus opciones, la mandamos a la línea de comandos con la función *system* para que la ejecute:

```
system(comando2);
```

Esta ha sido la implementación para extraer, y almacenar en el fichero *datoscoche*, el código fuente de la página Web final de la que extraeremos el tiempo de desplazamiento entre origen y destino realizando el trayecto en coche.

- Una vez que ya tenemos los ficheros que contienen los datos de tiempo solicitados, lo siguiente es extraerlos para devolverlos a la función principal del módulo de extracción de información. Lo primero que hay que hacer es analizar el código fuente almacenado en dichos ficheros y encontrar dónde se sitúa el dato del tiempo que buscamos. Un fragmento de dichos ficheros se muestra en la figura 27:

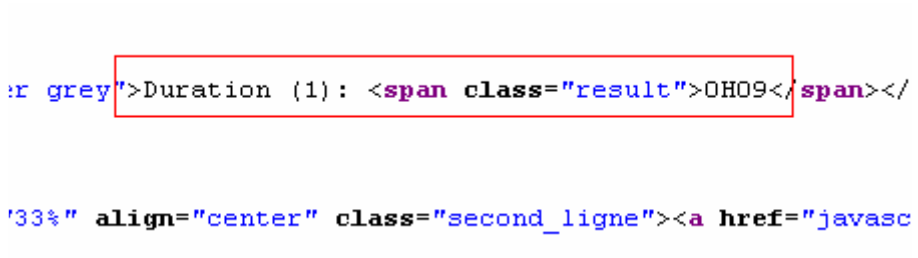


Figura 27. Fragmento de fichero extraído

Como vemos en el recuadro rojo en una de las líneas que se muestra en la figura, aparece el tiempo del trayecto total. Por lo tanto, lo que hacemos para encontrar dónde se sitúa el dato de tiempo buscado es localizar la palabra “Duration” que se encuentra justo antes del dato de tiempo que queremos, ya que esta palabra “Duration” es la única vez que aparece en el fichero. Una vez que obtenemos dónde se encuentra la palabra “Duration”, sólo habría que desplazarse unos caracteres para obtener el tiempo solicitado.

La implementación de la extracción del tiempo solicitado de los ficheros obtenidos se realiza de la misma forma para los dos casos, correspondientes a realizar el trayecto a pie y en coche. Por lo tanto, explicamos el caso de realizar el trayecto a pie y ya quedaría explicado el otro caso.

- Abrimos con la función *fopen* el fichero *datospie* en modo lectura “*r*”:

```
if ((fichero_pie = fopen("datospie", "r")) == NULL) {
    printf("Error: No se puede abrir el fichero %s\n",
            "datospie");
    exit(1);
}
```

- Obtenemos con la función *strstr* un puntero a la primera ocurrencia de la cadena "Duration" en el array pie, ya que es donde nos indica la información de tiempo solicitada:

```
puntero_tiempo = strstr(array_pie,"Duration");
```

- Almacenamos los datos de tiempo solicitados, que devolveremos a la función principal del módulo de extracción de información:

```
cp *datos_cp = (cp *)calloc(1,sizeof(cp));

int x=0;
while(puntero_tiempo[x+35]!='<'){
   x++;
}
datos_cp->pie = (char *)calloc(x,sizeof(char));
x=0;
while(puntero_tiempo[x+35]!='<'){
   datos_cp->pie[x]=puntero_tiempo[x+35];
   x++;
}
```

- Antes de acabar, volvemos a cambiar los caracteres "+" por caracteres en blanco " " de los campos origen y destino:

```
for (v=0;v<strlen(piecoche->origen);v++) {
    if (piecoche->origen[v]=='+') piecoche->origen[v]=' ';
}
for (w=0;w<strlen(piecoche->destino);w++) {
    if (piecoche->destino[w]=='+') piecoche->destino[w]=' ';
}
```

- Por último, liberamos con la función *free* la memoria reservada y devolvemos los datos de tiempo en coche y a pie extraídos a la función principal del módulo de extracción de información con la función *return*:

```
return(datos_cp);
```

Con esto finaliza la implementación del módulo *cochepie* correspondiente a obtener el tiempo que se tarda entre origen y destino en coche y a pie.

4. RESULTADOS EXPERIMENTALES

4.1. Introducción

En este apartado se exponen los resultados experimentales que se obtienen con la implementación del proyecto realizado. Se muestra la respuesta que crea la aplicación implementada a partir de la petición de entrada, tanto para peticiones correctas como para peticiones que resultan erróneas para el sistema. Además, se indica en todo caso la evolución del fichero base de datos que crea la aplicación, ya que para cada petición que se hace, éste sufre modificaciones.

En este caso, la medida del correcto funcionamiento del sistema no viene determinada por el tiempo de respuesta del sistema, ya que éste varía en función de la petición realizada. Si es una petición nueva, es decir, cuyos puntos de origen y destino no se han solicitado nunca, el sistema tarda más en dar la respuesta porque tiene que navegar por Internet para extraer los datos solicitados, mientras que si la petición ya se ha realizado anteriormente, el sistema obtiene los datos del fichero base de datos sin tener que navegar por Internet, con lo que el tiempo empleado para devolver los resultados es menor. Por lo tanto, los medida del correcto funcionamiento del sistema viene determinado por la corrección de los resultados devueltos, es decir, que el sistema devuelva correctamente los datos de tiempo solicitados, ya que esto significa que la aplicación navega correctamente por Internet y extrae perfectamente los datos de la Web, además del correcto funcionamiento del sistema Wrapper-Mediador, en el que el fichero base de datos es fundamental.

Además, con los resultados obtenidos se podrá comprobar si las páginas Web de las que se realiza la extracción de la información son fiables y ofrecen buenos resultados, ya que si esto no es así, la aplicación no funciona correctamente.

4.2. Resultados

La petición de información se realiza mediante un fichero XML, cuyo formato es el siguiente:

```
<?xml version="1.0" encoding="UTF-8"?>
<Peticion>
        <Origen></Origen>
        <Destino></Destino>
</Peticion>
```

La petición de entrada es muy simple, ya que solamente hay que especificar el lugar de origen y el lugar de destino entre los que se quiere hallar el tiempo de desplazamiento.

Podemos comprobar que se trata de un fichero XML. La primera línea es obligatoria y nos dice que es un documento XML. Además nos muestra dos atributos, indicándonos que la versión de XML usada en el documento es la 1.0 y que la forma en que se ha codificado el documento es UTF-8. A continuación aparecen las etiquetas que componen el documento. La primera es la etiqueta raíz, que es la que está precedida por el prólogo e incluye el documento por completo, y se llama <Peticion>, indicándonos que este fichero es la petición de entrada. Al final del documento está la etiqueta de

cierre de dicha etiqueta raíz. El documento lo forman las etiquetas <Origen> y <Destino> y sus correspondientes etiquetas de cierre, cuyos elementos nos indican los puntos origen y destino entre las que se quiere obtener el tiempo de desplazamiento.

La respuesta que ofrece el módulo del servidor al usuario va también en un fichero XML, cuyo formato es el siguiente:

```
<?xml version="1.0" encoding="UTF-8"?>
<Respuesta>
        <Origen></Origen>
        <Destino></Destino>
        <Bus></Bus>
        <Metro></Metro>
        <Cercanias></Cercanias>
        <Bus_Metro_Cercanias></Bus_Metro_Cercanias>
        <Coche></Coche>
        <Pie></Pie>
</Respuesta>
```

Comprobamos que se trata de un fichero XML, ya que contiene la primera línea obligatoria que nos dice que es un documento XML. Además nos muestra dos atributos, indicándonos que la versión de XML usada en el documento es la 1.0 y que la forma en que se ha codificado el documento es UTF-8. A continuación aparecen las etiquetas que componen el documento. La primera es la etiqueta raíz, que es la que está precedida por el prólogo e incluye el documento por completo, y se llama <Respuesta>, indicándonos que este fichero es la respuesta de salida. Al final del documento está la etiqueta de cierre de dicha etiqueta raíz. El documento lo forman las etiquetas <Origen>, <Destino>, <Bus>, <Metro>, <Cercanias>, <Bus_Metro_Cercanias>, <Coche> y <Pie> y sus correspondientes etiquetas de cierre. Las dos primeras se corresponden con los mismos campos de origen y destino entre los que se quiere hallar el tiempo de desplazamiento que la petición de entrada. El resto se corresponden con medios de trasporte, indicándonos el tiempo que se tarda en hacer el trayecto utilizando dicho medio de transporte. Es decir, hay una etiqueta que indica el tiempo que se tarda en hacer el trayecto en autobús (<Bus>), otra en metro (<Metro>), otra en cercanías (<Cercanias>), otra utilizando una combinación de los tres anteriores (<Bus_Metro_Cercanias>), otra en coche (<Coche>) y otra a pie (<Pie>). La aplicación implementada devuelve el tiempo que se tarda en hacer el trayecto deseado para cada uno de los medios de transporte, de manera que el usuario pueda elegir cuál le viene mejor. Cuando alguna de las etiquetas devuelve valor NO quiere decir que no se puede realizar ese trayecto con ese medio de transporte, por lo que no se puede devolver el tiempo.

A continuación se muestran los resultados que se obtienen para distintas peticiones realizadas al sistema, mostrando el fichero de respuesta que se crea y la evolución del fichero base de datos.

La primera petición que hacemos al sistema tiene como origen la Calle de Atocha y como destino la Calle de Serrano. En este caso, la petición es la que se muestra en la figura 28:

```
<?xml version="1.0" encoding="UTF-8"?>
<Peticion>
    <Origen>Calle de Atocha</Origen>
    <Destino>Calle de Serrano</Destino>
</Peticion>
```

Figura 28. Petición de entrada

Como vemos, está formada por la primera línea obligatoria en cualquier documento XML, y a continuación aparecen las etiquetas que forman el documento. La primera es la que indica que es una petición y las dos siguientes son las que contienen las calles de origen y destino entre las que se quiere hallar el tiempo de desplazamiento utilizando los distintos medios de transporte públicos. La etiqueta de origen contiene la Calle de Atocha y la etiqueta de destino contiene la Calle de Serrano.

La respuesta que devuelve el sistema a la petición anterior se muestra en la figura 29:

```
<?xml version="1.0" encoding="UTF-8"?>
<Respuesta>
    <Origen>Calle de Atocha</Origen>
    <Destino>Calle de Serrano</Destino>
    <Bus>20 min</Bus>
    <Metro>19 min</Metro>
    <Cercanias>NO</Cercanias>
    <Bus_Metro_Cercanias>19 min</Bus_Metro_Cercanias>
    <Coche>0H09</Coche>
    <Pie>0H55</Pie>
</Respuesta>
```

Figura 29. Respuesta del sistema

Podemos comprobar que es un documento XML perfectamente formado, que está compuesto por la primera línea obligatoria en cualquier documento XML y las etiquetas que componen el documento. La primera etiqueta es la que indica que es una respuesta. El resto de etiquetas nos muestran los datos de búsqueda y los datos de tiempo extraídos de Internet. Vemos que las etiquetas de origen y destino son las mismas que en la petición de entrada, que son la Calle de Atocha y la Calle de Serrano respectivamente. El resto de etiquetas nos muestran el tiempo que se tarda entre el origen y el destino utilizando los distintos medios de transporte. En autobús se tarda 20 minutos, en metro 19 minutos y en cercanías el dato devuelto es NO, lo que quiere decir que no se puede realizar este trayecto con este transporte. Utilizando una combinación de autobús, metro y cercanías se tarda 19 minutos. En coche se puede llegar en 9 minutos y a pie 55 minutos. El sistema ha devuelto perfectamente los datos de tiempo solicitados en la respuesta, con lo que el usuario los puede utilizar para elegir qué medio de transporte le conviene mejor para realizar el trayecto.

Si nos fijamos en los datos de tiempo devueltos en la respuesta, se aprecia que el formato utilizado para indicar el tiempo que se tarda desplazándose en autobús, metro, cercanías y una combinación de los tres es distinto al formato utilizado para indicar el tiempo cuando nos desplazamos en coche y a pie. Esto lo hemos implementado así debido a que empleamos una página Web para extraer la información de tiempo

utilizando autobús, metro, cercanías y una combinación de los tres y otra página Web para los casos de utilizar coche o a pie. Debido a esto, al ser páginas Web diferentes, el formato del tiempo que devuelven es distinto, y nuestro Wrapper extrae directamente la información de tiempo solicitada de dichas páginas Web y los manda tal cual al módulo del servidor para que cree la respuesta con ellos.

A continuación, para comprobar que los resultados devueltos en la respuesta son correctos, vamos a mostrar los datos que se obtienen al realizar la misma petición a las páginas Web de las que se ha extraído la información.

- Trayecto realizado en autobús:

Figura 30. Trayecto realizado en autobús

En autobús se tarda 20 minutos, tiempo que coincide perfectamente con el tiempo que devuelve el sistema en la etiqueta <Bus>20 min</Bus> de la respuesta.

- Trayecto realizado en metro:

Figura 31. Trayecto realizado en metro

En metro se tarda 19 minutos, tiempo que coincide perfectamente con el tiempo que devuelve el sistema en la etiqueta <Metro>19 min</Metro> de la respuesta.

- Trayecto realizado en cercanías:

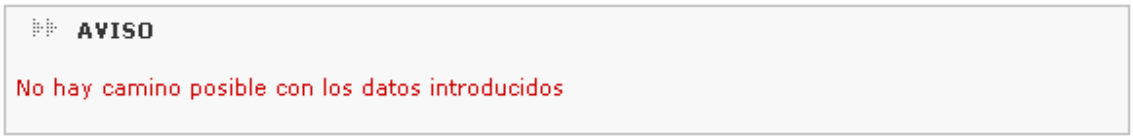


Figura 32. Trayecto realizado en cercanías

En cercanías no se puede realizar el trayecto, resultado que coincide perfectamente con lo que devuelve el sistema, ya que en este caso la respuesta es NO, y esto es lo que aparece en la etiqueta <Cercanias>NO</Cercanías> de la respuesta.

- Trayecto realizado en una combinación de autobús, metro y cercanías:

Figura 33. Trayecto realizado en una combinación de autobús, metro y cercanías

En una combinación de autobús, metro y cercanías se tarda 19 minutos, tiempo que coincide perfectamente con el tiempo que devuelve el sistema en la etiqueta <Bus_Metro_Cercanias>19 min</Bus_Metro_Cercanias> de la respuesta.

- Trayecto realizado en coche:

Figura 34. Trayecto realizado en coche

En coche se tarda 0H09, tiempo que coincide perfectamente con el tiempo que devuelve el sistema en la etiqueta <Coche>0H09</Coche> de la respuesta.

- Trayecto realizado a pie:

Figura 35. Trayecto realizado a pie

A pie se tarda 0H55, tiempo que coincide perfectamente con el tiempo que devuelve el sistema en la etiqueta <Pie>0H55</Pie> de la respuesta.

Hemos comprobado que el sistema funciona perfectamente, ya que crea y devuelve la respuesta con los datos correctos, que coinciden con los que se obtienen si los obtenemos directamente de las páginas Web de las que el sistema extrae la información de tiempo de la respuesta creada.

Lo último que nos queda probar para corroborar el correcto funcionamiento del sistema con la primera petición es la evolución del fichero base de datos en el que se almacenan los datos extraídos para poder obtenerlos de él en posteriores peticiones. En este caso, al ser la primera petición, el fichero base de datos no existe, con lo que el sistema lo tiene que crear e introducir en él las cabeceras y los datos extraídos para la primera petición. En la figura 36 se muestra el contenido que aparece en el fichero *basedatos.txt* que ha creado el sistema:

```
Origen - Destino - Bus - Metro - Cercanias - Bus_Metro_Cercanias - Coche - Pie
Calle de Atocha - Calle de Serrano - 20 min - 19 min - NO - 19 min - 0H09 - 0H55
```

Figura 36. Fichero base de datos en la primera petición

Como vemos, en la primera línea ha introducido las cabeceras:

Origen - Destino - Bus - Metro - Cercanias - Bus_Metro_Cercanias - Coche - Pie

que indican el orden en que se almacenan los datos en el fichero base de datos. Primero se almacena la Calle de Origen, después la Calle de Destino, y a continuación se almacenan los datos de tiempo extraídos de Internet, es decir, el tiempo que se tarda en realizar el trayecto entre la Calle de Origen y la Calle de Destino utilizando autobús, después utilizando metro, después utilizando cercanías, después utilizando una combinación de autobús, metro y cercanías, después utilizando coche y, por último, realizando el camino a pie.

En la línea que va a continuación de las cabeceras aparecen los datos que el sistema ha almacenado para la primera petición. Primero está la Calle de Atocha, correspondiente al origen de la petición, y a continuación está la Calle de Serrano,

correspondiente al destino de la petición. Seguidamente se almacenan los datos de tiempo extraídos por el sistema correspondientes al tiempo que se tarda en realizar el trayecto entre las calles de origen y destino utilizando los distintos medios de transporte. Podemos comprobar que los dichos datos coinciden con los que aparecen en la respuesta que crea el sistema. De esta forma, cuando al sistema vuelva a llegar una petición cuyo origen es la Calle de Atocha y el destino es la Calle de Serrano, la aplicación puede obtener los datos de tiempo solicitados a partir de esta información almacenada en el fichero base de datos, creando la respuesta con ellos sin necesidad de acceder a Internet.

Una vez que hemos comprobado que el sistema funciona correctamente tanto en la creación de la respuesta como en la creación y actualización del fichero base de datos, podemos afirmar que la implementación del sistema es la correcta para el caso de la primera petición.

A continuación vamos a probar el funcionamiento del sistema al realizar otra petición:

```
<?xml version="1.0" encoding="UTF-8"?>
<Peticion>
    <Origen>Calle de Alcala</Origen>
    <Destino>Calle de Costa Rica</Destino>
</Peticion>
```

Figura 37. Petición de entrada

La respuesta creada y devuelta por el sistema es:

```
<?xml version="1.0" encoding="UTF-8"?>
<Respuesta>
    <Origen>Calle de Alcala</Origen>
    <Destino>Calle de Costa Rica</Destino>
    <Bus>42 min</Bus>
    <Metro>27 min</Metro>
    <Cercanias>NO</Cercanias>
    <Bus_Metro_Cercanias>42 min</Bus_Metro_Cercanias>
    <Coche>0H07</Coche>
    <Pie>01H02</Pie>
</Respuesta>
```

Figura 38. Respuesta del sistema

Vemos cómo las etiquetas de la respuesta correspondientes al origen y al destino coinciden con las de la petición. A continuación aparecen los datos de tiempo extraídos de Internet por la aplicación, correspondientes al tiempo que se tarda en el trayecto entre la calle de origen y la calle de destino utilizando los distintos medios de transporte.

Si ahora nos fijamos en la actualización del fichero base de datos:

```
Origen - Destino - Bus - Metro - Cercanias - Bus_Metro_Cercanias - Coche - Pie
Calle de Atocha - Calle de Serrano - 20 min - 19 min - NO - 19 min - 0H09 - 0H55
Calle de Alcala - Calle de Costa Rica - 42 min - 27 min - NO - 42 min - 0H07 - 01H02
```

Figura 39. Fichero base de datos en la segunda petición

Vemos cómo los nuevos datos, que coinciden con los de la respuesta creada por el sistema, han sido añadidos a continuación de los que ya estaban almacenados en el fichero base de datos.

Con esto comprobamos el correcto funcionamiento del sistema para una segunda petición, tanto en la creación de la respuesta como en la actualización del fichero base de datos.

A continuación probamos la situación en la que el origen y el destino están cerca y la página Web de la que extraemos el tiempo que se tarda en autobús, metro y cercanías nos muestra que se puede ir caminando desde el origen al destino y no nos muestra el tiempo total, sino el tiempo que se tarda en esa etapa caminando. En este caso la implementación que hemos realizado devuelve como tiempo de trayecto 9 minutos, que es el tiempo medio que hemos obtenido al realizar muchas pruebas con esta situación. Un ejemplo de esta situación, cuyos resultados mostraremos después al aplicarlo en el sistema implementado, es:

- Trayecto realizado en autobús:

Figura 40. Trayecto realizado en autobús

- Trayecto realizado en metro:

Figura 41. Trayecto realizado en metro

- Trayecto realizado en cercanías:

Figura 42. Trayecto realizado en cercanías

- Trayecto realizado en una combinación de autobús, metro y cercanías:

Figura 43. Trayecto realizado en una combinación de autobús, metro y cercanías

Como podemos ver, en todos los casos la aplicación interpreta que el origen y el destino están cerca, con lo que nos muestra el tiempo que se emplea realizando el trayecto caminando. En este caso no muestra el tiempo donde pone tiempo de trayecto total porque al ser una sola etapa, no hay que sumar tiempos intermedios, con lo que el tiempo de trayecto total es el que muestra esa única etapa caminando. A continuación comprobamos cómo funciona nuestro sistema implementado en este caso.

La petición realizada es:

```
<?xml version="1.0" encoding="UTF-8"?>
<Peticion>
    <Origen>Calle del Topete</Origen>
    <Destino>Avenida de Pablo Iglesias</Destino>
</Peticion>
```

Figura 44. Petición de entrada

Obteniendo como respuesta:

```
<?xml version="1.0" encoding="UTF-8"?>
<Respuesta>
    <Origen>Calle del Topete</Origen>
    <Destino>Avenida de Pablo Iglesias</Destino>
    <Bus>9 min</Bus>
    <Metro>9 min</Metro>
    <Cercanias>9 min</Cercanias>
    <Bus_Metro_Cercanias>9 min</Bus_Metro_Cercanias>
    <Coche>0H02</Coche>
    <Pie>0H11</Pie>
</Respuesta>
```

Figura 45. Respuesta del sistema

Actualizando el fichero base de datos también:

```
Origen - Destino - Bus - Metro - Cercanias - Bus_Metro_Cercanias - Coche - Pie
Calle de Atocha - Calle de Serrano - 20 min - 19 min - NO - 19 min - 0H09 - 0H55
Calle de Alcala - Calle de Costa Rica - 42 min - 27 min - NO - 42 min - 0H07 - 01H02
Calle del Topete - Avenida de Pablo Iglesias - 9 min - 9 min - 9 min - 9 min - 0H02 - 0H11
```

Figura 46. Fichero base de datos

Podemos comprobar que el sistema implementado funciona correctamente, creando la respuesta acorde a esta situación y actualizando el fichero base de datos perfectamente. Vemos cómo nuestro sistema devuelve en todos los casos 9 minutos, en vez de los 12 minutos que vimos que realmente estima la página Web. Esto ya lo hemos explicado anteriormente, y lo hacemos así porque tras realizar muchas pruebas con casos como este, la página Web devuelve tiempos desde 2 minutos a 18 minutos, con lo que nuestro sistema lo que devuelve es un tiempo medio de 9 minutos.

En el caso anterior de los 9 minutos, se podría implementar como respuesta un NO en lugar de los 9 minutos, ya que en realidad lo que te dice la página Web de la que se extrae la información es que entre las calles de origen y destino cercanas se puede ir caminando empleando dicho tiempo, sin utilizar autobús, metro, cercanías o una combinación de los tres. En nuestro caso, hemos decidido devolver datos de tiempo, ya que la página Web de la que se extrae la información devuelve datos de tiempo.

Tras realizar varias peticiones, el fichero base de datos presenta el siguiente aspecto:

```
Origen - Destino - Bus - Metro - Cercanias - Bus_Metro_Cercanias - Coche - Pie
Calle de Atocha - Calle de Serrano - 20 min - 19 min - NO - 19 min - 0H09 - 0H55
Calle de Alcala - Calle de Costa Rica - 42 min - 27 min - NO - 42 min - 0H07 - 01H02
Calle del Topete - Avenida de Pablo Iglesias - 9 min - 9 min - 9 min - 9 min - 0H02 - 0H11
Calle de Ariadna - Avenida de los Poblados - 101 min - 70 min - NO - 85 min - 0H20 - 05H09
Avenida de los Poblados - Calle de Ariadna - 94 min - 70 min - NO - 83 min - 0H17 - 05H09
Calle de Alberto Aguilera - Calle de Huesca - 34 min - 24 min - NO - 24 min - 0H07 - 01H06
Calle del Sauco - Calle de Rafael Salgado - 23 min - 24 min - NO - 24 min - 0H13 - 0H31
Calle del Padre Damian - Avenida de Pablo Iglesias - 23 min - 27 min - NO - 23 min - 0H06 - 0H56
Calle de Alberto Aguilera - Calle de Ariadna - 81 min - 59 min - NO - 53 min - 0H23 - 03H20
Calle de Huesca - Avenida de los Poblados - 76 min - 52 min - NO - 52 min - 0H22 - 03H16
Calle del Topete - Calle de Ariadna - NO - 47 min - NO - 50 min - 0H21 - 02H37
Calle del Desengaño - Calle de Vitruvio - 28 min - 21 min - NO - 28 min - 0H07 - 0H58
Calle de Recoletos - Calle de Rios Rosas - 25 min - 23 min - NO - 25 min - 0H05 - 0H47
```

Figura 47. Fichero base de datos

A continuación realizamos una petición cuyos datos ya estén almacenados en el fichero base de datos. En este caso, la aplicación obtiene los datos de tiempo solicitados de aquí sin necesidad de acceder a Internet para extraerlos.

La petición realizada es:

```
<?xml version="1.0" encoding="UTF-8"?>
<Peticion>
    <Origen>Calle de Alberto Aguilera</Origen>
    <Destino>Calle de Ariadna</Destino>
</Peticion>
```

Figura 48. Petición de entrada

Obteniendo como respuesta:

```
<?xml version="1.0" encoding="UTF-8"?>
<Respuesta>
    <Origen>Calle de Alberto Aguilera</Origen>
    <Destino>Calle de Ariadna</Destino>
    <Bus>81 min</Bus>
    <Metro>59 min</Metro>
    <Cercanias>NO</Cercanias>
    <Bus_Metro_Cercanias>53 min</Bus_Metro_Cercanias>
    <Coche>0H23</Coche>
    <Pie>03H20</Pie>
</Respuesta>
```

Figura 49. Respuesta del sistema

Comprobamos cómo los datos de la respuesta creada por el sistema coinciden con los que estaban almacenados en el fichero base de datos para las calles de la petición:

```
Origen - Destino - Bus - Metro - Cercanias - Bus_Metro_Cercanias - Coche - Pie
Calle de Atocha - Calle de Serrano - 20 min - 19 min - NO - 19 min - 0H09 - 0H55
Calle de Alcala - Calle de Costa Rica - 42 min - 27 min - NO - 42 min - 0H07 - 01H02
Calle del Topete - Avenida de Pablo Iglesias - 9 min - 9 min - 9 min - 9 min - 0H02 - 0H11
Calle de Ariadna - Avenida de los Poblados - 101 min - 70 min - NO - 85 min - 0H20 - 05H09
Avenida de los Poblados - Calle de Ariadna - 94 min - 70 min - NO - 83 min - 0H17 - 05H09
Calle de Alberto Aguilera - Calle de Huesca - 34 min - 24 min - NO - 24 min - 0H07 - 01H06
Calle del Sauco - Calle de Rafael Salgado - 23 min - 24 min - NO - 24 min - 0H13 - 0H31
Calle del Padre Damian - Avenida de Pablo Iglesias - 23 min - 27 min - NO - 23 min - 0H06 - 0H56
Calle de Alberto Aguilera - Calle de Ariadna - 81 min - 59 min - NO - 53 min - 0H23 - 03H20
Calle de Huesca - Avenida de los Poblados - 76 min - 52 min - NO - 52 min - 0H22 - 03H16
Calle del Topete - Calle de Ariadna - NO - 47 min - NO - 50 min - 0H21 - 02H37
Calle del Desengaño - Calle de Vitruvio - 28 min - 21 min - NO - 28 min - 0H07 - 0H58
Calle de Recoletos - Calle de Rios Rosas - 25 min - 23 min - NO - 25 min - 0H05 - 0H47
```

Figura 50. Fichero base de datos

Con lo que podemos afirmar que el sistema implementado funciona correctamente en el caso de que los datos de tiempo solicitados estén almacenados en el fichero base de datos.

A continuación vamos a medir el tiempo de respuesta del sistema ante peticiones en los dos casos estudiados, es decir, cuando los datos de tiempo solicitados se

encuentran almacenados en el fichero base de datos y cuando no. De esta forma pretendemos corroborar la ventaja que supone en cuanto a menor tiempo de respuesta el hecho de que los datos solicitados se encuentren almacenados.

El tiempo de respuesta cuando los datos no se encuentran almacenados viene determinado en su gran mayoría por el comando *lynx*, ya que con él hay que navegar hasta llegar a la página Web de la que se extrae el tiempo solicitado. Con el comando *wget* no se tarda tanto tiempo debido a que éste permite acceder directamente a la página Web final con el tiempo solicitado. Con el comando *lynx* se recorre una lista que almacena las calles de Madrid ordenadas por orden alfabético hasta encontrar las correspondientes al origen y destino de la petición. Según esto, el mínimo tiempo de respuesta es cuando las calles de origen y destino son la primera y segunda de la lista, mientras que el máximo tiempo de respuesta lo determina cuando el origen y destino son las calles última y penúltima de la lista. Por lo tanto, los tiempos de respuesta mínimo y máximo del sistema cuando los datos no se encuentran almacenados en el fichero base de datos son:

- Tiempo mínimo de respuesta:

```
<?xml version="1.0" encoding="UTF-8"?>
<Peticion>
    <Origen>Calle de Abad Juan Catalan</Origen>
    <Destino>Calle de Abada</Destino>
</Peticion>
```

Figura 51. Petición de entrada para el tiempo mínimo de respuesta

El tiempo medio realizando varias pruebas es de aproximadamente 21 segundos.

- Tiempo máximo de respuesta:

```
<?xml version="1.0" encoding="UTF-8"?>
<Peticion>
    <Origen>Calle de Zurita</Origen>
    <Destino>Paseo del Zurron</Destino>
</Peticion>
```

Figura 52. Petición de entrada para el tiempo máximo de respuesta

El tiempo medio realizando varias pruebas es de aproximadamente 2 minutos y 28 segundos.

De acuerdo a los resultados obtenidos, cuando los datos de tiempo solicitados no se encuentran almacenados en el fichero base de datos, el tiempo de respuesta del sistema se encuentra en un rango de [21 segundos - 2 minutos y 28 segundos].

Ahora pasamos a comprobar el caso en el que los datos de tiempo solicitados se encuentran almacenados en el fichero base de datos. Hemos realizado una serie de 10 peticiones correspondientes a este caso, y en todas ellas hemos comprobado que el tiempo que tarda el sistema en dar respuesta a la petición es casi instantáneo, tanto para casos en los que el fichero base de datos tiene mucha información almacenada como

cuando tiene poca. De esta forma verificamos la enorme ventaja en cuanto al menor tiempo de respuesta que supone para la aplicación el hecho de tener almacenada en el fichero base de datos la información de tiempo solicitada. Este tiempo es mucho menor porque la aplicación no tiene que navegar por la Web hasta encontrar los datos solicitados, sino que simplemente tiene que buscarlos en el fichero base de datos, y eso se hace comparando cadenas de caracteres, en lo que se emplea un tiempo casi instantáneo.

Hemos comprobado que el sistema funciona correctamente para todo tipo de peticiones correctas. Ahora vamos a comprobar el comportamiento de la aplicación implementada para peticiones erróneas.

El primer caso erróneo que vamos a probar es cuando no existe la calle de origen introducida en la petición. La petición realizada es:

```
<?xml version="1.0" encoding="UTF-8"?>
<Peticion>
    <Origen>Calle de Adsejs</Origen>
    <Destino>Calle de Ariadna</Destino>
</Peticion>
```

Figura 53. Petición de entrada

Obteniendo como respuesta:

```
<?xml version="1.0" encoding="UTF-8"?>
<Respuesta>
    <Origen>Calle de Adsejs</Origen>
    <Destino>Calle de Ariadna</Destino>
    <Error>No existe la calle Origen="Calle de Adsejs"</Error>
</Respuesta>
```

Figura 54. Respuesta del sistema

En este caso, el sistema finaliza porque ha detectado que la calle de origen no existe, pero antes de salir crea el fichero de respuesta que vemos. En él podemos comprobar que aparecen los datos de la petición de entrada correspondientes a las calles de origen y destino, y a continuación aparece una etiqueta de <Error> que contiene un mensaje de error indicando que la calle de origen no existe. De esta forma, al llegar la respuesta al usuario se da cuenta que ha introducido una calle errónea y puede corregirla para realizar la petición de forma correcta.

El segundo caso erróneo es cuando no existe la calle de destino introducida en la petición. La petición realizada es:

```
<?xml version="1.0" encoding="UTF-8"?>
<Peticion>
    <Origen>Calle de Alberto Aguilera</Origen>
    <Destino>Calle de cusodke</Destino>
</Peticion>
```

Figura 55. Petición de entrada

Obteniendo como respuesta:

```
<?xml version="1.0" encoding="UTF-8"?>
<Respuesta>
    <Origen>Calle de Alberto Aguilera</Origen>
    <Destino>Calle de cusodke</Destino>
    <Error>No existe la calle Destino="Calle de cusodke"</Error>
</Respuesta>
```

Figura 56. Respuesta del sistema

También en este caso, el sistema finaliza porque ha detectado que la calle de destino no existe, pero antes de salir crea el fichero de respuesta que vemos. En él podemos comprobar que aparecen los datos de la petición de entrada correspondientes a las calles de origen y destino, y a continuación aparece una etiqueta de <Error> que contiene un mensaje de error indicando que la calle de destino no existe.

En el caso de que no existan ni la calle de origen ni la calle de destino, el sistema se comporta igual que si es únicamente la calle de origen la que no existe, ya que la implementación realizada primero comprueba si la calle de origen existe o no. Si existe, lo siguiente que hace es comprobar que la calle de destino exista, y si la calle de origen no existe, el sistema finaliza, con lo que ya no analiza si la calle de destino existe o no.

El tercer caso erróneo que queremos comprobar es cuando la calle de origen y la calle de destino son iguales. Esto es un caso erróneo porque no tiene sentido solicitar el tiempo de desplazamiento entre dos puntos que son el mismo. La petición realizada es:

```
<?xml version="1.0" encoding="UTF-8"?>
<Peticion>
    <Origen>Calle de Alberto Aguilera</Origen>
    <Destino>Calle de Alberto Aguilera</Destino>
</Peticion>
```

Figura 57. Petición de entrada

Obteniendo como respuesta:

```
<?xml version="1.0" encoding="UTF-8"?>
<Respuesta>
    <Origen>Calle de Alberto Aguilera</Origen>
    <Destino>Calle de Alberto Aguilera</Destino>
    <Error>Origen="Calle de Alberto Aguilera" y Destino="Calle de Alberto Aguilera" son iguales</Error>
</Respuesta>
```

Figura 58. Respuesta del sistema

El sistema finaliza porque ha detectado que la calle de origen y la calle de destino son iguales, pero antes de salir crea el fichero de respuesta que vemos. En él podemos comprobar que aparecen los datos de la petición de entrada correspondientes a las calles de origen y destino, y a continuación aparece una etiqueta de <Error> que contiene un mensaje de error indicando que la calle de origen y la calle de destino son iguales. Como explicamos antes, al llegar la respuesta al usuario se da cuenta que ha introducido la misma calle en el origen y en el destino, con lo que puede corregirla para realizar la petición de forma correcta.

Hemos comprobado que el sistema funciona correctamente para cada uno de los tres casos erróneos que hemos implementado.

Con todo esto, podemos afirmar que la implementación realizada funciona correctamente, tanto para los casos de peticiones correctas como para los casos de peticiones erróneas.

Un último dato que queremos aportar es el tamaño máximo aproximado que puede alcanzar el fichero base de datos en el caso de hacer todas las peticiones posibles utilizando todas las calles que tenemos almacenadas. En el fichero *calles* tenemos 8962 calles, con lo que cada calle se puede emparejar con cada una de las 8961 restantes para formar una petición, pudiendo ser ambas origen y destino. Hay una fórmula matemática que se adapta a lo que buscamos, y que recibe el nombre de *variaciones sin repetición*. Las *variaciones sin repetición de n elementos tomados de p en p* se definen como las distintas agrupaciones formadas con *p* elementos distintos, eligiéndolos de entre los *n* elementos de que disponemos, considerando una variación distinta a otra tanto si difieren en algún elemento como si están situados en distinto orden. El número de variaciones que se pueden construir se puede calcular mediante la fórmula:

$$V_n^p = \frac{n!}{(n-p)!}$$

En nuestro caso, n = 8962 y p = 2, ya que lo que queremos son variaciones sin repetición de 8962 calles tomadas de 2 en 2. Aplicando esto en la fórmula, obtenemos:

$$V_n^p = \frac{n!}{(n-p)!}\Bigg|_{\substack{n=8962\\p=2}} = \frac{8962!}{(8962-2)!} = \frac{8962!}{8960!} = 8962 \cdot 8961 = 80308482$$

Tenemos un total de 80308482 variaciones posibles con las 8962 calles. Por lo tanto, tendremos un máximo de 80308482 entradas posibles a almacenar en el fichero base de datos. Cada entrada ocupa una media de 90 caracteres, o lo que es lo mismo, 90 Bytes. Por lo tanto, tendremos un tamaño máximo aproximado del fichero base de datos de $80308482 \cdot 90 = 7227763380$ Bytes $\approx 7{,}2$ GB.

Este tamaño de fichero se puede considerar elevado en cuanto a información almacenada. Sin embargo, como ya se mencionó anteriormente, hemos comprobado tras varias pruebas que el tiempo de acceso al fichero a la hora de buscar la información de tiempo solicitada no varía apenas tanto para el fichero con mucha información almacenada como con poca, ya que solo hay caracteres en él y lo único que hay que hacer es comparar cadenas de caracteres en cada petición, y eso no lleva apenas tiempo. La respuesta del sistema, tanto para mucha información almacenada en el fichero base de datos como para poca, es casi instantánea.

5. CONCLUSIONES

En este apartado se muestran las conclusiones que he extraído tras la realización del Proyecto Fin de Carrera.

Respecto a la implementación realizada, se puede concluir que su funcionamiento es el correcto, cumpliendo los objetivos marcados de programación y resultados. He sido capaz de programar los distintos módulos que componen la aplicación obteniendo con ellos los resultados esperados, referentes al cálculo de modos y tiempos de desplazamiento entre dos puntos de una ciudad usando fuentes públicas. La programación de dos módulos independientes ha sido una buena idea, ya que cada vez que no funcionaba algo de uno de ellos y lo tenía que retocar, se podía hacer sin tener que modificar nada del otro. Cada uno de los módulos realiza su función perfectamente. El módulo del servidor procesa la petición y crea la respuesta para el usuario y el módulo de extracción de la información extrae de Internet la información de tiempo solicitada y la devuelve al módulo del servidor. Ante una petición de entrada, el sistema devuelve al usuario una respuesta, tanto para peticiones correctas como para peticiones erróneas, devolviendo en el primer caso los datos de tiempo solicitados, y en el segundo, un mensaje indicando el error cometido en la petición para que el usuario lo pueda subsanar y realizar la petición correctamente.

El programa implementado ha sido el más grande que he realizado en la carrera, lo cual ha servido para probar mi nivel de programación. Me ha ayudado para poner en práctica mis conocimientos del lenguaje C, y sobre todo, a resolver los problemas que la programación conlleva por mí solo, buscando y pensando hasta encontrar la solución adecuada. El mayor problema que he encontrado durante la programación del sistema ha sido el acceder a Internet y obtener el código fuente de páginas Web desde mi programa, ya que no he encontrado ninguna función del lenguaje C que realice dicha tarea. Para solucionar este problema he tenido que recurrir a una solución que no está relacionada con la programación, ya que se han utilizado comandos de Linux. Es una muestra importante de lo que me ha ayudado este proyecto a buscarme la vida para solucionar los problemas que aparecen.

Pero no sólo me ha servido este proyecto para familiarizarme más con el lenguaje de programación C, sino también para ponerme al día y aprender todo lo relacionado con la Web Semántica, la Minería Web y los Wrappers. Había oído de la existencia de estos temas, pero no los había tratado nunca. Ha sido en este proyecto donde, tras un estudio teórico previo y la posterior aplicación en la implementación realizada, he aprendido el conocimiento de estas materias. Una muestra clara de estos conocimientos adquiridos es que he sabido aplicarlos en la implementación del proyecto realizado. Además, me resulta satisfactorio haber adquirido estos conocimientos no sólo porque no los había tratado nunca, sino porque forman el presente y el futuro de la Web y las aplicaciones relacionadas con ésta, y seguramente me resultarán de mucha utilidad en el futuro.

Como conclusión general se puede decir que con el proyecto realizado se han cumplido los objetivos marcados al principio de programación, de resultados y personales.

6. AMPLIACIÓN DEL PROYECTO

En este apartado se exponen las posibles aplicaciones o líneas de trabajo que se pueden realizar en un futuro como ampliación o mejora de lo que se ha realizado en este Proyecto Fin de Carrera:

- La implementación realizada calcula los modos y tiempos de desplazamiento entre dos calles, plazas o parques de Madrid determinadas por una lista de las mismas que contiene una de las páginas Web de las que se extrae la información. Una posible mejora sería incluir entre los posibles puntos de origen y destino lugares emblemáticos de Madrid, como pueden ser edificios importantes de cultura, de deportes, de ocio, civiles, religiosos, monumentos, jardines… En tal caso habría que buscar otras páginas Web de las que extraer dicha información, ya que con las utilizadas en este proyecto no se podría realizar esta ampliación al no disponer de toda esa información.

- Se podría ampliar la información a extraer entre los puntos de origen y destino en cuestión, como puede ser la distancia entre ellos, o el dinero que se gasta en realizar el trayecto utilizando los distintos medios de transporte, o el horario de los medios de transporte.

- Se puede trasladar el cálculo del modo y tiempo de desplazamiento entre dos puntos de otras ciudades de España, no sólo de Madrid. Para ello haría falta encontrar páginas Web adecuadas.

- También a partir de la idea implementada en este proyecto se podría realizar una aplicación que calcule el tiempo y modo de desplazamiento entre dos ciudades, no entre calles.

- Hemos almacenado la información extraída en un fichero debido a que la cantidad de datos a almacenar no es muy grande. Se podría mejorar el lugar de almacenamiento creando una base de datos, con MySQL o algún software similar. De esta forma se podría almacenar y gestionar mejor grandes cantidades de información, que estaría mejor organizada y relacionada.

- Hemos construido un Wrapper específico para las páginas Web de las que se ha extraídos la información, basado en ciertas partes o palabras específicas de sus códigos fuente, con lo que cualquier variación de estas páginas Web podría afectar al Wrapper creado. Se podría mejorar el Wrapper para que se adaptase a algún cambio que se produjese en las páginas Web fuente, siempre que este cambio no fuese muy grande, ya que en ese caso habría que construir otro Wrapper.

- Se podría mejorar la respuesta que se da en los casos de peticiones erróneas. En nuestro caso se responde indicando el error cometido en la petición. En una ampliación, se podría indicar alguna sugerencia, como la existencia de alguna calle cuyo nombre se parece al que se ha introducido erróneo.

- Hay casos en los que las calles de origen y destino están cerca y el tiempo que devuelve la aplicación utilizando autobús, metro, cercanías y una combinación de los tres es de 9 min., debido a que, como ya se explicó, hemos considerado que es la media del tiempo que devuelve la página Web en todos estos casos. Una posible mejora de la aplicación es devolver el tiempo exacto que da la página Web de la que se extrae la información.

- En una futura aplicación se podría mejorar el tiempo de respuesta del sistema cuando los datos de tiempo solicitados no se encuentran almacenados en el fichero base de datos, ya que se puede considerar que en este caso son un poco elevados al encontrarse en un rango de [21 segundos - 2 minutos y 28 segundos].

7. MANUAL DE USUARIO

En este apartado se explican los pasos que tiene que seguir el usuario para utilizar la aplicación implementada, además de los requisitos de instrumental y software que se necesitan para poder ejecutarla.

El equipamiento técnico y de software que se necesita para esta aplicación es:

- Un ordenador personal, fijo o portátil, ya que, aunque esta implementación forma parte de un sistema pensado para una PDA o dispositivo móvil, para ejecutarla por sí sola se necesita un ordenador.

- Conexión a Internet, ya que es de Internet de donde se extrae la información de tiempo solicitada. No importa la velocidad contratada de la conexión, debido a que en esta aplicación no influye el tiempo que se tarda en navegar o extraer la información solicitada, sino que se basa en que los datos extraídos sean los correctos.

- Sistema Operativo: Linux. Para la ejecución del programa implementado se necesitan comandos de GNU/Linux, como *wget* y *lynx*. Además, el programa se ha implementado con el lenguaje C sobre Linux, con lo que para compilarlo se necesita *gcc*, que son un conjunto de compiladores estándar para Linux con el que se compila el lenguaje C.

Los pasos que tiene que seguir el usuario para ejecutar la aplicación implementada son:

- Lo primero que hay que hacer es abrir una consola de la línea de comandos y acceder al directorio donde se encuentran los ficheros que componen el proyecto. En nuestro caso se encuentran en el directorio *pfc*, con lo que para acceder a ese directorio, lo hacemos con el comando *cd pfc*.

- Lo siguiente es abrir y editar el fichero *petición.txt*, introduciendo entre las etiquetas de <origen></origen> y <destino></destino> los puntos de origen y destino, respectivamente, entre los que queremos calcular los modos y tiempos de desplazamiento.

- A continuación ejecutamos el programa que implementa la aplicación, que se llama *pfc*. La forma de ejecutarlo es utilizando el comando *./pfc*.

- Como consecuencia de la ejecución del programa, la aplicación crea un fichero de respuesta llamado *respuesta.txt*, donde se pueden ver los datos que devuelve el sistema, ya sean los datos de tiempo extraídos para peticiones correctas como el mensaje de error para peticiones erróneas.

8. FICHEROS DEL PROYECTO

En este apartado se enumeran y explican todos los ficheros que componen el proyecto, que se dividen en fichero de entrada, fichero de salida y ficheros que implementan la aplicación:

- Fichero de entrada: *petición.txt*.

Es un fichero de texto con formato XML en el que hay que especificar el lugar de origen y el lugar de destino entre los que se quiere hallar el tiempo de desplazamiento.

- Fichero de salida: *respuesta.txt*.

Es un fichero de texto con formato XML mediante el que el sistema devuelve los datos de tiempo extraídos correspondientes al tiempo de desplazamiento entre origen y destino utilizando los distintos medios de transporte.

- Ficheros que componen la aplicación:

  - Fichero *pfc.c*.

    Es el fichero que contiene el código fuente en lenguaje C que implementa el programa de la aplicación.

  - Fichero *pfc.o*.

    Es el fichero objeto que resulta de compilar el fichero *pfc.c*. Este fichero se obtiene con el comando *gcc -c pfc.c*.

  - Fichero *pfc*.

    Es el fichero que contiene el programa ejecutable que resulta de enlazar el fichero objeto *pfc.o* junto a las librerías de sistema. Este fichero se obtiene con el comando *gcc –o pfc pfc.o*.

  - Fichero *itinerario.h*.

    Es un fichero de definiciones que hemos creado donde se encuentran las definiciones de las funciones y estructuras que ese utilizan en el programa principal *pfc.c*.

  - Fichero *calles*.

    Es un fichero de texto que contiene todas las calles de Madrid que pueden ser origen o destino de la aplicación. Se ha creado a partir de la lista de calles de la que se sirve una de las páginas Web utilizadas para la extracción de la información.

9. GLOSARIO

En este apartado se hace una breve descripción de algunos de los términos que aparecen en el texto de la memoria del Proyecto Fin de Carrera realizado.

- Base de datos.- Es un conjunto de datos pertenecientes a un mismo contexto y almacenados sistemáticamente para su posterior uso.

- Código fuente.- El código fuente de un programa informático es un conjunto de líneas de texto que son las instrucciones que debe seguir el ordenador para ejecutar dicho programa. El término código fuente también se usa para hacer referencia al código fuente de una página Web que está escrito en el lenguaje de marcado HTML (aunque puede estar escrita usando también otros lenguajes como JavaScript y otros) y que es posteriormente ejecutado por el navegador Web para visualizar dicha página cuando es visitada.

- Compilador.- Es un programa informático que traduce un programa escrito en un lenguaje de programación a otro lenguaje de programación, generando un programa equivalente que la máquina será capaz de interpretar. Usualmente el segundo lenguaje es código máquina, pero también puede ser simplemente texto. Este proceso de traducción se conoce como compilación.

- Etiqueta (o Tag).- Es una marca con tipo que delimita una región en los lenguajes de marcado. Los paréntesis angulares < > enmarcan normalmente la etiqueta de apertura o inicio, y </ > acotan la etiqueta de cierre o final.

- gcc.- Siglas de GNU Compiler Collection (Colección de Compiladores GNU). Es un conjunto de compiladores creados por el proyecto GNU. gcc es software libre y son compiladores que se consideran estándar para los sistemas operativos derivados de UNIX, como Linux. Originalmente GCC significaba *GNU C Compiler* (*compilador GNU para C*), porque sólo compilaba el lenguaje C. Posteriormente se extendió para compilar C++, Fortran, Ada y otros.

- Hipertexto.- Es el nombre que recibe el texto que en la pantalla de un ordenador conduce a su usuario a otro texto relacionado.

- HTML.- Siglas de HyperText Markup Language (Lenguaje de Marcas de Hipertexto). Es el lenguaje de marcado predominante para la construcción de páginas Web. Es usado para describir la estructura y el contenido en forma de texto, así como para complementar el texto con objetos tales como imágenes.

- HTTP.- Siglas de HyperText Transfer Protocol (Protocolo de Transferencia de Hipertexto). Es el protocolo usado en cada transacción de la Web (WWW).

- Internet.- Es un conjunto descentralizado de redes de comunicación interconectadas, que utilizan la familia de protocolos TCP/IP, garantizando que las redes físicas heterogéneas que la componen funcionen como una red lógica única, de alcance mundial.

- JavaScript.- Es un lenguaje de programación interpretado, es decir, que no requiere compilación, utilizado principalmente en páginas Web, con una sintaxis semejante a la del lenguaje Java y el lenguaje C.

- Lenguaje C.- Es un lenguaje de programación creado en 1972 por Ken Thompson y Dennis M. Ritchie en los Laboratorios Bell. Es un lenguaje orientado a la implementación de Sistemas Operativos, concretamente Unix. C se caracteriza por la eficiencia del código que produce y es el lenguaje de programación más popular para crear software de sistemas, aunque también se utiliza para crear aplicaciones.

- Lenguaje de programación.- Es un conjunto de símbolos y reglas sintácticas y semánticas que definen su estructura y el significado de sus elementos y expresiones. Es utilizado para controlar el comportamiento físico y lógico de una máquina.

- Lenguaje Java.- Es un lenguaje de programación orientado a objetos desarrollado por Sun Microsystems a principios de los años 90.

- Línea de comandos.- Es la interfaz más simple de un sistema operativo, en la que todo lo que se desea hacer tiene que ser expresado en forma de instrucciones o comandos.

- Linux.- Se suele utilizar el término *Linux* para referirse al sistema operativo Unix-like que utiliza como base las herramientas de sistema de GNU y el núcleo Linux.

- Mediador.- Es una especie de base de datos convencional o base de conocimiento representada con datos estructurados extraídos desde fuentes semi o no estructurados.

- Metadatos.- Son datos que describen otros datos. Datos estructurados y codificados que describen características de instancias conteniendo informaciones para ayudar a identificar, descubrir, valorar y administrar las instancias descritas.

- Minería Web (o Web Míning).- Es el uso de técnicas para descubrir y extraer de forma automática información de los documentos y servicios de la Web.

- Motor de búsqueda.- Es un sistema informático que indexa archivos almacenados en servidores Web. Las búsquedas se hacen con palabras clave o con árboles jerárquicos por temas. El resultado de la búsqueda es un listado de direcciones Web en los que se mencionan temas relacionados con las palabras clave buscadas.

- Navegador Web.- Es un programa que permite visualizar la información que contiene una página Web. Puede tener una interfaz gráfica de usuario como Internet Explorer, Opera (navegador), Netscape Navigator, Mozilla Firefox, etc. o puede tener una interfaz en modo texto como Lynx.

- Ontología.- Es un exhaustivo y riguroso esquema conceptual dentro de uno o varios dominios dados, con la finalidad de facilitar la comunicación y la compartición de la información entre diferentes sistemas y entidades.

- OWL.- Siglas de Ontology Web Language (Lenguaje de Ontologías Web). Es un lenguaje de marcado para publicar y compartir datos usando ontologías en la WWW. Tiene como objetivo facilitar un modelo de marcado construido sobre RDF y codificado en XML.

- Página Web.- Es una fuente de información adaptada para la World Wide Web (WWW) y accesible mediante un navegador Web que normalmente forma parte de un sitio Web.

- Programa.- Es un conjunto de instrucciones para un ordenador. Un programa se puede referir tanto a un programa ejecutable como a su código fuente, el cual es transformado en un ejecutable cuando es compilado.

- RDF.- Siglas de Resource Description Framework (Marco de Descripción de Recursos). Es un lenguaje para la definición de ontologías y metadatos en la Web.

- SGML.- Siglas de Standard Generalized Markup Language (Lenguaje de Marcado Generalizado). Consiste en un sistema para la organización y etiquetado de documentos.

- Sistema Operativo.- Es un software para controlar e interactuar con el sistema, es decir, un conjunto de programas de computación destinados a realizar muchas tareas entre las que destaca la administración eficaz de sus recursos.

- Sitio Web.- Es un conjunto de páginas Web, típicamente comunes a un dominio de Internet o subdominio en la World Wide Web en Internet.

- URI.- Siglas de Uniform Resource Identifier (identificador uniforme de recurso). Es una cadena corta de caracteres que identifica inequívocamente un recurso (servicio, página, documento, dirección de correo electrónico, enciclopedia, etc.). Normalmente estos recursos son accesibles en una red o sistema.

- URL.- Siglas de Uniform Resource Locator (Localizador Uniforme de Recurso). Es una secuencia de caracteres, de acuerdo a un formato

estándar, que se usa para nombrar recursos, como documentos e imágenes en Internet, por su localización.

- Web Semántica.- Consiste en añadir metadatos semánticos y ontológicos a la World Wide Web. Esas informaciones adicionales que describen el contenido, el significado y la relación de los datos se deben proporcionar de manera formal, para que así sea posible evaluarlas automáticamente por máquinas de procesamiento.

- Wrapper.- Es un software que acepta consultas de usuarios de datos en la Web y después de extraer la información relevante retorna los resultados.

- WWW.- Siglas de World Wide Web (Red Global Mundial). Es un sistema de documentos de hipertexto y/o hipermedios enlazados y accesibles a través de Internet. Con un navegador Web, un usuario visualiza páginas Web que pueden contener texto, imágenes, vídeos u otros contenidos multimedia, y navega a través de ellas usando hiperenlaces.

- XML.- Siglas de Extensible Markup Language (lenguaje de marcas ampliable). Es un metalenguaje (lenguaje usado para hacer referencia a otros lenguajes) extensible de etiquetas desarrollado por el World Wide Web Consortium (W3C). XML no es realmente un lenguaje en particular, sino una manera de definir lenguajes. Es una simplificación y adaptación del SGML.

10. BIBLIOGRAFÍA

[Abrams 1998] M. Abrams, ed., World Wide Web - Beyond the Basics, Prentice Hall, 1998. Versión on-line disponible en http://ei.cs.vt.edu/~wwwbtb/book/ (último acceso en Octubre de 2008).

[Araque 2000] ARAQUE CUENCA, HURTADO TORRES, SAMOS JIMENEZ. Extracción de Información de Fuentes de Datos Heterogéneas e Incorporación al Data Warehouse. Universidad de Granada. 2000.

[Berners-Lee 1989] T. Berners-Lee. Information Management: A Proposal. Internal Project Proposal, CERN, 1989. Disponible en http://www.w3.org/History/1989/proposal.html (último acceso en Octubre de 2008).

[Berners-Lee 2001] T. Berners-Lee, J. Hendler, O Lassila. The Semantic Web. Scientific American, May 2001.

[Connolly 2000] D. Connolly. A Little History of the World Wide Web. W3C Consortium, 2000. Disponible en http://www.w3.org/History.html (último acceso en Octubre de 2008).

[Etzioni 1996] Etzioni O. "The World Wide Web: quagmire or gold mine?", In Communications of the ACM 39(11). 1996

[Gruber 1993] T. R. Gruber. A Translation Approach to Portable Ontology Specifications. Knowledge Acquisition, 5(2), pp. 199-220, 1993.

[Inmon 1993] Inmon, W.: "The Operational Data Store. Prism Tech Topics. Vol.1, No.17. 1993.

[Laender 2002] Laender, A., Ribeiro-Neto, B., Silva, A. and Teixeira, J. A Brief Survey of Web Data Extraction Tools, in: SIGMOD Record, Volume 31, Number 2, Junio 2002.

[McGuinness and Wright 1998] McGuinness, D.L. and Wright, J.. Conceptual Modeling for Configuration: A Description Logic-based Approach. *Artificial Intelligence for Engineering Design, Analysis, and Manufacturing - special issue on Configuration*, 1998.

[McGuinness et al. 2000] McGuinness, D.L., Fikes, R., Rice, J. and Wilder, S.. An Environment for Merging and Testing Large Ontologies. *Principles of Knowledge Representation and Reasoning: Proceedings of the Seventh International Conference (KR2000)*. A. G. Cohn, F. Giunchiglia and B. Selman, editors. San Francisco, CA, Morgan Kaufmann Publishers. 2000.

[Musen 1992] Musen, M.A.. Dimensions of knowledge sharing and reuse. *Computers and Biomedical Research* 25: 435-467, 1992.

[O'Neill 2003] E. T. O'Neill, B. F. Lavoie, R. Bennett. Trends in the Evolution of the Public Web. D-Lib Magazine, Volume 9 Number 4, April 2003. Disponible en http://www.dlib.org/dlib/april03/lavoie/04lavoie.html (último acceso en Octubre de 2008).

[Rothenfluh et al. 1996] Rothenfluh, T.R., Gennari, J.H., Eriksson, H., Puerta, A.R., Tu, S.W. and Musen, M.A.. Reusable ontologies, knowledge-acquisition tools, and performance systems: PROTÉGÉ-II solutions to Sisyphus-2. *International Journal of Human-Computer Studies* 44: 303-332, 1996.

[Scotto 2004] Scotto, M. Sillitti, A. Succi, G. Vernazza, T. "Managing Web-Based Information", *International Conference on Enterprise Information Systems (ICEIS 2004),* Porto, Portugal, April 2004. Page 1-3

[Studer 1998] Studer S, Benjamins R., and Fensel D., "Knowledge Engineering: Principles and Methods", Data and Knowledge Engineering, 25, 161-197, 1998.

[Wikipedia] http://es.wikipedia.org.