E.U.I.T. TELECOMUNICACIÓN

PROYECTO FIN DE CARRERA
PLAN 2000

TEMA: Internet Móvil

TÍTULO: WSM: METODOLOGÍA DE WEB SCRAPING PARA ANDROID Y EJEMPLIFICACIÓN MEDIANTE LA APLICACIÓN UPMDROID.

AUTOR: Víctor Ramón Gracia Nicolás, Alberto Galán Sánchez

TUTOR: Antonio da Silva Fariña

DEPARTAMENTO: DIATEL

Miembros del Tribunal Calificador:

PRESIDENTE: Juan M. Meneses Chaus

VOCAL: Antonio da Silva Fariña **Vº Bº.**

VOCAL SECRETARIO: Carlos Ramos Nespereira

DIRECTOR:

Fecha de lectura:

Calificación: **El Secretario,**

RESUMEN DEL PROYECTO:

El auge de los terminales móviles ha provocado en la web la necesidad de adaptación de sus contenidos a las características de los nuevos terminales. El termino Web scraping o web harvesting es una técnica para la recolección y presentación de datos de sitios web heredados que no disponen de un acceso específico para terminales móviles.

Se pretende analizar y describir la metodología de interacción entre un navegador y un servidor web, extraer la información útil y, posteriormente, definir un esqueleto de aplicación Android genérica que visualice la información extraída.

Como prueba de concepto se desarrollará la aplicación "UPMdroid". Mediante esta aplicación se pretende que un alumno tenga acceso a la información relacionada con sus estudios en la ETSIST y la UPM como su horario de clases y laboratorios y fechas de exámenes de las asignaturas en las que este matriculado.

UNIVERSIDAD POLITÉCNICA DE MADRID

Escuela Técnica Superior de Ingeniería
y Sistemas de Telecomunicación

PROYECTO FIN DE CARRERA

WSM: METODOLOGÍA DE WEB SCRAPING PARA ANDROID Y EJEMPLIFICACIÓN MEDIANTE LA APLICACIÓN UPMDROID.

Autores:
VÍCTOR RAMÓN GRACIA NICOLÁS
ALBERTO GALÁN SÁNCHEZ

Tutor:
ANTONIO DA SILVA FARIÑA

Madrid, Junio de 2014

Agradecimientos

Mis agradecimientos son para todos aquellos que han hecho
posible que esto se convierta en realidad.

Especialmente doy gracias a mi familia y mis padres
que me han estado apoyando, sufriendo conmigo y sin los cuales
no estaría escribiendo esto ahora.
A ti, María, que eres la persona más importante de mi vida.
A mis abuelas, que les habría gustado ver este momento.

A mis amigos, con los que he pasado momentos
inolvidables a lo largo de estos años: cDc, Grupo EUITT, telemáticos, etc.

A mi tutor Antonio, por la enorme paciencia que ha tenido.
Y a todas esas personas que no he nombrado pero también tienen su huequito.
Gracias.

"Más vale saber que haber, dice la común sentencia,
el saber nunca se acaba y el haber no tiene ciencia"
Refrán popular.

Víctor.

A mi familia por haberme apoyado siempre.
A Marta, por ser mi pasado, presente y futuro y estar siempre ahí.
A mis amigos, y a todos mis compañeros de la escuela, por todas esas risas.

"El mundo es como un libro abierto, quien no viaja sólo ha leído la primera página."

Alberto

Especial agradecimiento para Aarón Gómez Cordero que nos permitió con su usuario
y contraseña de la escuela poder hacer las pruebas necesarias para el correcto
funcionamiento de la aplicación.

Resumen

En la realización de este proyecto se ha tratado principalmente la temática del web scraping sobre documentos HTML en Android. Como resultado del mismo, se ha propuesto una metodología para poder realizar web scraping en aplicaciones implementadas para este sistema operativo y se desarrollará una aplicación basada en esta metodología que resulte útil a los alumnos de la escuela.

Web scraping se puede definir como una técnica basada en una serie de algoritmos de búsqueda de contenido con el fin de obtener una determinada información de páginas web, descartando aquella que no sea relevante. Como parte central, se ha dedicado bastante tiempo al estudio de los navegadores y servidores Web, y del lenguaje HTML presente en casi todas las páginas web en la actualidad así como de los mecanismos utilizados para la comunicación entre cliente y servidor ya que son los pilares en los que se basa esta técnica.

Se ha realizado un estudio de las técnicas y herramientas necesarias, aportándose todos los conceptos teóricos necesarios, así como la proposición de una posible metodología para su implementación.

Finalmente se ha codificado la aplicación UPMdroid, desarrollada con el fin de ejemplificar la implementación de la metodología propuesta anteriormente y a la vez desarrollar una aplicación cuya finalidad es brindar al estudiante de la ETSIST un soporte móvil en Android que le facilite el acceso y la visualización de aquellos datos más importantes del curso académico como son: el horario de clases y las calificaciones de las asignaturas en las que se matricule.

Esta aplicación, además de implementar la metodología propuesta, es una herramienta muy interesante para el alumno, ya que le permite utilizar de una forma sencilla e intuitiva gran número de funcionalidades de la escuela solucionando así los problemas de visualización de contenido web en los dispositivos.

Abstract

The main topic of this project is about the web scraping over HTML documents on Android OS. As a result thereof, it is proposed a methodology to perform web scraping in deployed applications for this operating system and based on this methodology that is useful to the ETSIST school students.

Web scraping can be defined as a technique based on a number of content search algorithms in order to obtain certain information from web pages, discarding those that are not relevant. As a main part, has spent considerable time studying browsers and Web servers, and the HTML language that is present today in almost all websites as well as the mechanisms used for communication between client and server because they are the pillars which this technique is based.

We performed a study of the techniques and tools needed, providing all the necessary theoretical concepts, as well as the proposal of a possible methodology for implementation.

Finally it has codified UPMdroid application, developed in order to illustrate the implementation of the previously proposed methodology and also to give the student a mobile ETSIST Android support to facilitate access and display those most important data of the current academic year such as: class schedules and scores for the subjects in which you are enrolled.

This application, in addition to implement the proposed methodology is also a very interesting tool for the student, as it allows a simple and intuitive way of use these school functionalities thus fixing the viewing web content on devices.

Índice

Resumen.....1

Abstract.....2

Índice de figuras.....6

Introducción.....8

BLOQUE 1: ANÁLISIS DE LAS TECNOLOGÍAS USADAS.....11

1.1 Navegador web.....12

1.2 HTML.....14

1.2.1 Historia de HTML.....14

1.2.2 Conceptos básicos del lenguaje.....16

1.2.3 Principales tecnologías alrededor de HTML.....18

1.3 Servidor.....19

1.3.1 Concepto.....19

1.3.2 Tipos de servidores.....19

1.3.3 Servidor web.....21

1.4 Interacción entre navegador y servidor.....23

1.4.1 Protocolo HTTP.....23

1.5 Análisis de la información recibida: Parsing.....32

1.5.1 SAX.....35

1.5.2 DOM.....37

1.6 Extracción de la información necesaria: Scraping.....43

1.6.1. Definición.....43

1.6.2. Estado del arte.....44

1.6.3. Funcionamiento.....45

1.6.4. Herramientas de Scraping.....47

1.6.5. Utilidades y aplicaciones comerciales.....48

1.7 Android.....53

1.7.1 Recorrido por las diferentes versiones 53
1.7.2 ¿Qué se necesita para programar? 61
1.7.3 Elementos básicos 61
1.7.4 Ciclo de vida de una aplicación 63

BLOQUE 2: METODOLOGÍA PARA WEB SCRAPING EN ANDROID 66
2.1 Android y web scraping 67
2.2 Metodología 71
2.2.1 Metodología: definición y conceptos 71
2.2.2 Ventajas del uso de metodologías para desarrollo de software 72
2.2.3 Desventajas del uso de metodologías para desarrollo de software 73
2.2.4 Metodologías tradicionales y ágiles 73
2.3 Propuesta de metodología para Web Scraping: WSM, Web Scraping Methodology ... 77
2.4 Fase 1: Análisis de la comunicación entre cliente y servidor 79
2.5 Fase 2: Comunicación y extracción de la respuesta del servidor. 84
2.5.1 Comunicación para webs estáticas 84
2.5.2 Extracción de la respuesta 86
2.5.3 Comunicación para webs dinámicas 888
2.6 Fase 3: Parseo y extracción de la información relevante al usuario. 101
2.7 Fase 4: Presentación de la información al usuario. 103

BLOQUE 3: Aplicación UPMdroid 104
3.1 Introducción 105
3.2 Análisis de requisitos 111
3.2.1 Requisitos Funcionales 111
3.2.2 Requisitos no Funcionales 112
3.2.3 Conclusiones 113
3.3 Entorno 114
3.4 Diagrama de casos de uso 116
3.4.1 Casos de uso general 116

3.5 Diagramas de actividad 120
3.5.1 Comportamiento para el caso de uso Consultar calificaciones de exámenes 120
3.5.2 Comportamiento para el caso de uso Consultar horario de clases 122
3.6 Diagrama de clases 123
3.7 Diagramas de secuencia 126
3.8 Implementación 131
3.8.1 Prueba de la aplicación sobre la plataforma Intranet: 131
3.8.2 Prueba de funcionamiento sobre la plataforma Politécnica Virtual: 133

4. Conclusiones 135
4.1 Trabajos futuros 136

5. Anexo I. Manual de usuario 137
5.1 Acceder a la aplicación 137
5.2 Realizar una consulta 138
5.3 Preferencias 139
5.4 Acerca de 143

6. Glosario 144
7. Bibliografía 145

Índice de figuras

Figura 1. Familia SGML 15
Figura 2. Servidor web 21
Figura 3. *Esquema de funcionamiento de SAX* 36
Figura 4. *Esquema de la arquitectura del API de DOM.* 38
Figura 5. *Ejemplo de documento XML.* 39
Figura 6. *Estructura DOM del documento XML anterior.* 39
Figura 7. Representación web, HTML, DOM 43
Figura 8. *Ejemplo de Web Scraping.* 46
Figura 9. *Gráfico de motores de búsqueda.* 49
Figura 10. *Funcionamiento de un spider.* 51
Figura 11. Funcionamiento de un bot de adquisición 52
Figura 12. Ciclo de vida de aplicación Andoid 63
Figura 13. Metodologías ágiles y tradicionales 76
Figura 14. *Utilización de Mozilla Firebug* 80
Figura 15. *Captura del tráfico HTTP con HTTP Fox* 81
Figura 16. *Captura del tráfico HTTP con la herramienta de Google Chrome.* 82
Figura 17. *Detalle de una petición HTTP.* 83
Figura 18. *Vista de la Actividad.* 98
Figura 19. *Ejemplo de árbol DOM creado a partir de un documento HTML.* 102
Figura 20. *Extracto del código HTML de la Web de Intranet de la ETSIST.* 106
Figura 21. *Extracto del código HTML de la Web de Intranet de la ETSIST.* 107
Figura 22. *Extracto del código HTML de la Web de Politécnica Virtual de la UPM.* 108
Figura 23. *Extracto del código HTML de la Web de Politécnica Virtual de la UPM.* 109
Figura 24. *Entorno del sistema.* 114
Figura 25. *Caso de uso general de la aplicación* 116
Figura 26. *Caso de uso para consultar calificaciones de exámenes.* 117
Figura 27. *Caso de uso para consultar el horario de clases.* 119
Figura 28. *Diagrama de actividad para la consulta de calificaciones.* 121
Figura 29. *Diagrama de actividad para la consulta del horario de clases.* 122
Figura 30. *Diagrama de clases* 124
Figura 31. *Diagrama de secuencia de consulta de horario de clases.* 127
Figura 32. *Diagrama de secuencia para la consulta de calificaciones.* 129
Figura 33. *Primer paso.* 131
Figura 34. *Resultado final.* 132
Figura 35. *Primer paso.* 133

Índice de figuras

Figura 36. *Resultado final.* 134
Figura 37. *Escritorio del dispositivo y pantalla principal de la aplicación.* 137
Figura 38. *Solicitud de datos de autenticación.* 138
Figura 39. *Pantalla principal de preferencias.* 139
Figura 40. *Modificar preferencias.* 140
Figura 41. *Modificar usuario y modificar contraseña* 140
Figura 42. *Mostrar preferencias.* 141
Figura 43. *Borrar preferencias.* 142
Figura 44. *Opción Acerca de* 143

Introducción

La evolución de las **Tecnologías de la Información y las Comunicaciones (TIC)**, además de su progresiva introducción en muchos ambientes de la vida ha posibilitado la oportunidad de que las personas puedan tener acceso a cualquier información y en cualquier lugar, desde la palma de su mano gracias a dispositivos móviles como los *Smartphone*. Esto ha provocado que el acceso a **Internet** y su uso se hayan convertido en partes fundamentales de la vida cotidiana.

Gracias esta evolución de las tecnologías y al cambio social producido por las mismas, esta posibilidad, de la que se ha hablado, se ha convertido en una necesidad en muchos aspectos de la vida como la consulta de información relacionada con la persona interesada.

Desde que aparecieran los primeros equipos capaces de acceder a la Red de Redes hasta los dispositivos existentes hoy en día se ha producido un cambio revolucionario en este campo y, actualmente, pocos dispositivos existen que no soporten la conexión a Internet. Esto ha provocado que se creen dispositivos móviles cada vez más dependientes de Internet hasta el punto de que si no soportaran su uso prácticamente sólo servirían para llamar y poco más.

Junto al avance de los dispositivos móviles y la creación de sistemas operativos para ellos, ha sido posible la implementación de aplicaciones para los mismos con el fin de facilitar las necesidades de sus usuarios hasta tal punto de poder realizar cualquier consulta a través de Internet sin tener que recurrir al ordenador de sobremesa o al ordenador portátil; se han ido convirtiendo en ordenadores de bolsillo.

Actualmente casi todas las personas tienen algún tipo de dispositivo móvil ya sea Smartphone o Tablet e incluso muchas de ellas tienen ambos. Entre los sistemas operativos móviles que incorporan estos dispositivos, Android ha llegado a convertirse en uno de los dos más utilizados en todo el mundo, por tanto, la creación de aplicaciones para él, que ayuden a sus usuarios, se ha convertido en fundamental y ya existen empresas dedicadas exclusivamente a la creación de aplicaciones (Apps) para Android.

Actualmente, consultar mediante el Smartphone información de importancia para el interesado es algo fundamental, sin embargo, algunas veces, esta tarea se convierte en algo engorroso debido a la introducción de información que no tiene nada que ver con la deseada y que ralentiza su búsqueda o que simplemente aparece junto a ella provocando una posible confusión en el usuario.

A raíz de esta problemática surgió la técnica de web scraping, que elimina esta información secundaria y presenta solo la información que el cliente desea visualizar.

Objetivos

Con el desarrollo de este proyecto se pretende proponer una metodología que permita realizar web scraping en aplicaciones para Android, entendiendo en el funcionamiento de esta técnica, así como desarrollar en un caso práctico la metodología propuesta mediante la implementación de una aplicación que permita consultar algunos de los datos académicos más importantes para un alumno de la ETSIST de la UPM.

Asimismo, se ha conseguir conocimientos sobre el funcionamiento de Android y la implementación de aplicaciones para el mismo.

Por otra parte, se pretende también conocer las diferentes herramientas que permitirán llevar a cabo esta técnica, así como el protocolo HTTP, usado para la comunicación con el servidor, y HTML, usado para la creación de páginas web.

Estructuración de la memoria

Este proyecto se desarrolla en 7 bloques o capítulos.

En el primer bloque se describirán las tecnologías existentes alrededor de web scraping así como el propio web scraping y su estado del arte. Además se describirá el sistema operativo Android y su funcionamiento.

El segundo bloque se centrará en la explicación del término Metodología de desarrollo de Software, además de detallar diferentes tipos de metodologías y describir algunos ejemplos de cada tipo. También se detallará la metodología propuesta, así como, las diferentes fases que la componen.

El tercer bloque abordará el diseño, e implementación de la aplicación UPMdroid siguiendo los pasos detallados en el segundo bloque e identificando los requisitos que debe cumplir dicha aplicación.

El cuarto bloque se compondrá de las conclusiones finales además de los posibles trabajos futuros alrededor de este proyecto y de la aplicación implementada.

Introducción

El bloque cinco describirá un anexo compuesto de un breve manual de usuario sobre las opciones que presenta la aplicación.

Por último, el bloque seis consistirá en un glosario con los términos usados, y en el bloque siete se especificará la bibliografía utilizada para la realización de este proyecto.

BLOQUE 1
ANÁLISIS DE LAS TECNOLOGÍAS USADAS

1.1 Navegador web

Un navegador web es una aplicación que opera a través de Internet, interpretando la información de archivos y sitios web para que éstos puedan ser leídos.

La funcionalidad básica de un navegador web es permitir la visualización de documentos de texto, posiblemente con recursos multimedia incrustados. Los documentos pueden estar ubicados en el equipo del usuario, pero también pueden estar en cualquier otro dispositivo que esté conectado a dicho equipo o a través de Internet, y que tenga los recursos necesarios para la transmisión de los documentos o "páginas web" (un software servidor web).

El seguimiento de enlaces de una página a otra, ubicada en cualquier computadora conectada a Internet, se llama *navegación*, de donde se origina el nombre *navegador*.

Así pues lo que hace un navegador web es descargar documentos y mostrarlos en pantalla. En la actualidad, no solamente descarga este tipo de documentos sino que muestra con el documento sus imágenes, sonidos e incluso vídeos *streaming* en diferentes formatos y protocolos. Además, permiten almacenar la información en el disco o crear marcadores de las páginas más visitadas.

Los primeros navegadores web sólo soportaban una versión muy simple de HTML (Actualmente existe hasta la versión 5.0). El rápido desarrollo de los navegadores web propietarios condujo al desarrollo de dialectos no estándares de HTML y a problemas de interoperabilidad en la web.

Tipos de navegadores

Actualmente existen multitud de navegadores web, pero son 3 los que son ampliamente utilizados por la mayoría de usuarios de PC: Internet Explorer (IE), Mozilla Firefox y Google Chrome.

Internet Explorer es el navegador que proporciona Microsoft junto a su sistema operativo Windows; nació en 1995 a partir del código fuente de Spyglass Inc. Mosaic; llegó a tener una cuota de mercado del 95% entre 2002 y 2003. Actualmente esa cuota se mantiene en un 55% que hace que siga siendo el navegador más utilizado.

1.1 Navegador Web

Mozilla Firefox es un navegador web libre, multiplataforma y de código abierto desarrollado por la Corporación Mozilla y la Fundación Mozilla. Posee un 20% de cuota de mercado aproximadamente, convirtiéndose en el segundo navegador web más utilizado.

Google Chrome, por su parte, es el navegador web desarrollado por Google que vio la luz en 2008, es de código abierto y disponible gratuitamente al igual que Firefox. Posee una cuota de mercado del 16,5%; esto hace que sea el tercer navegador en discordia.

Para los usuarios de Apple el más utilizado es el que proporciona dicha compañía: Safari, que actualmente tiene una cuota de mercado del 5,5% pasando a ser el cuarto navegador más utilizado.

Cabe destacar que existen muchos más navegadores de diferentes compañías como Opera de Opera Software cuya cuota de mercado es significativamente inferior a la de los tres primeros, pero que no por ello dejan de ser herramientas útiles; en muchas ocasiones no son tan conocidos como otros.

Entrando ya en el ámbito de los dispositivos móviles, concretamente en el de los smartphones, debido a la gran demanda que tienen estos dispositivos se ha hecho necesario proporcionar distintos servicios a sus usuarios. Uno de los principales es el acceso a Internet ya que al auge de las tecnologías enfocadas a la red no ha hecho más que dispararse al tiempo que se han ido convirtiendo en necesarias tanto en el ámbito laboral como en el de tiempo libre.

Así pues, el desarrollo de navegadores web ha tenido que adaptarse también estas plataformas móviles. Mozilla, Google y Opera Software sacaron versiones de Firefox, Chrome y Opera para la mayoría de terminales móviles, sin embargo Microsoft lanzó una versión de Internet Explorer para su sistema operativo en estos dispositivos perdiendo gran cuota de mercado en este campo. Apple por su parte, también lanzó Safari, sólo para los dispositivos fabricados por esta empresa. Asimismo otras compañías aprovecharon para implementar navegadores web como Dolphin Browser exclusivo en estas plataformas.

1.2 HTML

La gran mayoría de los documentos o páginas web están escritos en lenguaje HTML (Hyper Text Markup Language). HTML es un "lenguaje de marcado" o de "etiquetas" empleado para dar forma al contenido de las páginas web. También puede describir, hasta un cierto punto, la apariencia de un documento, y puede incluir o hacer referencia a un tipo de programa llamado *script*, el cual puede afectar el comportamiento del navegador web. Suele confundirse con un lenguaje de programación donde se podrían procesar entradas, incluir condiciones, etc., pero es el que permitirá visualizar la información generada por otros lenguajes de programación web.

Haciendo un poco de historia se puede justificar la necesidad de crear un lenguaje como HTML. En un principio la intención de Internet era la de servir documentos simples de texto. Los primeros navegadores estaban basados en texto y la inclusión de imágenes fue un gran avance. Se necesitaba un lenguaje capaz de soportar el diseño de estas novedades y claro está la lectura por parte de las aplicaciones cliente. Actualmente los servicios que aporta Internet van desde el envío de mensajes de correo electrónico hasta ver la televisión. Internet se ha convertido en mucho más que un mecanismo para el transporte de documentos de texto simple y se necesita un lenguaje acorde a las consecuencias. El objetivo de HTML es el de soportar los nuevos servicios.

1.2.1 Historia de HTML

La expresión "lenguaje de marcas" apareció por primera vez en 1967 mediante el desarrollo del estándar GenCode, dirigido por William Tunnicliffe, y destinado a la industria editorial. Sin embargo, es Charles Goldfarb, investigador de IBM, el considerado como padre de los lenguajes de marcas al diseñar GML (Generalized Markup Language), junto con Edward Mosher y Raymond Lorie, destinado en un principio a mantener grandes cantidades de documentos. Así pues nació un nuevo lenguaje centrado en definir la estructura del texto y no su presentación visual.

Posteriormente GML se fue extendiendo y popularizándose hasta que en 1986 y tras un largo proceso, la ISO (Organización Internacional para la Estandarización) lo publicó, con rango de estándar internacional pasándose a llamar SGML (Standard Generalized Markup Language) en la ISO 8879.

1.2 HTML

SGML especifica la sintaxis para la inclusión de marcas además de la sintaxis del documento que especifica qué etiquetas están permitidas y donde. Básicamente, define las reglas de etiquetado de documentos y no impone conjuntos de etiquetas en especial. Esto permitía al autor emplear cualquier marca que quisiera eligiendo los nombres para las etiquetas que considerara oportunos tanto por el tema del documento como por el idioma. Por tanto SGML se convirtió en un metalenguaje ya que especificaba reglas y características de otros lenguajes.

En 1991 surge la primera descripción de HTML que es considerada como una ampliación de SGML.

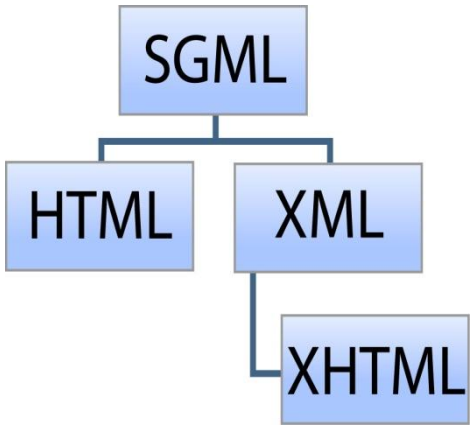

Figura 1. Familia SGML

HTML es un pilar fundamental de la WWW (World Wide Web) ya que se basa en la distribución de información basada en hipertexto que es accesible a través de Internet y que se puede visualizar mediante un navegador web.

La Web (WWW) fue creada en 1989 por Tim Berners-Lee y Robert Cailiau y publicada su propuesta en 1991. Berners-Lee también escribió el primer navegador web: WorldWideWeb en 1991 que posteriormente pasaría a llamarse Nexus. Además se caracterizó por desarrollar un sistema de identificadores únicos globales que permitía aunar hipertexto con Internet: el URI (Uniform Resource Identifier) más comúnmente conocido como URL.

En 1993 se lanza la primera propuesta llamada HTML+ para convertir HTML en un estándar por parte del organismo IETF. Aunque se lograron avances significativos como la definición de las etiquetas para imágenes, tablas y formularios, y a pesar de ser la base más parecida a las especificaciones actuales la propuesta no consiguió convertirse en un estándar.

El primer estándar oficial se logra en 1995. El organismo IETF organiza un grupo de trabajo de HTML y consigue publicar ese mismo año, el estándar HTML 2.0.

1.2 HTML

En 1994 se crea el consorcio W3C y a partir de ahora será este grupo será el que publique los estándares. Ese mismo año se propone el borrador de HTML 3.0, con el se introdujeron nuevas capacidades como crear tablas de una forma más sencilla o mostrar elementos matemáticos complejos entre otras. Se diseño de forma compatible con el estándar oficial HTML 2.0 pero la falta de apoyos de los fabricantes de navegadores web y la complejidad del estándar hizo que se abandonara el proyecto. La versión de HTML 3.2 – HTML 3.1 nunca fue propuesto oficialmente – se publicó en Enero de 1997. Esta versión incorporaba los últimos avances desarrollados inicialmente por los navegadores web Netscape y Mosaic. Se incorporan applets de Java.

HTML 4.0 se publicó en 1997 teniendo una versión corregida de la publicación en 1998. Supone un gran salto desde las versiones anteriores. Las novedades más destacadas se encuentran las hojas de estilos CSS, la posibilidad de incluir pequeños programas o *scripts* en las páginas web, mejora de la accesibilidad de las páginas diseñadas, tablas complejas y mejoras en los formularios.

Desde la publicación de HTML 4.01 – que no incluyo novedades significativas – el W3C se centro en el desarrollo del estándar XHTML. El desarrollo de HTML5 comenzó gracias a la preocupación de las compañías Apple, Mozilla y Opera. La falta de actividad de W3C – centrada en XHTML – en HTML hizo que en 2004 estas empresas formarán una organización llamada WHATWG para impulsar el nuevo estándar.

En 2006, el W3C se interesó en el desarrollo de HTML5 y un año después se unió al grupo de trabajo del WHATWG para retomar la actividad de HTML. En 2008 se publica el primer borrador oficial. Que servirá para que los desarrolladores comiencen a programar sobre HTML5. Finalmente W3C se centra totalmente en HTML5 tras el fracaso de XHTML 2.0 en 2009.

1.2.2 Conceptos básicos del lenguaje

El lenguaje HTML puede ser implementad con cualquier editor de textos básico. Existen además, otros editores para la creación de páginas web, estos editores permiten ver el resultado de lo que se está realizando en tiempo real, a medida que se va desarrollando el documento.

HTML utiliza etiquetas que consisten en breves instrucciones de comienzo y final, mediante las cuales se determinan la forma en la que debe aparecer en su navegador el texto, así como también las imágenes y los demás elementos, en la pantalla del ordenador.

1.2 HTML

Toda etiqueta se identifica porque está encerrada entre los signos menor que y mayor que (<>) y algunas tienen atributos que pueden tomar algún valor. En general las etiquetas se aplicarán de dos formas especiales: Se abren y se cierran; negrita, no pueden abrirse y cerrarse; como <hr />.

Las etiquetas más básicas son:

- ✓ **<html>** Define el inicio del documento HTML. Indica al navegador que lo que viene a continuación debe ser interpretado como código HTML.
- ✓ **<script>** Incrusta un script en una web.
- ✓ **<head>** Define la cabecera del documento HTML; esta cabecera suele contener información como el título de la ventana del navegador.

  Dentro de la cabecera <head> podemos encontrar:

  - **<title>** Define el título de la página. El título aparece en la barra de título encima de la ventana.
  - **<link>** Para vincular el sitio a hojas de estilo o iconos.
  - **<style>** Para colocar el estilo interno de la página; ya sea usando CSS, u otros lenguajes similares.

- ✓ **<body>** Define el contenido principal o cuerpo del documento. Esta es la parte del documento HTML que se muestra en el navegador. Dentro de esta etiqueta pueden definirse propiedades comunes a toda la página, como color de fondo y márgenes.

1.2.3 Principales tecnologías alrededor de HTML

1.2.3.1 CSS

Las *hojas de estilo en cascada* o CSS es un lenguaje usado para definir la presentación de un documento estructurado escrito en HTML o XML - y por extensión en XHTML -. El W3C (World Wide Web Consortium) es el encargado de formular la especificación de las hojas de estilo que servirán de estándar para los agentes de usuario o navegadores.

Gracias las CSS podemos separar la estructura del documento de su presentación pudiendo estar en el mismo documento HTML o separado.

1.2.3.2 JavaScript

JavaScript es un lenguaje de programación interpretado que permite que el cliente pueda hacer cambios en un sitio web sin tener que recargarlo cada vez que quiera modificar algo además de permitir mejoras en la interfaz de usuario y páginas web dinámicas.

JavaScript se diseñó con una sintaxis similar al C, aunque adopta nombres y convenciones del lenguaje de programación Java. Sin embargo Java y JavaScript no están relacionados y tienen semánticas y propósitos diferentes.

Tradicionalmente se venía utilizando en páginas web HTML para realizar operaciones y únicamente en el marco de la aplicación cliente, sin acceso a funciones del servidor.

1.3 Servidor

1.3.1 Concepto

En el ámbito informático, se denomina servidor a un nodo, perteneciente a una red, que provee servicios a otros nodos de la red llamados clientes. Dicho nodo contiene una aplicación informática que realiza tareas en beneficio de otras aplicaciones llamadas clientes; un navegador web es considerado una aplicación cliente.

Los servicios que presta habitualmente un servidor son los servicios de archivos, que permiten a los clientes almacenar y acceder a los archivos de una computadora y los servicios de aplicaciones, que realizan tareas en beneficio directo del usuario final.

Asimismo un servidor puede ser un proceso que entrega información o sirve a otro proceso, ya que un mismo ordenador puede cumplir simultáneamente las funciones de cliente y de servidor cuando un proceso cliente solicite algo a un proceso servidor que se encuentre en el mismo ordenador.

Sin embargo, en la mayoría de casos el servidor se encuentra en otro ordenador diferente del cliente y es este el que, estableciendo una comunicación, accede a los recursos del servidor.

1.3.2 Tipos de servidores

A continuación, se presentan una serie de servidores muy utilizados a día de hoy, todos ellos especializados en un determinado servicio.

- **Servidor de Correo**: los servidores especializados en correo electrónico suelen utilizar los protocolos POP3 y SMTP. Son los encargados de almacenar, enviar y recibir los e-mails de los clientes.

- **Servidores Telnet**: un servidor telnet permite a los usuarios entrar en un ordenador huésped y realizar tareas como si estuviera trabajando directamente en ese ordenador.

- **Servidor Proxy**: estos servidores actúan como intermediarios de forma que el servidor receptor de una petición no conozca el emisor de la misma.

- **Servidor de Base de Datos**: Da servicios de almacenamiento y gestión de bases de datos a sus clientes. Una base de datos es un sistema que nos permite almacenar grandes cantidades de información.

- **Servidor Clúster**: Son servidores de back-up, especializados en el almacenamiento de la información teniendo grandes capacidades de almacenamiento y permitiendo evitar la pérdida de la información en otros servidores por problemas inesperados.

- **Servidor de imágenes**: Permiten alojar gran cantidad de imágenes sin consumir recursos de nuestro servidor web en almacenamiento. Algunos de ellos son gratuitos y frecuentemente utilizados en la red como por ejemplo www.imageshack.us, www.theimagehosting.com, www.flickr.com, picasaweb.google.com.

- **Servidores de Audio/Video:** Añaden capacidades multimedia a los sitios web permitiéndoles mostrar contenido multimedia en forma de flujo continuo (***streaming***).

- **Servidor DNS:** Siglas de Domain Name System. Estos servidores son los encargados de traducir y encontrar las direcciones web (URL) para poder conectar con ella.

- **Servidor FTP:** Siglas File Transfer Protocol, protocolo utilizado para la transferencia de archivos entre un cliente y un servidor, permitiendo al cliente descargar el archivo desde el servidor o al servidor recibir un archivo enviado desde un cliente. Por defecto FTP no lleva ningún tipo de encriptación permitiendo la máxima velocidad en la transferencia de los archivos, pero puede presentar problemas de seguridad, por lo que muchas veces se utiliza SFTP que permite un servicio de seguridad encriptada.

- **Servidor DHCP:** Siglas de *Dynamic Host Configuration Protocol. Este servidor contiene y* consulta una lista de direcciones IP dinámicas y las asigna a los clientes conforme éstas van estando libres, manteniendo en todo momento información de la relación de direcciones asociadas, tiempo de uso y dirección del siguiente poseedor de una misma dirección.

1.3.3 Servidor web

Un servidor web está formado por una aplicación informática encargada de realizar conexiones bidireccionales y/o unidireccionales y síncronas o asíncronas con el cliente generando una respuesta en cualquier lenguaje o Aplicación del lado del cliente.

Se considera servidor web al tipo de servidor que ofrece información a los clientes a través de Internet, atendiendo peticiones de estos mediante una comunicación establecida por el cliente o por el mismo servidor.

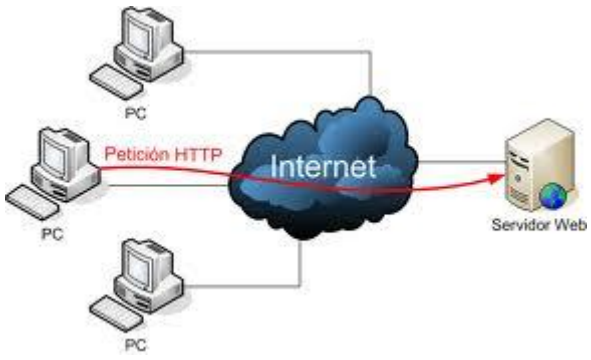

Figura 2. Servidor web

Existen tres clases diferentes de servidores web que escogeremos según el tipo de servicio que necesitemos ofrecer:

- **Servidor compartido:** Estos servidores son los más comunes, utilizados principalmente por pequeños y medianos negocios ya que es la forma más económica de poder albergar multitud de páginas aunque en algunas ocasiones la velocidad de acceso puede verse reducida sin demasiadas visitas. Algunos de los servicios de hosting compartido más conocidos son 1&1, Bluehost.com, Hostmonster.com, Dreamhost.com y Hostgator.com. Lo más recomendable es elegir un host que este en nuestro continente o país, para reducir el tiempo de respuesta.

- **Servidor dedicado:** Utilizados en páginas con gran cantidad de tráfico y de visitas además de un alto consumo de datos. Cada cliente dispone de un servidor dedicado en el que puede definir y ajustar características como el tamaño del disco duro, cantidad de RAM, tipo de procesador, etc. El servicio brindado por estos servidores es de mejor calidad pero a su vez es el más costoso ya que puede implicar tener que poseer el propio hardware.

1.3 Servidor

- **Servidor Privado Virtual:** Es un servidor virtual o VPS (Virtual Private Servers), que se ejecuta en una misma computadora junto a otros servidores de su tipo, y que pretende brindar los mismos beneficios que un servidor dedicado a bajo coste. También existen algunos gratuitos, pero no son recomendables si se quiere asegurar un mínimo de calidad en el servicio que se pretende otorgar.

Actualmente en el mercado disponemos de multitud de distribuciones para implementar en nuestros servidores web, a continuación se detallan los más comunes.

- **Apache**: es el más utilizado en todo el mundo, es gratuito y de código abierto, lo que le hacen un servidor multiplataforma capaz de correr en cualquier SSOO. Se caracteriza por ser modular y extensible y gracias a su popularidad existen comunidades de ayuda y soporte.

- **Microsoft IIS**: servidor exclusivo para sistemas Windows lo que limita mucho su utilización en otros SSOO ya que se tendrían que utilizar máquinas virtuales para emular Windows.

- **Sun Java System Web Server:** producto perteneciente a Sun Microsystems, es multiplataforma, y recientemente se han distribuirlo algunas licencias de código abierto como BSD.

- **Ngnix:** servidor Web muy ligero compatible con sistemas Unix y Windows. Se ha convertido en el 4º servidor web más popular de la red y también se distribuye bajo licencia BSD. Es utilizado en sitios populares como WordPress, Ohloh, SourceForge, TorrentReactor y partes de Facebook.

- **Lighttp:** este servidor Web es otro de los más ligeros que hay en el mercado. Se caracteriza por ser rápido, seguro y ligero y está especialmente diseñado para hacer cargas pesadas, utilizando poca RAM y poca CPU. Es utilizado por páginas que soportan gran cantidad de tráfico diaria como son Youtube y Wikipedia, es gratuito y se distribuye bajo licencia BSD.

1.4 Interacción entre navegador y servidor

La interactuación entre un cliente (navegador web) y el servidor – normalmente un servidor remoto – se realiza mediante el mecanismo petición-respuesta iniciado por el cliente hacia el servidor, solicitando un documento, unos archivos o cualquier otra información que el cliente considere necesario.

Este mecanismo de comunicación entre cliente y servidor se realiza siguiendo un protocolo. Cuando hablamos de protocolo nos referimos a un conjunto de reglas y normas que permiten que dos o más entidades de un sistema de comunicación se comuniquen entre ellos para transmitir información por medio de cualquier tipo de variación de una magnitud física. Un protocolo suele ser un estándar, aprobado por las organizaciones pertinentes, que define la sintaxis, semántica y sincronización de la comunicación, así como posibles métodos de recuperación de errores. La organización que regula los diferentes estándares para protocolos de comunicación es la ITU (Unión Internacional de Telecomunicaciones) en el Sector de Normalización de las Telecomunicaciones o ITU-T.

Generalmente para la interactuación entre cliente y servidor se usa el protocolo HTTP.

1.4.1 Protocolo HTTP

HTTP es el acrónimo de Hypertext Transfer Protocol. Es un protocolo del nivel de aplicación del modelo OSI orientado a transacciones y sigue el esquema petición-respuesta entre un cliente y un servidor.

Fue desarrollado por el World Wide Web Consortium y la Internet Engineering Task Force, colaboración que culminó en 1999 con la publicación de una serie de RFC, el más importante de ellos es el RFC 2616 que especifica la versión 1.1. HTTP define la sintaxis y la semántica que utilizan los elementos de software de la arquitectura web (clientes, servidores, proxies) para comunicarse. Al cliente que efectúa la petición (un navegador web) se le conoce como "user agent" (agente de usuario). A la información transmitida se la llama recurso y se la identifica mediante un localizador uniforme de recursos (URL). Los recursos pueden ser archivos, el resultado de la ejecución de un programa, una consulta a una base de datos, la traducción automática de un documento, etc.

1.4 Interacción entre navegador y servidor

Es un protocolo sin estado, es decir, que no guarda ninguna información sobre conexiones anteriores. El desarrollo de aplicaciones web necesita frecuentemente mantener estado. Para esto se usan las cookies, que es información que un servidor puede almacenar en el sistema cliente. Esto le permite a las aplicaciones web instituir la noción de "sesión", y también permite rastrear usuarios ya que las cookies pueden guardarse en el cliente por tiempo indeterminado.

Está basado en peticiones-respuestas entre cliente y servidor. Una petición o respuesta HTTP está formada por un encabezado seguido, opcionalmente, por una línea en blanco y algún dato. El encabezado especificará cosas como la acción requerida del servidor, o el tipo de dato retornado, o el código de estado.

El uso de campos de encabezados enviados en las transacciones HTTP le da gran flexibilidad al protocolo. Estos campos permiten que se envíe información descriptiva en la transacción, permitiendo así la autenticación, cifrado e identificación de usuario.

HTTP ha pasado por múltiples versiones del protocolo, muchas de las cuales son compatibles con las anteriores. El cliente le dice al servidor al principio de la petición la versión que usa, y el servidor usa la misma o una anterior en su respuesta. La versión que se usa actualmente es la 1.1.

Tipos de peticiones

HTTP especifica 8 tipos de peticiones (HTTP request), por parte del cliente hacia el servidor, que indican la acción a realizar sobre el recurso identificado:

- **GET**
  Esta petición se realiza cuando se quiere una representación del recurso especificado. Para transacciones de autenticación mediante usuario y contraseña no debería usarse este tipo de petición ya que las credenciales irían agregadas, sin ocultar, como parámetros a la URL.

- **HEAD**
  Se realiza una petición, por parte de cliente, para que el servidor envíe una respuesta idéntica a la de la petición GET, pero sin el cuerpo de la respuesta, es decir, solo el encabezado. Es útil para obtener los encabezados de la respuesta sin transportar todo el contenido.

- **POST**
  Esta petición se realiza cuando se tienen que enviar datos al servidor como usuario y contraseña, datos de formularios que solicita el servidor, etc. y que no corresponden a datos del encabezado. También se usa en algunas ocasiones como sustituto de una petición GET por motivos de seguridad. Estos datos a enviar viajan en el cuerpo de la petición a diferencia de otras peticiones que solo tienen el encabezado.

- **PUT**
  Sube o carga de un recurso especificado (archivo); es el camino más eficiente para subir archivos a un servidor, esto es porque en POST utiliza un mensaje multi-parte (encabezado y cuerpo) y el mensaje es decodificado por el servidor. En contraste, la petición PUT permite escribir un archivo en una conexión socket establecida con el servidor.

- **DELETE**
  Borra el recurso especificado.

- **TRACE**
  Esta petición solicita al servidor que envíe de vuelta un mensaje de respuesta, que en el cuerpo de dicha respuesta contenga todos los datos que reciba del mensaje de solicitud. Se utiliza con fines de comprobación y diagnóstico.

- **OPTIONS**
  Esta petición solicita que se devuelva en la respuesta todas las peticiones HTTP que soporte el servidor para un determinado recurso. Suele ser usada para comprobar la funcionalidad de un servidor web en general.

- **CONNECT**
  Se utiliza para saber si se tiene acceso a un equipo (host) bajo condiciones especiales, no necesariamente llega la petición al servidor.

El servidor envía al cliente como respuesta:

- Un código de estado que indica si la petición fue correcta o no. Los códigos de error típicos indican que el archivo solicitado no se encontró, que la petición no se realizó de forma correcta o que se requiere autenticación para acceder al archivo.

- La información propiamente dicha. Como HTTP permite enviar documentos de todo tipo y formato, es ideal para transmitir multimedia, como gráficos, audio y video. Esta libertad es una de las mayores ventajas de HTTP.
- Información sobre el objeto que se retorna.

Códigos de respuesta

A continuación se muestran los códigos de respuesta definidos en HTTP; cada respuesta del servidor a una petición llevará uno de estos códigos dependiendo de la petición y la acción a realizar sobre el recurso deseado:

- 100 ó 111: Indica conexión rechazada por el servidor.
- 200: Operación exitosa, OK.
- 201 – 203: Operación exitosa, información no oficial.
- 204: Operación exitosa, sin contenido (cuerpo del mensaje).
- 205: Operación exitosa, contenido para recargar.
- 206: Operación exitosa, contenido parcial.
- 301: Redirección, recurso mudado temporalmente.
- 302: Redirección, encontrado.
- 303: Redirección, vea otros.
- 304: Redirección, no modificado.
- 305: Redirección, utilice un proxy.
- 307: Redirección, redirección temporal.
- 400: Error cliente, solicitud incorrecta.
- 401: Error cliente, no autorizado.
- 402: Error cliente, pago requerido.
- 403: Error cliente, prohibido.
- 404: Error cliente, recurso no encontrado.
- 409: Error cliente, conflicto.
- 410: Error cliente, recurso ya no disponible.
- 412: Error cliente, falló precondición.
- 500: Error servidor, error interno.
- 501: Error servidor, no implementado.
- 502: Error servidor, pasarela incorrecta.
- 503: Error servidor, servicio no disponible.
- 504: Error servidor, tiempo de espera de la pasarela agotado.
- 505: Error servidor, versión de HTTP no soportada.

1.4 Interacción entre navegador y servidor

Al protocolo HTTP se le asigna habitualmente el puerto TCP 80. Las peticiones al servidor suelen realizarse mediante HTTP utilizando el método de petición GET en el que el recurso se solicita a través de la URL al servidor web.

Estructura de una petición HTTP

HTTP define una serie de campos o parámetros que van en el encabezado de cada petición que podrán o no ser rellenados en función del tipo de petición que se realice al servidor. Estos campos son:

- **Petition type:** Es la primera línea de una petición. Contiene el tipo de petición HTTP que se va realizar al servidor. Su estructura es: <TIPO PETICIÓN> <URL> <VERSION HTTP>. Por ejemplo:
  GET /index.html HTTP/1.1
- **Referer:** Especifica la URL desde la que se hizo la petición. Por ejemplo: Referer: comment.php
- **Content-Length:** Especifica la longitud en bytes de los datos enviados en el cuerpo de la petición. En la petición GET no se incluye al no tener cuerpo. Por ejemplo: Content-Type: 57
- **Origin:** Especifica la URL principal donde está el recurso. Por ejemplo: Origin: http://www.marca.com
- **User-Agent:** Especifica el identificador del navegador Web desde el cual se hizo la petición. Por ejemplo: User-Agent: Mozilla/5.0 (Windows; U; Windows NT 6.1; en-US)
- **Content-Type:** Indica el formato de los datos enviados en el cuerpo de la petición. Puede ser de tipo MIME, que son una serie de especificaciones dirigidas al intercambio de todo tipo de archivos (video, texto, audio, etc.) a través de Internet, o de tipo aplicación. En la petición GET tampoco se usa este campo. El formato más común en este campo es: Content-Type: application/x-www-form-urlencoded.
- **Accept:** Indica el formato de datos que se espera o que se admite en el cuerpo de la respuesta. Puede ser uno o varios tipos de formatos separados por comas. Por ejemplo: Accept: application/xml, application/xhtml+xml.
- **Accept-Language:** Indica el código del lenguaje esperado en la respuesta. Por ejemplo: Accept-Language: es-ES,es;q=0.8.
- **Accept-Charset:** Indica la codificación esperada en la respuesta. Por ejemplo: Accept-Charset: ISO-8859-1,utf-8.

- **Cookie:** Indica el identificador de sesión en la petición, para sitios que lo requieran. Por ejemplo: Cookie: PHPSESSID=gm0ugf96iojuldio8i51u92716
- **Accept-Encoding:** Indica el tipo de compresión de datos que se espera en la respuesta. No todos los navegadores envían este campo. Por ejemplo: Accept-Encoding: gzip,deflate,sdch

A continuación de estos campos o cabeceras se incluye el cuerpo de la petición en aquellas que lo necesiten:

- **Content:** Contenido de la petición.

Cada petición no tiene por qué incluir todos los campos anteriormente descritos; generalmente se incluyen los campos **Petition-Type**, **Accept**, **Accept-Languaje**, **Accept-Charset** y **Accept-Encoding**. Pero también puede añadirse cualquier otro campo cuando sea necesario.

Tanto GET como POST son las peticiones más usadas a la hora de comunicarse con un servidor web. A continuación se muestra un ejemplo de petición GET y otro de petición POST:

- Ejemplo de petición GET:

  GET /index.html HTTP/1.1
  Accept: text/html
  User-Agent: Mozilla/5.0 (Windows; U; Windows NT 6.1; en-US)

  Se puede observar que es una petición sólo con encabezado, es decir, sin cuerpo en el mensaje. La petición GET suele ser así.

  Respuesta del servidor:

  HTTP/1.1 200 OK
  Date: Fri, 31 Dec 2003 23:59:59 GMT
  Content-Type: text/html
  Content-Length: 1221

  <html>
  <body>
  <h1>Página principal</h1>
  (Contenido)
  .
  .

.
</body>
</html>

- Ejemplo de petición POST:

  POST www.miweb.org HTTP/1.1
  User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.7.6...
  Content-Type: application/x-www-form-urlencoded
  Content-Length: 40
  Connection: Keep-Alive

  id=20&nombre=Fulanito%20de%20tal

  Se puede observar como entre los campos de la cabecera y el cuerpo del mensaje hay una línea en blanco; esto es interpretado en el servidor como el fin del encabezado del mensaje y el principio del cuerpo del mensaje que es donde están los datos.

Pasos para la comunicación entre cliente y servidor

El navegador por medio de la interfaz de usuario permite al usuario realizar una o varias peticiones web. La interfaz de usuario o entorno de usuario es el conjunto de elementos del navegador que permiten realizar la petición de forma activa.

Elementos del entorno de usuario más comunes en navegadores Web:

- **Hipervínculo enlace o link**: Es una porción de contenido Web, texto, imagen y otros elementos, que enlaza con una dirección Web. Al pulsar un hipervínculo el navegador genera una petición GET automática a la dirección URL de dicho link.
- **Formulario web:** Al realizar el envío satisfactorio de los datos de un formulario, el navegador Web genera una petición GET o POST (comúnmente POST) automática a la par que envía los datos al servidor.
- **Barra de direcciones:** Todos los navegadores incluyen una barra de direcciones mediante la cual puede accederse manualmente a cualquier dirección URL, de modo que el navegador generará una petición GET automática a dicha URL cada vez que el usuario lo desee.
- **Script activo o pasivo:** Cualquier aplicación Javascript tiene acceso al estado del navegador, cómo puede modificar los datos que describen tal estado, de forma

pasiva (sin medio de la intervención del usuario) o de forma activa (mediante alguna acción del usuario).

1. Conexión a dirección DNS:

   Se produce una conexión con un servidor de DNS dado en dirección IP mediante el protocolo TCP. Por lo general, las direcciones que el navegador posee inicialmente son direcciones DNS (direcciones alfanuméricas) que deberá convertir a direcciones numéricas.

2. Resolución de DNS a IP:

   El cliente solicita al servidor DNS la o las direcciones IP correspondientes al servidor donde se encuentra el recurso deseado. El navegador crea una nueva regla y almacena la dirección IP junto a la dirección DNS en su base de datos de reglas DNS.

3. Recuperación de la regla DNS:

   Una vez almacenada a regla se realiza una petición a la base de datos DNS para recuperar los valores de la regla almacenada.

4. Conexión a dirección IP:

   El cliente realiza una conexión con el servidor donde se encuentra el recurso una vez a obtenido su dirección IP.

5. Creación de la petición:

   Se crea la petición HTTP estableciendo la URL, la prioridad de la petición y el método (generalmente GET).

6. Realización de la petición:

   Se envía la petición HTTP a la dirección IP obtenida añadiéndole el puerto TCP por el que se va a enviar; generalmente el 80. El servidor lee las cabeceras y el cuerpo –si hubiera– de la petición y manda la respuesta.

7. Consulta en caché:

   El navegador consulta en el caché de disco si existe una entrada en el caché asociada al recurso que se ha solicitado. Si la entrada no existe se escriben los datos en el cache de disco.

8. Presentación visual del recurso:

   Se concluye el proceso mostrando, si es preciso, la información por pantalla.

El Servidor web se ejecuta en un ordenador manteniéndose a la espera de peticiones por parte de un cliente (un navegador web) y responde a estas peticiones adecuadamente, mediante el envío de una página web que se exhibirá en el navegador o mostrando el respectivo mensaje si se detectó algún error.

1.5 Análisis de la información recibida: Parsing

Generalmente, una vez que el cliente ha obtenido el recurso que buscaba, una página web, etc. el proceso finaliza; sin embargo, si se desea manipular los datos conseguidos, extraerlos y/o utilizarlos para otras aplicaciones o fines, existen técnicas para analizar dicha información.

Una de estas técnicas es el *parseo* o *parsing* de la información. El parseo consiste en analizar la información obtenida leyéndola y almacenándola siguiendo una determinada estructura para una posterior y sencilla manipulación de la misma.

La herramienta que lleva a cabo esta función se denomina *parser* o *parseador.* Por lo general, un parseador forma parte de la estructura de un compilador. Un compilador es una herramienta informática que "traduce" una aplicación escrita en un determinado lenguaje de programación a otro lenguaje de programación interpretable por la máquina (normalmente instrucciones en código máquina), generando a su vez un programa ejecutable con ese código. La función de un parseador dentro de un compilador es analizar el código fuente en el que está escrita la aplicación para su posterior "traducción".

Aunque a primera vista un parseador pueda parecer una herramienta generada automáticamente y que forma parte de un compilador, también existen parseadores que se pueden implementar manualmente, que no tienen por qué encontrarse dentro del compilador y que tienen una función similar a la que tienen en el compilador pero que, sin embargo, son usados para otros fines como procesar un documento y organizar su información de manera que pueda ser extraída fácilmente.

La técnica de parsing también es realizada en el lado del cliente para estructurar la información recibida del servidor. Esta información llega al cliente con uno de los dos formatos más usados para la representación de datos: XML o JSON.

La idea fundamental del modelo Internet es manejar los documentos de forma organizada para facilitar el intercambio y la manipulación de sus datos. A partir de esta idea, en 1986 se lanza unos de los primeros lenguajes destinado a la representación de los datos: el lenguaje SGML o Standard Generalized Markup Language. SGML consiste en un sistema para la organización.

A partir de SGML, nace en 1989 el lenguaje HTML destinado para navegadores Web. Pero es también a partir de SGML el lanzamiento, en 1998, del lenguaje XML por parte de la W3C.

1.5 Análisis de la información recibida: Parsing

Así pues XML es un subconjunto de SGML. Sus principales características son: extensibilidad, porque al ser un lenguaje de etiquetas cada dato está representado por una o varias etiquetas con o sin atributos; es estructurado, es decir, un documento en XML posee una estructura con una jerarquía de etiquetas; y tiene que ser válido al ajustarse dicha estructura a unas determinadas reglas o prototipos de estructuración de los datos (DTD y XML Schemas). Además, separa el contenido –los datos– de su representación –cómo se ven esos datos–.

XML no es una solución de nada por sí mismo, sino un marco que permite crear soluciones. Permite generar un conjunto personalizado de etiquetas que sirven para codificar tipos específicos de información. De un documento XML se puede ver su estructura, pero no el significado de su contenido. XML es un conjunto de reglas para estructurar datos. No es un lenguaje de programación. Es parecido al lenguaje HTML ya que ambos usan etiquetas (< >) y atributos (nombre = "valor"). En HTML tienen un significado para el navegador, pero en XML sólo delimitan piezas de datos.

Un ejemplo de documento XML podría ser:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE Edit_Mensaje SYSTEM "Edit_Mensaje.dtd">

 <menu id="file" value="File">
  <popup>
   <menuitem value="New" onclick="CreateNewDoc()" />
   <menuitem value="Open" onclick="OpenDoc()" />
   <menuitem value="Close" onclick="CloseDoc()" />
  </popup>
 </menu>
```

Por su parte JSON o JavaScript Object Notation es un formato ligero para el intercambio de datos. La simplicidad de JSON ha dado lugar a la generalización de su uso, especialmente como alternativa a XML. Una de las supuestas ventajas de JSON sobre XML como formato de intercambio de datos en este contexto es que es mucho más sencillo escribir un parser de JSON.

Se emplea habitualmente en entornos donde el tamaño del flujo de datos entre cliente y servidor es limitado y de vital importancia cuando la fuente de datos es explícitamente de fiar y donde no es importante el no disponer de procesamiento XSLT para manipular los datos en el cliente.

1.5 Análisis de la información recibida: Parsing

JSON es independiente del lenguaje, menos pesado que XML, es un formato compuesto por texto plano, posee una estructura jerárquica que facilita el intercambio de datos complejos, no es un lenguaje de marcas o etiquetas, ni define funciones no tiene validador y no es extensible.

JSON tiene dos estructuras:

- Una colección de pares *nombre* ⟷*valor:* Se trata como un objeto. Comienza con { y acaba con }. Sus propiedades se separan con comas , y el nombre y el valor están separados por dos puntos :
- Una lista ordenada de valores: se tratará como un array. Comienza con [y acaba con] . Los valores se separan con coma ,

Un ejemplo de un documento JSON podría ser:

```
{"menu": {
  "id": "file",
  "value": "File",
  "popup": {
    "menuitem": [
      {"value": "New", "onclick": "CreateNewDoc()"},
      {"value": "Open", "onclick": "OpenDoc()"},
      {"value": "Close", "onclick": "CloseDoc()"}
    ]
  }
 }
}
```

Tanto XML como JSON son legibles por el ser humano aunque esa no es su principal funcionalidad, ambos tienen una sintaxis muy simple, son jerárquicos en cuanto a representación de la información, son independientes del lenguaje de programación usado y son fáciles de crear y de manipular.

Precisamente la técnica de parsing tiene que ver con la manipulación de este tipo de archivos, para ello existen dos tecnologías ampliamente usadas: SAX y DOM.

1.5 Análisis de la información recibida: Parsing

1.5.1 SAX

Son las siglas de "Simple API for XML", originalmente, una API únicamente para el lenguaje de programación Java, que después se convirtió en la API estándar *de facto* para usar XML en JAVA. Existen versiones de SAX no sólo para JAVA, sino también para otros lenguajes de programación (como Python).

Permite procesar un documento XML elemento a elemento de manera orientada a eventos. Su función es serializar el contenido de un documento XML y enviarlo a la aplicación. Crea un parser que sirve para validar el documento XML, permite modificar el formato, no construye una representación en memoria del contenido del documento XML y es rápido en su funcionamiento y consume pocos recursos de memoria.

Así pues el parser de SAX:

- Detecta cuándo empieza y termina un elemento o el documento, o un conjunto de caracteres, etc. (genera eventos)
- Gestiona los espacios de nombres.
- Comprueba que el documento está bien formado.
- Las aplicaciones necesitan implementar manejadores de los eventos notificados.
- SAX lee secuencialmente de principio a fin, sin cargar todo el documento en memoria.

Cómo funciona SAX en Java:

1. Se crea una instancia de la clase SAXParserFactory para generar una instancia del parser.
2. El parser envuelve un objeto SAXParser. Cuando el método parser( ) es invocado se invoca uno de los métodos definidos por los interfaces ContentHandler, ErrorHandler, DTDHandler y EntityResolver.
3. Para realizar esto, se crea un objeto DefaultHandler que implementa una o las cuatro interfaces para que sólo haga falta redefinir los métodos necesarios. Normalmente se implementa ContentHandler que es donde están los métodos donde se recogen los datos del documento XML.
4. Se pasa al objeto SAXParser el documento XML a procesar y el objeto DefaultHandler como parámetros en el método parse( ).
5. El método parse ( ) instancia el objeto SAXReader que es el que está en comunicación con los manejadores definidos por el usuario.

1.5 Análisis de la información recibida: Parsing

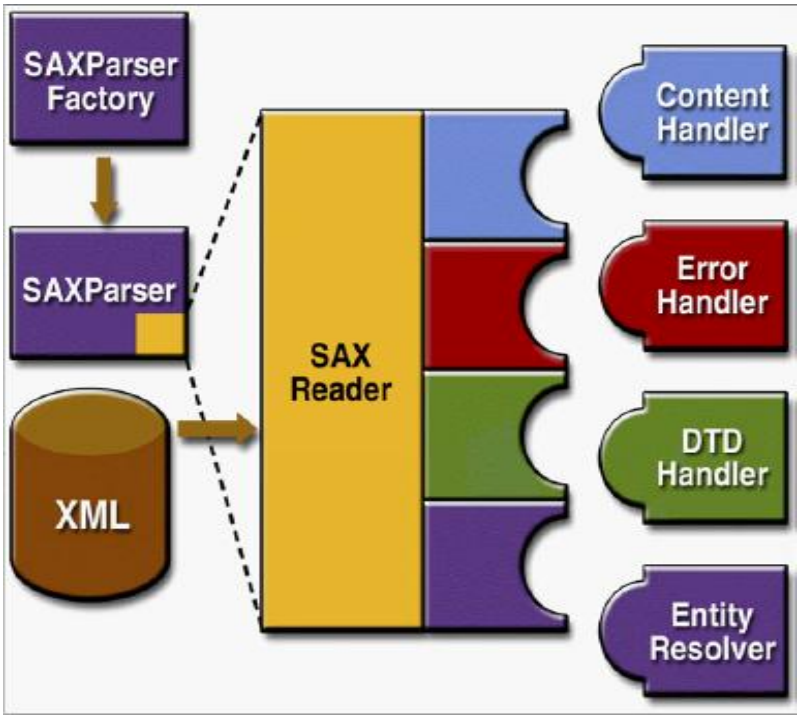

Figura 3. ***Esquema de funcionamiento de SAX***

Ejemplo gráfico de funcionamiento:

<? Xml version=”1.0”?> ⟶ startDocument( )

<libro> ⟶ startElement(libro)

<titulo> ⟶ startElement(titulo)

La sombra del águila ⟶ characters(“La sombra del águila”)

</titulo> ⟶ endElement(titulo)

<autor> ⟶ startElement(autor)

Arturo P. Reverte ⟶ characters(“Arturo P. Reverte”)

</autor> ⟶ endElement(autor)

</libro> ⟶ endElement(libro)

⟶ endDocument( )

Los métodos *startDocument( ), startElement( ), endElement( ), characters( ) y endDocument( )* forman parte de la interfaz *ContentHandler* que es la que tiene que implementar el usuario redefiniendo dichos métodos.

1.5.2 DOM

Son las siglas de *Document Object Model* es una interfaz de programación de aplicaciones (API) que proporciona un conjunto estándar de objetos para representar documentos HTML y XML, un modelo estándar sobre cómo pueden combinarse dichos objetos, y una interfaz estándar para acceder a ellos y manipularlos. La primera versión vio la luz en Octubre de 1998, la segunda en Noviembre de 2000 y la tercera en Abril de 2004 todas ellas a cargo de la W3C.

En DOM un documento es un conjunto de objetos jerárquicamente organizados en forma de árbol. Cada objeto es un nodo y un nodo puede ser un elemento, un atributo, una entidad, etc.

Por lo general la API de DOM necesita más recursos del sistema en tiempo de ejecución que SAX, sin embargo resulta más sencillo de utilizar y de entender.

Arquitectura del API de DOM

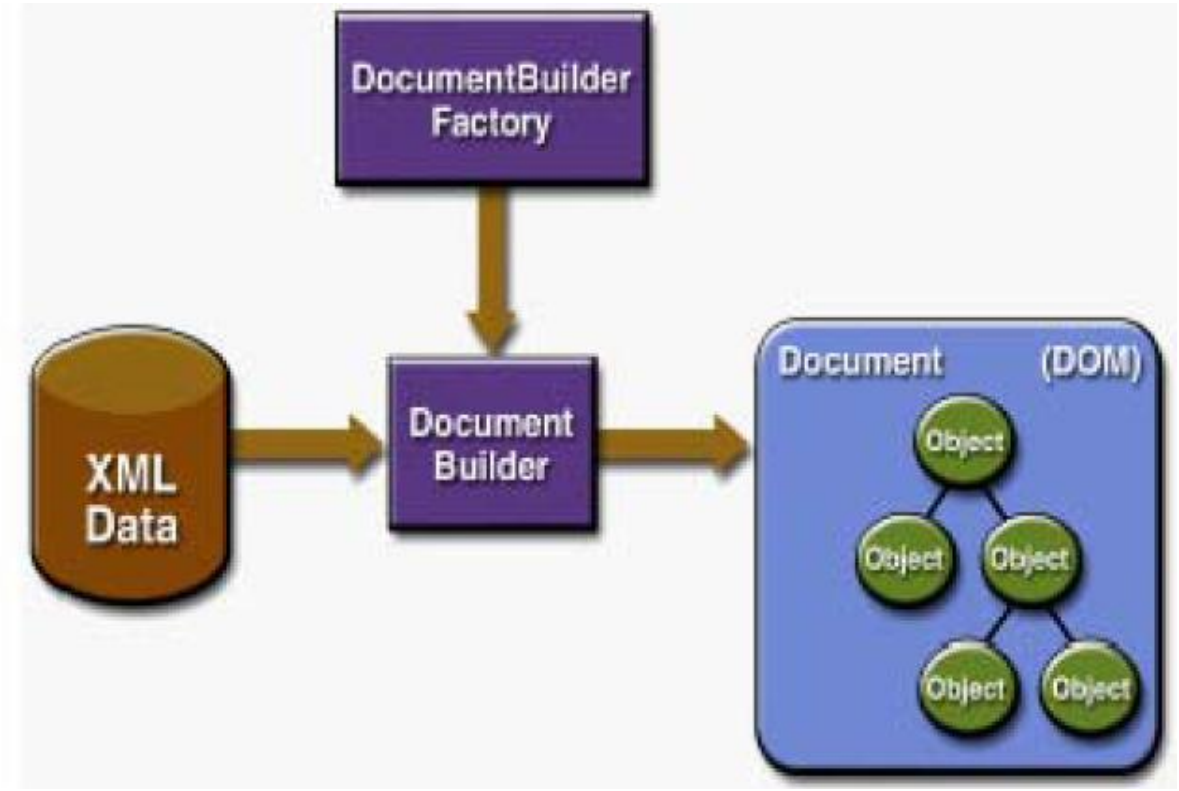

Figura 4. ***Esquema de la arquitectura del API de DOM.***

DOM permite construir nuevos documentos, modificar documentos ya existentes, navegar por la jerarquía del documento así como modificarla y gestionar los objetos del documento. Organiza un documento como un árbol de nodos, en forma de árbol y con una única raíz. El nodo es la unidad básica de procesamiento del documento y todo nodo excepto el nodo raíz tiene padre (parent), puede tener hijos (child) y puede tener hermanos (sibling).

DOM posee 11 tipos de nodo:

- Document: Es el nodo de todo el documento.
- Element: los elementos o etiquetas del documento incluido el raíz.
- Attribute: para los atributos de los elementos o etiquetas.
- Processing Instruction
- Comment
- Text: para el contenido de texto de cada elemento.
- Cdata section
- Document fragment
- Entity
- Entity reference
- Document type

Ejemplo de estructura de un documento

```
<xdoc>

      <title> The Adventures of Xman </title>

      <greeting> Hello world! </greeting>

      <greeting manner="cordial">
            Hello <emphasis> XML </emphasis> DOM
      </greeting>

   <!- Serie de aventuras-->

      <farewell>
            Goodbye cruel world
      </farewell>

</xdoc>
```

Figura 5. ***Ejemplo de documento XML.***

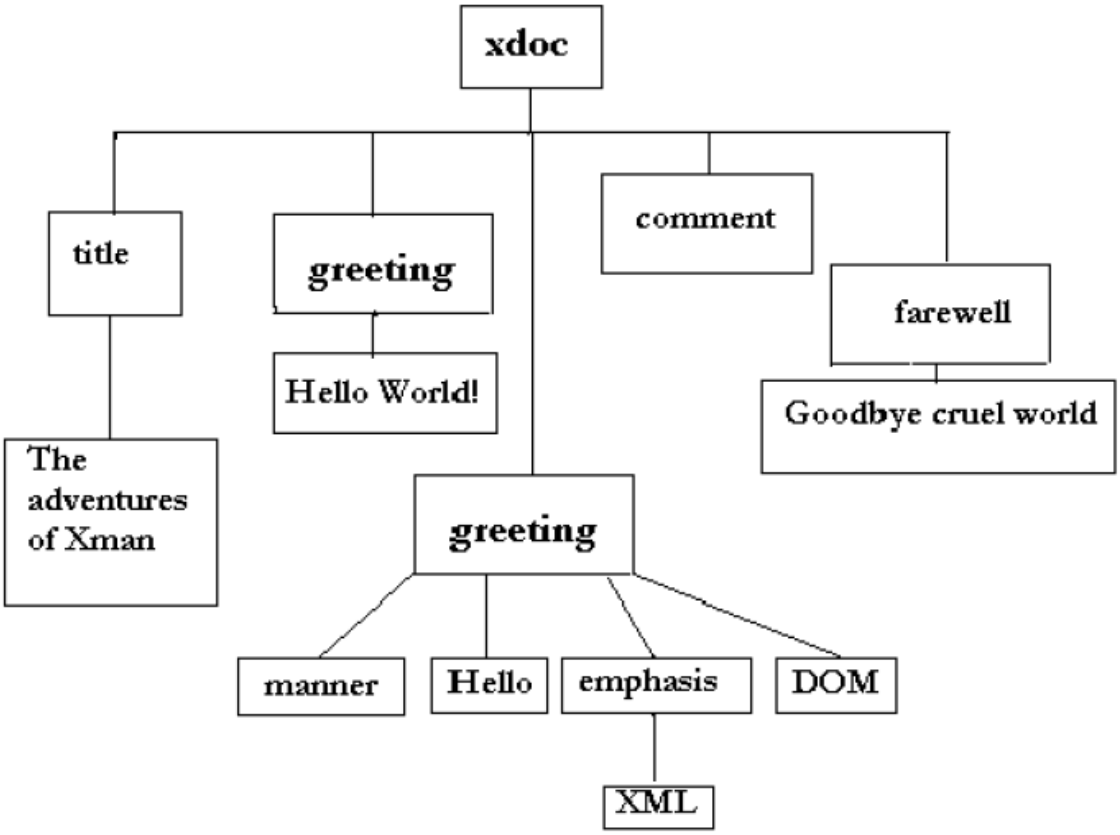

Figura 6. ***Estructura DOM del documento XML anterior.***

1.5 Análisis de la información recibida: Parsing

Paquetes DOM para Java

- Org.w3c.dom: Define los interfaces DOM para documentos XML.
- Java.xml.parsers: Define la clase factoría *java.xml.parsers.Document.BuilderFactory* y proporciona una instancia de la clase *DocumentBuilder* que se usa para instanciar un objeto *Document* que es conforme a la especificación DOM.

Parseo del documento XML:

A continuación se muestra un ejemplo en Java de parseo de un archivo xml; con este ejemplo se pretende ver los pasos básicos que se siguen cuando se parsea un documento xml cualquiera:

1.5 Análisis de la información recibida: Parsing

1º Se coge el fichero con el nombre del archivo xml:

```
File nombre = new File (“nombre.xml”);
```

2º Se crea la instancia de documentBuilderFactory:

```
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance( );
```

3º Instanciar DocumentBuilder con la instancia de DocumentBuilderFactory:

```
DocumentBuilder builder = factory.newDocumentBuilder( );
```

4º Se crea el documento que contendrá el árbol DOM con sus nodos, mediante la instancia de DocumentBuilder y el nombre del fichero xml:

```
Document doc = builder.parse(nombre);
```

Ejemplo completo de creación del árbol DOM:

```
try{
        System.out.println("\nEntra en manejadorDOM");
        File nombre = new File(“nombre.xml”);
        DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
        factory.setValidating(true);
        DocumentBuilder builder = factory.newDocumentBuilder();
        Document arbol = builder.parse(url);
} catch (ParserConfigurationException e) {
        e.printStackTrace();
} catch (SAXException e) {
        e.printStackTrace();
} catch (IOException e) {
        e.printStackTrace();
}
```

1.5 Análisis de la información recibida: Parsing

5º Se recorre el árbol DOM creado extrayendo la información necesaria.

Pasos para recorrer el árbol DOM:

1- **Localizar el elemento raíz con:**

```
Element raíz = doc.getDocumentElement( );
```

2- **Obtener los nodos hijos directos (Child Nodes) o bien obtener todos los nodos descendientes:**

Opción 1:

```
nodeList hijos = raíz.getChildNodes( );

Para recorrerlos:
Int = 0;
while(i < hijos.getLenght( )){
        Node son = hijos.item(i);
        if(son =! NULL){  … acciones…  }
        i++;
}
```

Opción 2:

```
for(node hijo = raíz.getFirstChild( ); hijo =! NULL; hijo = hijo.getNextSibling( )){
        …acciones para cada nodo…
}
```

3- **Tratamiento de los atributos.**

Se puede obtener nodos específicos buscándolos por determinados atributos que tengan mediante el API de DOM. Para un mayor nivel de detalle acerca de esta API consultar la documentación o javadoc en la web de Oracle[1].

[1] http://docs.oracle.com/javase/7/docs/api/org/w3c/dom/package-summary.html

1.6 Extracción de la información necesaria: Scraping.

1.6.1. Definición

El Web scraping es una técnica utilizada para recoger información de las páginas web. Las páginas Web son documentos escritos en HTML (Hypertext Markup Language), y más recientemente XHTML que está basado en XML. Los documentos web están representados por un árbol estructurado llamado DOM (Document Object Model) y el objetivo de HTML definir el modo en que el navegador web mostrará el contenido.

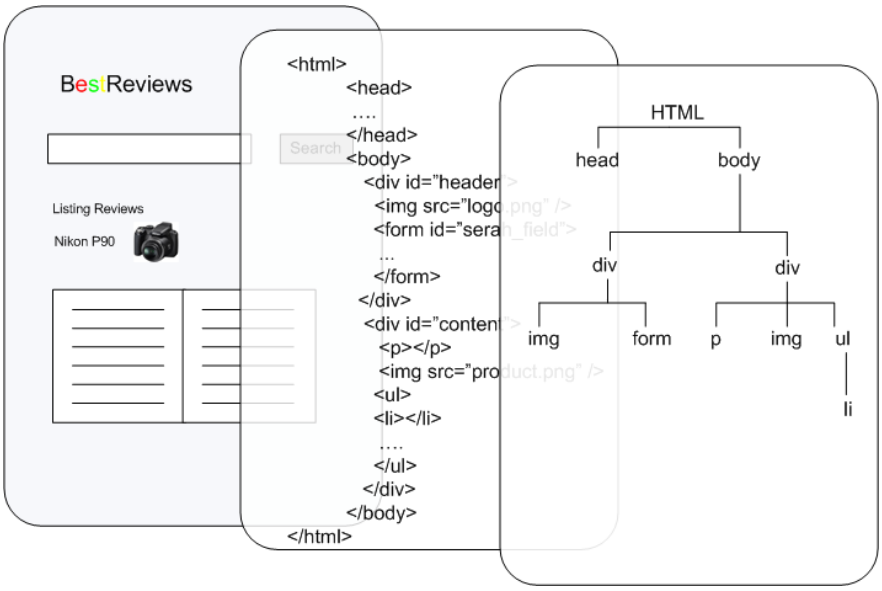

Figura 7. Representación web, HTML, DOM

La información recopilada se transforma en datos estructurados que pueden ser almacenados y analizados, por lo general en una base de datos u hoja de cálculo. Esta tecnología conduce una cantidad sustancial de los negocios, y la viabilidad de muchas empresas se basa en él. Sin embargo, la controversia puede surgir cuando las empresas comerciales utilizan web scraping para recoger grandes cantidades de datos de sitios web para su propio beneficio.

El web scraping se utiliza en múltiples propósitos, desde los más especializados hasta los más cotidianos. Entre ellos se destacan los siguientes:

- Seguridad y defensa: Los organismos de defensa cuentan con scripts que analizan el contenido de sitios web (blogs, páginas personales, dominios gratuitos, etc.) con el fin de detectar patrones de palabras que pudiesen incitar a actividades delictivas o ilegales. Una vez obtenidos los resultados, se puede escalar a un nuevo nivel de scraping , o se realizan análisis semi-manuales de los sitios identificados como sospechosos.

- Seguimiento de cambios de contenido (RSS): En oportunidades resulta particularmente útil mantenerse informado acerca de los cambios en el contenido de uno o varios sitios. Por ejemplo, para enterarse de nuevas noticias publicadas, seguir el cambio de precios en algún artículo, o verificar las ofertas laborales disponibles para un perfil profesional específico. En estos casos algunas herramientas de la web pueden ser configuradas para hacer el scraping periódicamente e informar en caso de cambios. Algunas aplicaciones de este tipo son google reader o feedity.com.

- Análisis de contenido: también es frecuente hacer scraping de un grupo de sitios con la finalidad de adelantar estudios de la información publicada. Esto puede ir desde un simple inventario de palabras y frases más utilizadas, hasta complejos análisis semánticos.

1.6.2. Estado del arte

Desde que el web scraping es utilizado para recopilar datos de otras fuentes online, es muy importante considerar la utilización de esta técnica, ya que el contenido que es copiado de un sitio web es generalmente difundido desde otro. Normalmente es suficiente con leer las políticas del sitio y comprobar que no se están violando estas o simplemente pedir permiso al propietario del sitio en cuestión para así evitar posibles problemas con la justicia.

Hay dos tipos de políticas de uso en línea: clickwrap y browsewrap. Las políticas clickwrap requieren que el usuario haga click en concordancia con los términos de uso mientras que las Browsewrap simplemente se enumeran en el sitio web, sin necesidad de una acción por parte del usuario. En consecuencia, si el usuario nunca vio estas políticas, no existiría contrato formalizado al no haber "acuerdo de voluntades".

El web scraping es principalmente utilizado por rastreadores y bots para extraer grandes cantidades de información. Las empresas que lo utilizan pueden estar sujetas a riesgos legales, pero bajo la ley actual, no está claro lo que pueden y no pueden hacer. La cuestión es saber que sucede tras romper los términos y políticas de los sitios web al recopilar su

información. La ley no es clara en cuanto a si esta actividad asciende a transgresión de los bienes muebles o incumplimiento de contrato. Además, si se recoge la información con derechos de autor (copyright), los operadores podrían ser responsables de la infracción.

Algunos propietarios de sitios web han salido victoriosos alegando estas infracciones, por lo que es un riesgo. Por lo general, lo único que se suele conseguir bajo el amparo de la ley, es la interrupción de la actividad de web scraping en el sitio web demandante, sin indemnización económica u otras responsabilidades penales.

1.6.3. Funcionamiento

Desde el punto de vista funcional, se podría comparar el web scraping con el método manual de copiar y pegar. La diferencia es que este trabajo se realiza de una manera organizada y automática, normalmente por una aplicación. De esta forma, cuando la aplicación recorre un árbol DOM, se está realizando la misma operación que haría un ser humano cuando interactúa con un sitio web. Esta puede seguir enlaces (mediante la emisión de Peticiones HTTP GET), y presentar los formularios (a través de HTTP POST), así como la navegación por multitud de sitios web.

Todo esto aporta una gran velocidad, ya que la velocidad de computación es infinitamente superior a la capacidad de un humano navegando. Aun así también puede ser un problema, ya que la gran velocidad realizando peticiones puede parecerse a un ataque DOS (Denial-of-Service) en el que se envían numerosas peticiones en un corto periodo de tiempo para desbordar al host. Por este motivo, hay que encontrar el punto donde obtener el mayor rendimiento posible pero donde los servidores no interpreten nuestro “diálogo” como un ataque.

Tras recuperar el documento DOM, la aplicación seguirá el path especificado para localizar la información deseada y extraerla con expresiones regulares de copiado en las que únicamente se seleccionarán los datos necesarios ignorando el resto del documento.

1.6 Extracción de la información necesaria: Scraping

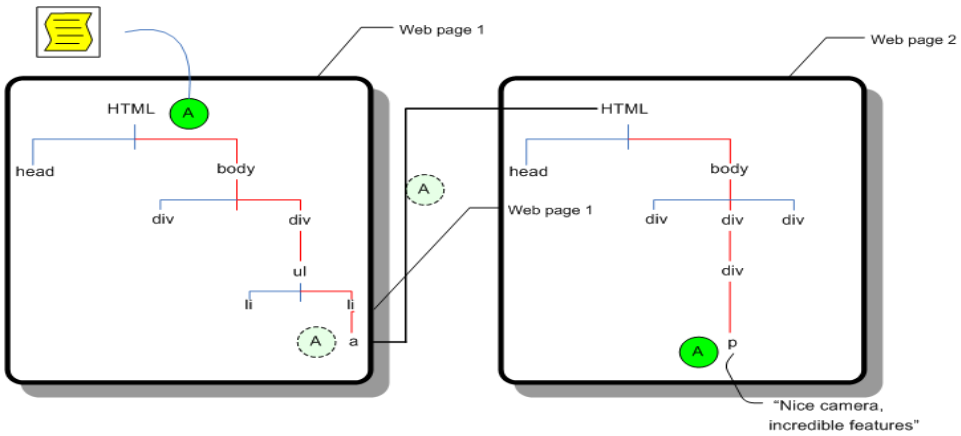

Figura 8. ***Ejemplo de Web Scraping.***

1.6.3.1 Dificultades

Como todas las técnicas, el web scraping tiene sus limitaciones y existen algunas tecnologías que limitan o hacen extremadamente complicada su utilización.

Ajax: causa problemas para los desarrolladores web, ya que permite a los nuevos contenidos ser descargados y presentados en una web sin la necesidad de una página. Esto nos rompe la relación uno a uno entre URL y contenido, ya que el contenido presentado depende en cierta parte de lo que el usuario hace después de cargar una página lo que causa que en general no se tenga acceso a este contenido.

JavaScript y cookies: Algunos sitios web manejan las cookies de forma muy complicada lo que hace realmente difícil emular su comportamiento. En algunas ocasiones además las páginas web contendrán JavaScript o imágenes que escribirán cookies o formularios. Estas cookies suelen estar condicionadas por el entorno, por ejemplo, la secuencia en la que se cargan estas imágenes. Este comportamiento variable e imprevisible hace que sea complicado utilizar las técnicas de scraping.

Contenido Flash: Flash siempre ha tenido polémica entre los desarrolladores web, ya que utiliza plug-ins embebidos en el navegador quedándose fuera del protocolo HTML. Esto hace complicado el uso de web scraping ya que los controles y contenido vienen descritos por estos plug-ins externos.

Aún así, existen técnicas avanzadas para conseguir emular el comportamiento de un navegador y así lograr obtener de forma automática el contenido de los sitios web que funcionan con alguna de las tecnologías que nos causan dificultades, se trata de las **Macros de navegador.**

Un Macro navegador es un plug-in que utiliza una secuencia de comandos para controlar las acciones de un navegador. Su principal ventaja es que puede aprovechar los plug-ins JavaScript y Flash o cualquier otra extensión disponible en el navegador.

Existe la posibilidad de utilizar iMacros (producido por iOpus) que es un macro de navegador disponible para Internet Explorer, Mozilla Firefox y iOpus. Es gratuito y se instala como cualquier otro plug-in.

1.6.4. Herramientas de Scraping

Hemos escogido diferentes tipologías de herramientas que existen para poder tener una visión más amplia de las diferentes opciones que nos podemos encontrar. Hay muchas en el mercado, pero hemos escogido las siguientes:

DEiXTo: Una potente herramienta de extracción de datos basada en W3C Document Object Model (DOM) que permite llegar un nivel muy bajo de detalle de extracción y a su vez disponiendo de una herramienta GUI para simplificar el trabajo. El motor principal se llama DEiXToBot (basado en Perl) y dispone de unas líneas de comandos para su ejecución múltiple y su programación.

Scraper Wiki: Plataforma web que permite escribir código en Python, Ruby o PHP y compartir el código con el resto de usuario para trabajar colaborativamente en la recolección de la información, en la generación de código y en la estructuración del mismo. No dispone de GUI y requiere altos conocimientos de programación.

Visual Web Ripper: Es la herramienta de scraping ideal para gran parte de usuarios. Gracias a su entorno gráfico y sus plantillas permite realizar un proyecto en poco tiempo y de manera ágil. A su vez se puede llegar a un nivel muy bajo de programación para soluciones más específicas o complejas. Permite exportar los datos a diferentes salidas, csv, Excel o Base de datos por ejemplo. A su vez se puede planificar y monitorizar con facilidad.

Mozenda: Es de las herramientas más usadas, es una aplicación web y se paga por páginas analizadas. Esto puede ser un problema para grandes volúmenes debido al coste que puede suponer.

DKS SocialSmart: Por su objetivo no es una herramienta de Scraping, pero utiliza metodologías de recolección de datos para lograrlo. Es un buen ejemplo de uso.

Solución de negocio para analizar grandes volúmenes de datos de las redes sociales mostrando los indicadores en un cuadro de mando final. Accede a cualquier información externa (Facebook, blogs, rss, Twitter, etc.).

Es una herramienta que además de recoger datos, los estandariza para poder cruzarlos analizándolos conjuntamente y aplicando un enriquecimiento de los mismos. Aporta un alto nivel de configuración a la vez que nos da un amplio trabajo en la captación de datos de redes sociales.

1.6.5. Utilidades y aplicaciones comerciales.

Actualmente el web scraping es utilizado por gran cantidad de aplicaciones y sitios web muy populares como por ejemplo la gran mayoría de motores de búsqueda.

Los motores de búsqueda son sistemas que buscan las páginas publicadas más relevantes en internet dada una palabra o una frase. Se han convertido en una pieza clave en internet debido al fuerte crecimiento que ha tenido en tan poco tiempo y a la gran cantidad de páginas web que hay publicadas. Es necesario que toda esta información sea ordenada facilitando al usuario su consulta y manejo. Existen varios motores de búsqueda utilizados por los usuarios y en el siguiente grafico se pueden ver algunos de ellos.

1.6 Extracción de la información necesaria: Scraping

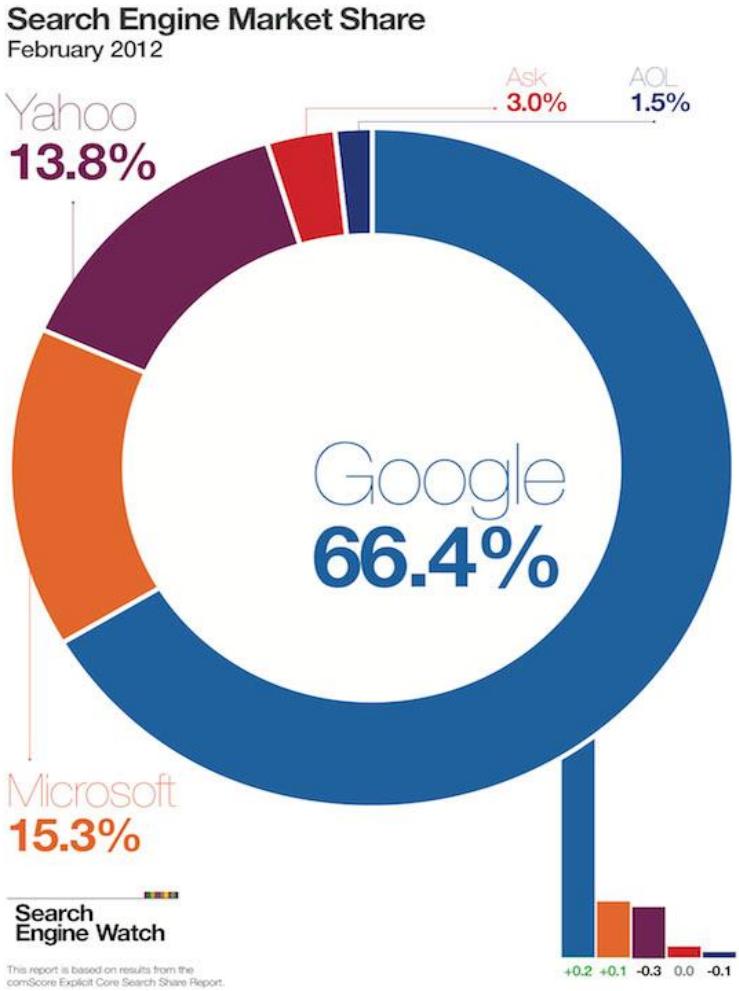

Figura 9. ***Gráfico de motores de búsqueda.***

Todos los motores de búsqueda se componen principalmente de tres partes:

- **Robots de búsqueda** para el análisis de páginas web.
- **Índices** para indexar y almacenar la información obtenida en una gran base de datos.
- **Algoritmos de búsqueda** para entregar al usuario final los resultados de la búsqueda que se asocien a su demanda.

Es en la parte de los robots de búsqueda donde entra en acción el web scraping ya que se necesita una herramienta que recorra automáticamente la red en busca de páginas web. A continuación veremos algunos tipos y sus diferencias:

1.6 Extracción de la información necesaria: Scraping

Web bots: El proyecto Web Bots se originó a finales de los años 90 de la mano de Clif High y George Ure, como un sistema de rastreo de Internet para predecir tendencias en los mercados financieros. El algoritmo que utiliza es sencillo. Rastrea la red en busca de palabras clave que mantiene en un diccionario, y dinámicamente analiza cuales de esas palabras adquieren más peso en un momento dado.

Llevando su invento al extremo, Ure y High proclamaron que los acontecimientos que ocurren en el mundo real pueden ser anticipados mediante Web Bots. Se le atribuye a esta herramienta el haber predicho los ataques del 11 de Septiembre, el desastre del Columbia, el terremoto del Índico del 2004, el huracán Katerina, y alguno más. Lo cierto es que también ha fallado en muchos otros, como el colapso del dólar en 2011, un terremoto en Vancouver en 2008 o la tercera guerra mundial en 2010. Estos últimos fallos se mencionan menos. El sentido común nos dice que un algoritmo de búsqueda que analiza el contenido de páginas web publicadas en todo el mundo, tiene una cierta capacidad para identificar tendencias, pero en absoluto puede prever el futuro, en la medida en que nadie de los que publicamos esas páginas podemos preverlo.

El término web bot se continúa utilizando y se pueden definir como scripts encargados de descargar las páginas web de internet para, una vez descargadas, parsear y manipular la información que estas contienen. Aunque las páginas web son las principales fuentes de información de estos scripts, técnicas más avanzadas permiten recopilar información de emails y servidores FTP. Cuando se dispone de la información, estos web bots pueden ejecutarse por línea de comandos o en el propio navegador.

Spiders: también conocidos como web spiders, crawlers (rastreadores) y web walkers, son webbots especializados que, a diferencia de web bots tradicionales con objetivos bien definidos, se encargan de descargar varias páginas web a través de múltiples sitios web. Como los spiders hacen su camino a través de Internet, es difícil anticipar por donde va a ir o lo que van a encontrar, ya que sólo tiene que seguir y enlazar páginas previamente descargados. Su imprevisibilidad hace divertida su implementación, ya que actúan casi de una manera independiente.

Los mejores y más conocidos spiders son los utilizados por los motores de búsqueda (Google, Yahoo! y Bing) para identificar el contenido en línea. Pero mientras spiders son sinónimo de motores de búsqueda para muchas personas, el potencial y la utilidad de estos es mucho mayor. Se puede escribir un spider que hace algo que cualquier otro web bot hace, con la ventaja de dirigirse a la totalidad de Internet.

Esto crea una base para los desarrolladores que diseñan spiders especializados. Aquí están algunas ideas para posibles proyectos utilizando spiders:

- Comparación y seguimiento periódico de precios.
- Seguimiento de los sitios web de competidores.
- Envío de mensajes masivos en redes sociales, por ejemplo, a todos los estudiantes de una universidad o a todos los residentes en una ciudad.
- Buscadores de empleo.
- Testeo y validación de los links de una página web para evitar enlaces caídos.

En este diagrama se muestra un resumen del funcionamiento de un spider simple.

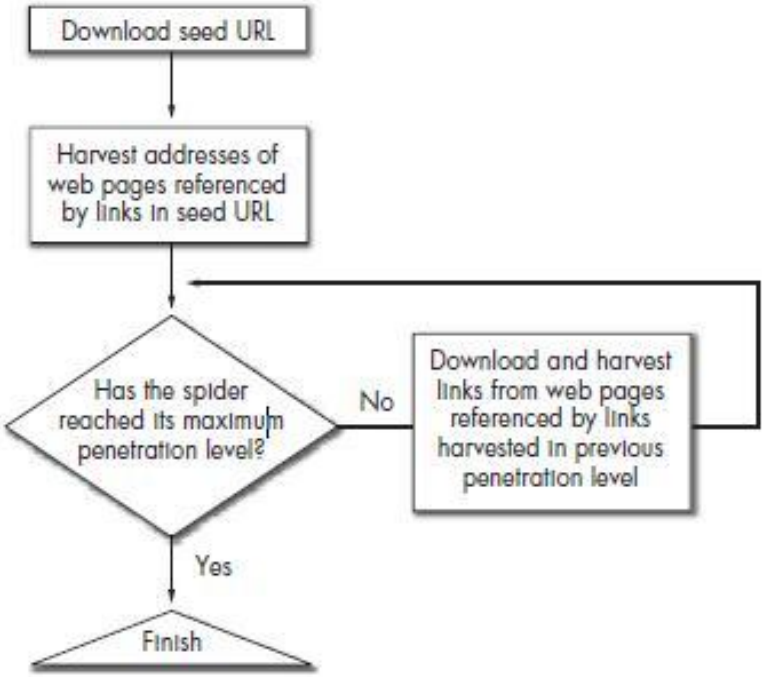

Figura 10. ***Funcionamiento de un spider.***

Procurement bot: un procurement bot es cualquier agente web inteligente que automáticamente hace compras en línea en nombre de un usuario. Esta automatización mejora la contratación en línea manual, ya que no sólo automatizan el proceso de compra en línea, sino que también detectará de forma autónoma eventos que indican el mejor momento para comprar. Normalmente hacen compras automatizadas en base a la disponibilidad de mercancía o de reducción de precios. Además se pueden programar de acuerdo a múltiples criterios, como por ejemplo, la compra automática ante pequeñas cantidades de stock. La ventaja del uso de estos bots en un negocio en que son capaces de identificar oportunidades disponibles por un corto período de tiempo, o con una búsqueda

manual en línea lo que puede resultar tedioso, lento y propenso a errores humanos. Esto puede suponer un gran ahorro en una empresa.

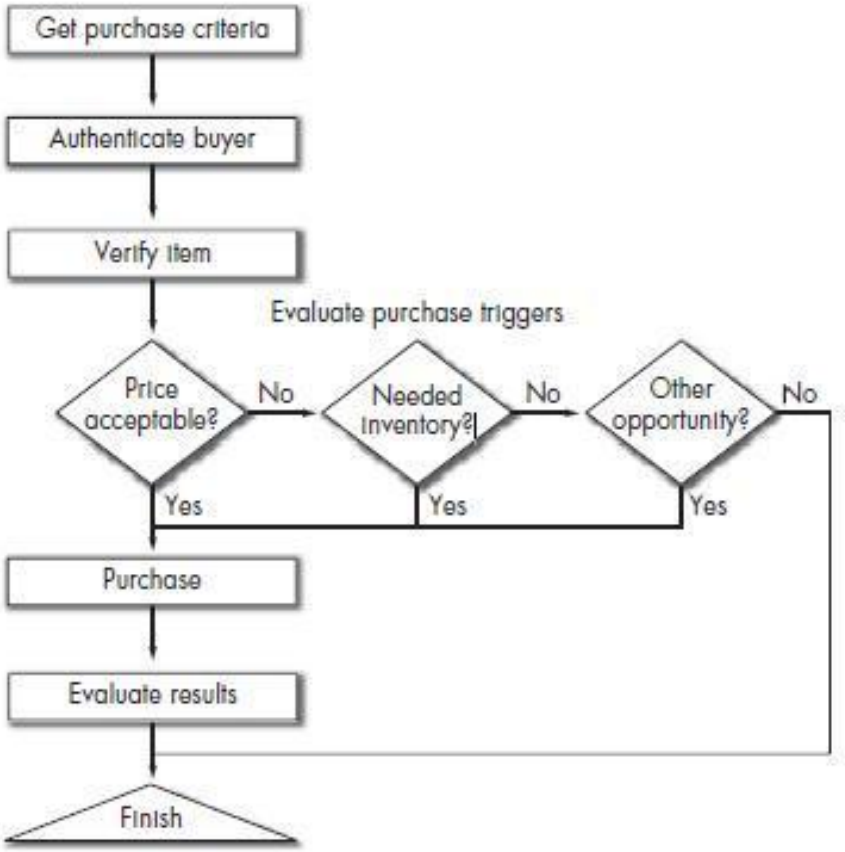

Figura 11. Funcionamiento de un bot de adquisición

1.7 Android

El sistema operativo de Google tiene 6-7 años y en este tiempo lo hemos visto evolucionar de una manera realmente impresionante que veremos en detalle a continuación.

Android se inició oficialmente el día 22 de octubre de 2008 en Estados Unidos, cuando fue lanzado el primer teléfono con Android a bordo, el G1 de T-Mobile.

La ventana de notificación desplegable fue la gran novedad que introdujo Android, apostó desde el principio por un sistema de notificación con el que tener toda la información a la vista y el tiempo le ha dado la razón ya que ha sido copiado por iOS en su última versión.

También estaba caracterizado por:

Widgets en la pantalla de inicio. Gran variedad de widgets configurables para anclar al inicio.

Integración de Gmail. Android 1.0 supuso la mejor experiencia de correo en el móvil que había en el mercado gracias al apoyo de Gmail a POP e IMAP.

Android Market. El primer Android Market salió sin apenas aplicaciones y con un diseño de una sola fila ubicada en la parte superior de la pantalla de inicio de la aplicación.

1.7.1 Recorrido por las diferentes versiones

Android 1.1

En febrero de 2009 llegó la primera actualización para Android, unos tres meses después del lanzamiento del G1. La versión 1.1 fue dedicada básicamente a reparar errores y a implementar las actualizaciones “over the air” que hasta ese momento ninguna plataforma estaba haciendo.

Android 1.5 Cupcake

Android 1.5 es más conocido por su nombre en clave, **Cupcake**, fue la primera versión en utilizar nombre de postres.

En esta versión comenzamos a ver algunos cambios en la interfaz de usuario, por poco que se puedan apreciar como son los cambios de en la barra del buscador y en la barra del menú, también cambió el logo del navegador. Las primeras versiones de Android no

contaban con un teclado virtual**,** ya que el G1 disponía de un teclado físico, en la 1.5 se introdujo el teclado virtual coincidiendo con la salida del primer Android con pantalla táctil y sin teclado físico, el HTC Magic.

Desde el principio Google introdujo la posibilidad de que desarrolladores terceros pudieran crear sus propios teclados virtuales, algo que a día de hoy sigue diferenciándolo de iOS y Windows Phone.

Widgets: Permitió que widgets de terceros estuvieran disponibles para los usuarios.

Mejoras en el portapapeles: La plataforma no ofrecía desde el principio la posibilidad de copiar y pegar en las ventanas del navegador o Gmail. Se integró por primera vez en esta versión y fue terminada en posteriores versiones.

Captura y reproducción de vídeo: La primera versión de Android no ofrecía la posibilidad de grabar vídeo ni de reproducción, si no que se integró con la 1.5. Los fabricantes comenzaron a crear sus propias interfaces para la cámara añadiendo soporte para escenas adicionales, modos, opciones y la posibilidad del enfoque presionando sobre la pantalla táctil.

Android 1.6 Donut

Con la llegada de Donut vino el soporte para redes CDMA haciendo que Android llegara a Estados Unidos y Asia**.** Pero tal vez la mejora más significativa fue la posibilidad de correr el sistema operativo en múltiples resoluciones de pantalla y relaciones de aspecto, a raíz de esta actualización es cuando podemos disfrutar hoy en día de pantallas con resolución QVGA, HVGA, WVGA, FWVGA, QHD y 720p.

Donut también introdujo la búsqueda rápida, generalmente conocida en el mundo del móvil como búsqueda universal. Antes de Donut la búsqueda se limitaba a Internet pero con las mejoras introducidas en la versión 1.6 se podría buscar además contenido propio del teléfono (contactos, aplicaciones, etcétera). Todo ello desde el mismo widget de búsqueda.

Android 2.0/2.1 Eclair

Un año después del estreno del G1, a principios de noviembre de 2009 fue lanzado Android 2.0 Eclair. Fue ofrecido en exclusiva con Verizon y el Motorola Droid, un teléfono que marcó un antes y después para Android y con el que Motorola volvió a ser la gran marca que fue.

Apoyo de varias cuentas: Por primera vez se podrían añadir varias cuentas en el mismo dispositivo con acceso al correo electrónico y a los contactos de cada una, además también

se introdujo soporte para cuentas de Exchange. También se abrió la puerta de las sincronizaciones automáticas para los contactos gracias a la información compartida entre los tipos de cuenta; Facebook fue la primera en integrar esta funcionalidad.

Google Mapas Navegación: Google Maps Navigation fue publicado junto con la versión 2.0 y fue un paso adelante parar integrar un sistema de navegación de automóviles en el móvil con vistas en 3D, guía de voz e información de tráfico de forma completamente gratuita. Hoy en día sigue siendo una de las mejores opciones para tu teléfono.

Contacto rápido: Eclair agregó una barra de contacto rápido, una barra de herramientas desplegable que se utiliza para realizar múltiples funciones de manera rápida (mandar email, mensajes, llamar, etc.)

Nuevas mejoras en el teclado: El Droid también fue lanzado con teclado QWERTY pero Google aprovechó para mejorar un poco más el teclado virtual.

Renovación del navegador: Google añadió soporte HTML5, incluyendo vídeo pero solo en modo pantalla completa. Aunque seguía sin estar disponible la posibilidad de multitouch pero se agrega la posibilidad de zoom tocando dos veces.

Después del Droid/Milestone prácticamente la mayoría de teléfonos lanzados llegaron con Android 2.1, una corrección de errores y que Google no renombró dejándola con el nombre de Eclair.

Fondos de pantalla animados: Por primera vez aparecieron los fondos de pantalla animados en Android, en lugar de usar una imagen estática.

De voz a texto: Los usuarios podían dictar a su teléfono y éste lo transcribía a texto (TTS). En Android 2.1 se reemplazó la tecla de la coma en el teclado en pantalla por un micrófono para utilizar dicho servicio.

Nueva pantalla de bloqueo: Android 2.0 introdujo una nueva pantalla de bloqueo. Para desbloquear el teléfono o silenciarlo tan solo teníamos que deslizar el dedo por la misma en una dirección u otra. Android 2.1 cambió la pantalla de bloqueo y la hizo más al estilo iOS.

Y entonces llegó el Nexus One, el primer teléfono de Google fabricado por HTC y que estaba destinado a mostrar el más puro estilo Android 2.1 sin ninguna modificación. Fue uno de los primeros smartphones con procesador de 1GHz Qualcomm Snapdragon del mercado, además contaba con una pantalla AMOLED con resolución WVGA. Hoy en día se sigue utilizando y es aún un buen teléfono.

1.7 Android

Android 2.2 Froyo

Lanzado a mediados de 2010 trajo una gran cantidad de cambios. La pantalla de inicio fue rediseñada, se ampliaron los 3 paneles existentes desde el inicio a 5 con un nuevo grupo de accesos directos dedicados y se agregaron unos puntos para saber en cada momento en la pantalla donde nos encontrábamos. El Nexus One fue el primer teléfono en actualizarse a Android 2.2.

Froyo también introducía una galería completamente rediseñada con imágenes en 3D que aparecen al inclinar el teléfono. Además se introdujo soporte para hotspot móvil (compartir la conexión 3G), algo que muchas compañías decidieron desactivar o activarla con la opción de pagar un coste extra. Se mejoró también el soporte para copiar y pegar en Gmail.

En esta versión se agregó la posibilidad de poner una contraseña o PIN en la pantalla de bloqueo para los usuarios que no les gustaba el patrón de desbloqueo.

Android 2.3 Gingerbread

Un año y medio después del lanzamiento de Froyo y el Nexus One, Google volvió con un nuevo móvil de marca propia pero esta vez en colaboración con Samsung, el **Nexus S** y aprovechó para lanzar la nueva versión del sistema operativo, Android 2.3 Gingerbread. Con el Nexus S llegó la pantalla curvada y el fin del trackball. Gingerbread fue una actualización menor en muchos sentidos pero trajo algunos cambios importantes en la interfaz de usuario.

Mejor control en copiar y pegar: Se añade en esta versión la posibilidad de seleccionar el texto que queremos copiar y pegar. Anteriormente solo se podía copiar el contenido de las cajas completas. Se agregan unas pestañas para seleccionar el texto que queremos copiar.

Teclado mejorado: Nuevamente Google pone su empeño en mejorar el teclado, cambios en el diseño y de coloración además del soporte multitouch.

Maximización de la batería y herramientas de gestión de desarrollo: Google pecó de ser demasiado permisivo con la multitarea y esto hacía mella en la duración de la batería. Se instaló una herramienta para la gestión de la batería que informa de qué aplicaciones están consumiendo la batería.

Soporte para cámara frontal: Gingerbread fue la primera versión en integrar soporte para varias cámaras, aunque la opción de video chat en Google Talk no llegaría hasta mediados de 2010. El Nexus S ya dispondría de cámara frontal, aunque en un principio solo servía para tomar fotos con ella.

Otras características: Apoyo a la tecnología NFC integrada en una antena incrustada en la tapa de la batería. En un principio es usada como si de un código QR se tratara para escanear sitios en Google Places pero más tarde Google presenta Google Wallet, una aplicación de pago utilizando la tecnología NFC integrada en el Nexus S.

Juegos: La nueva versión dio más libertad a los desarrolladores para poder escribir código más rápido y desarrollar juegos con gráficos en 3D que hasta entonces no disponía Android. Google estaba perdiendo la batalla de los juegos con iOS y tenía que reaccionar.

Android 3.0 Honeycomb

La versión de Android para tablets, que presentó de la mano de Motorola junto con el Xoom. Cambio de color, del verde típico de Android al azul que se utilizó para la batería, el widget del reloj, indicadores de señal y algunas otras características de la interfaz.

El final de los botones físicos: Se integra una barra en la parte inferior de la pantalla con una serie de botones virtuales que hacen que no se necesiten botones dedicados. Es el fin de los botones físicos, tendencia que continuará con Android 4.0 ICS.

Multitarea mejorada: La multitarea ha sido mejorada gracias al diseñador Matías Duarte, ex diseñador de webOS contratado por Google. Aparece un nuevo botón virtual de aplicaciones recientes en la parte inferior de la pantalla en la que podemos ver una lista de las últimas aplicaciones utilizadas con capturas de pantalla de las mismas.

Una nueva barra para las aplicaciones: se introduce el concepto de barra de acción, una barra permanente situada en la parte superior de cada aplicación que los desarrolladores pueden utilizar para mostrar las opciones de acceso frecuente, menús, etc. Es como una barra de estado dedicada a cada aplicación.

Android 3.1 y 3.2 fueron versiones de mantenimiento, prueba de ello es que Google no las renombró y continuaron llamándolas Honeycomb. Aunque algunas mejoras introducidas en estas actualizaciones se han ido implementando en la mayoría de tablets con Android 3.0 del mercado, como la posibilidad de modificar el tamaño de los widgets al presionar sobre ellos.

Android 4.0: “Ice Cream Sandwich”

Llegamos a la última versión del sistema operativo de Google, Android 4.0 Ice Cream Sandwich. Ha sido lanzada junto con el Galaxy Nexus, el nuevo smartphone Google

fabricado por Samsung. Toma prestadas muchas características de Honeycomb como los botones virtuales o la transición de tonos verdes a azules, la multitarea con una lista desplegable de miniaturas y las barras de acción dentro de las aplicaciones.

Letra: Por primera vez se modifica el tipo de letra. Droid fue la fuente utilizada desde la versión 1.0 y ahora se modifica por Roboto, una fuente que ha sido diseñada para aprovechar la mayor resolución de las pantallas de hoy en día.

Como no, el teclado virtual también ha sido modificado: esta vez incluye un sistema de corrección mucho más avanzado que subraya en color rojo las palabras mal escritas e incorpora también un diccionario.

La pantalla de notificaciones también ha recibido una pequeña actualización con las notificaciones individuales extraíbles que permiten deslizar cualquier notificación fuera de la pantalla y acceder a ella. Tenemos más mejoras en la pantalla de inicio: la pantalla adopta muchos cambios de los que se introdujeron en Honeycomb pero añade además algunas características nuevas como la posibilidad de crear carpetas con solo arrastrar un icono a otro. Además la pantalla principal recibe una bandeja de favoritos que puede ser configurada por el usuario.

NFC: El soporte de la tecnología NFC ya había sido promocionado fuertemente con el lanzamiento de Gingerbread y el Nexus S aunque no había prácticamente ninguna aplicación. En Ice Cream Sandwich se busca potenciar el uso de NFC con una nueva característica para transferencia de datos entre dos teléfonos con solo tocarlos.

Desbloqueo facial: Además del bloqueo con contraseña y con patrón de desbloqueo se ha agregado la opción del desbloqueo facial.

Análisis de los datos. Se añade un gestor para el uso de los datos en el que se informa de las aplicaciones que consumen más datos, se puede ver el uso total desglosado en un periodo de tiempo configurable por el usuario.

Nuevo calendario y aplicaciones de correo electrónico. El correo electrónico de Gmail ha sido revisado en Ice Cream Sandwich con nuevos diseños y con la incorporación de la barra de acción. El calendario está unificado, se pueden ver todos los eventos de todas las cuentas en el mismo calendario.

1.7 Android

Jelly Bean: Android 4.1:

Lanzado el 9 de Julio de 2012, fue una actualización centrada principalmente en mejorar la funcionalidad y el rendimiento de la interfaz de usuario.

Las principales mejoras y funcionalidades introducidas respecto a la versión anterior fueron:

- Aumentar automáticamente la velocidad de trabajo del procesador al tocar la pantalla, haciendo que el sistema sea mucho más rápido y fluido; sin embargo esto conlleva un mayor consumo de batería.
- Mejora del sistema de notificaciones permitiendo extender información gestualmente o interactuar con ciertas aplicaciones desde el propio sistema de notificaciones.
- Mejora del sistema de widgets.
- Mejora del teclado con una predicción de texto más inteligente.
- Mayor aprovechamiento de los recursos hardware.

EL primer dispositivo en incorporar Jelly Bean 4.1 fue el Nexus 7, la tablet de Google lanzada en Junio de 2012.

Jelly Bean: Android 4.2:

Esta actualización de Jelly Bean no incorporó grandes mejoras respecto a la anterior versión 4.1. Las principales mejoras y nuevas funcionalidades son:

- Soporte para diferentes usuarios, permitiendo gestionar diferentes cuentas de usuario. Esta funcionalidad sólo está disponible para tablets.
- Mejoras en la aplicación de la cámara de fotos.
- Mejora del teclado, permitiendo escribir las palabras deslizando el dedo, sin levantarlo, por las letras.

Jelly Bean: Android 4.3:

Versión lanzada el 24 de Julio de 2013; introduce mejoras respecto a su anterior versión:

- Mejora en las cuentas multi-usuarios, permitiendo introducir restricciones para determinadas cuentas, como las aplicaciones a las que se accede, el contenido que puede buscar un usuario y los contenidos de una propia aplicación.

- Mejora del teclado de marcación permitiendo la opción de autocompletar la marcación de un número de teléfono.
- Mejoras en la aplicación de la cámara de fotos, con una nueva interfaz de usuario y permitiendo realizar fotos con los botones de volumen además de incorporar opción de temporizador.
- Nuevas opciones de desarrollo. Con esta versión se puede revocar el acceso a la depuración USB de todos los ordenadores autorizados.
- Gestor de permisos oculto. Al estar oculto, para poder usarlo hay que hacerlo mediante una aplicación externa ya que esta versión de Android no incorpora una.
- Mejora de los gráficos al soportar OpenGL ES 3.0 la API más reciente de esta librería gráfica.
- Soporte para Bluetooth Smart (Bluetooth 4.0); esta versión de Bluetooth se caracteriza por un bajo consumo de batería permitiendo que las aplicaciones detecten y se comuniquen con dispositivos compatibles con Bluetooth Smart. Además en la comunicación entre aplicaciones con esta tecnología, permite el envío de metadatos entre ellas, útiles para aplicaciones como el reproductor de música.
- Mejoras del sistema y de la seguridad. Mejora de las políticas de seguridad, controles de acceso con el fin de evitar que una aplicación acceda a partes del sistema de las que no tiene permiso y una mayor dificultad para dar permisos de administrador o súper-usuario. Además, el sistema avisará mediante notificaciones qué aplicaciones se están ejecutando en primer plano.

KitKat: Android 4.4:

KitKat es la última versión disponible en la actualidad y cuenta con las siguientes mejoras:

- Mejora del rendimiento, optimizándose para terminales con 512Mb de RAM.
- Mejora de la seguridad en las transacciones NFC con Host Card Emulation **(**HCE**).**
- Integración de almacenamiento en la nube (Google Drive, DropBox, Box).
- Hangouts como opción para la gestión de SMS.
- Permite grabación de pantalla con ScreenCast.
- Soporta nuevos perfiles de bluetooth sobre GATT que permite a las aplicaciones un enlace de baja latencia con periféricos de baja potencia, como ratones y teclados.

- Se implementan mejoras estéticas como la pantalla completa, la interfaz de usuario traslucida y mejoras en el WebView con soporte para HTML5, CSS3 y JavaScript (V8).

1.7.2 ¿Qué se necesita para programar?

Si cualquier usuario desea programar aplicaciones para Android deberá:

- El desarrollador debe tener nociones básicas de lenguaje Java, ya que estas aplicaciones se desarrollan en dicho lenguaje.

- El desarrollador deberá tener en su ordenador instalado lo siguiente:
- Eclipse -> entorno de desarrollo integrado de código abierto usado, entre otras cosas, como plataforma para desarrollar el IDE de Java.
- Plugin ADT (Andoid Developer Tools) -> plug-in para Eclipse que proporciona un conjunto de herramientas que se integran con el IDE de Eclipse. Se le ofrece acceso a muchas características que ayudan a desarrollar aplicaciones para Android con rapidez.
- Android SDK -> conjunto de herramientas y programas de desarrollo que permite al programador crear aplicaciones para un determinado paquete de software, estructura de software, plataforma de hardware, sistema de computadora, consulta de videojuego, sistema operativo o similar.

1.7.3 Elementos básicos

Los elementos que conforman la estructura para las aplicaciones Android son:

- **Activity o Actividad:** Conforma los eventos y acciones que se realizan para una vista o pantalla de la aplicación. En dicha vista se incluirán los elementos de interfaz de usuario (botones, etc.). A la interfaz de usuario de una vista o pantalla se le denomina layout y está implementado en lenguaje XML.
- **Service o Servicio:** Está diseñado para permanecer ejecutándose en segundo plano, en caso de que fuera necesario, independientemente de otros Activity. Por ejemplo para comprobar un feed de RSS, establecer operaciones programadas en el tiempo (cron jobs) etc.

- **Content Provider:** Permite crear modelos de datos con información, que puedan ser accesibles por múltiples aplicaciones.
- **Intent:** Es un mensaje de sistema que, dentro del dispositivo, permiten enviar notificaciones a las aplicaciones; también se usa para realizar llamadas entre diferentes Activity de la misma aplicación:
    - ✓ Cambios en el HW (tarjeta SD insertada)
    - ✓ Información entrante (mensajes SMS)
    - ✓ Eventos desde las aplicaciones (una aplicación ha sido arrancada desde el menú principal)

Al crear una aplicación nos encontraremos con varios ficheros que también formarán parte de los elementos básicos de la aplicación:

- **AndroidManifest.xml**: XML que describe los componentes (*activities*, *services*...) que conforman la aplicación y los permisos que esta tiene.
- **bin/**: El directorio que guarda la aplicación ya compilada.
- **libs/**: El directorio donde se guardarán los JARs de terceros que pudiera utilizar nuestra app.
- **res/**: Se almacenan los recursos: iconos, imágenes, layouts de interfaz de usuario...
- **src/**: Donde nos encontraremos con el código fuente de la App
- **gen/**: La herramientas del SDK de Android generan archivos fuente a partir de los XML GUI por ejemplo R.java
- **assets/**: Almacena archivos estáticos que queramos empaquetar con la app.
- **proguard.cfg/**: Archivo empelado para la integración con Proguard.

El Directorio res/ :

- **res/drawable/**: Para imágenes (PNGs, JPEG...).
- **res/layout/**: Guarda los XML que especifican los layouts de interfaz de usuario.

- **res/63xce/**: Para los XML empleados para definir los menús.

- **res/raw/**: Para archivos de propósito general que no estén implementados en XML: archivos, Excel, audios, etc.

- **res/values/**: Definen los contenidos de las etiquetas de texto. De este modo se centralizan en este archivo.

1.7.4 Ciclo de vida de una aplicación

El ciclo de vida de una aplicación es el que se describe la siguiente imagen:

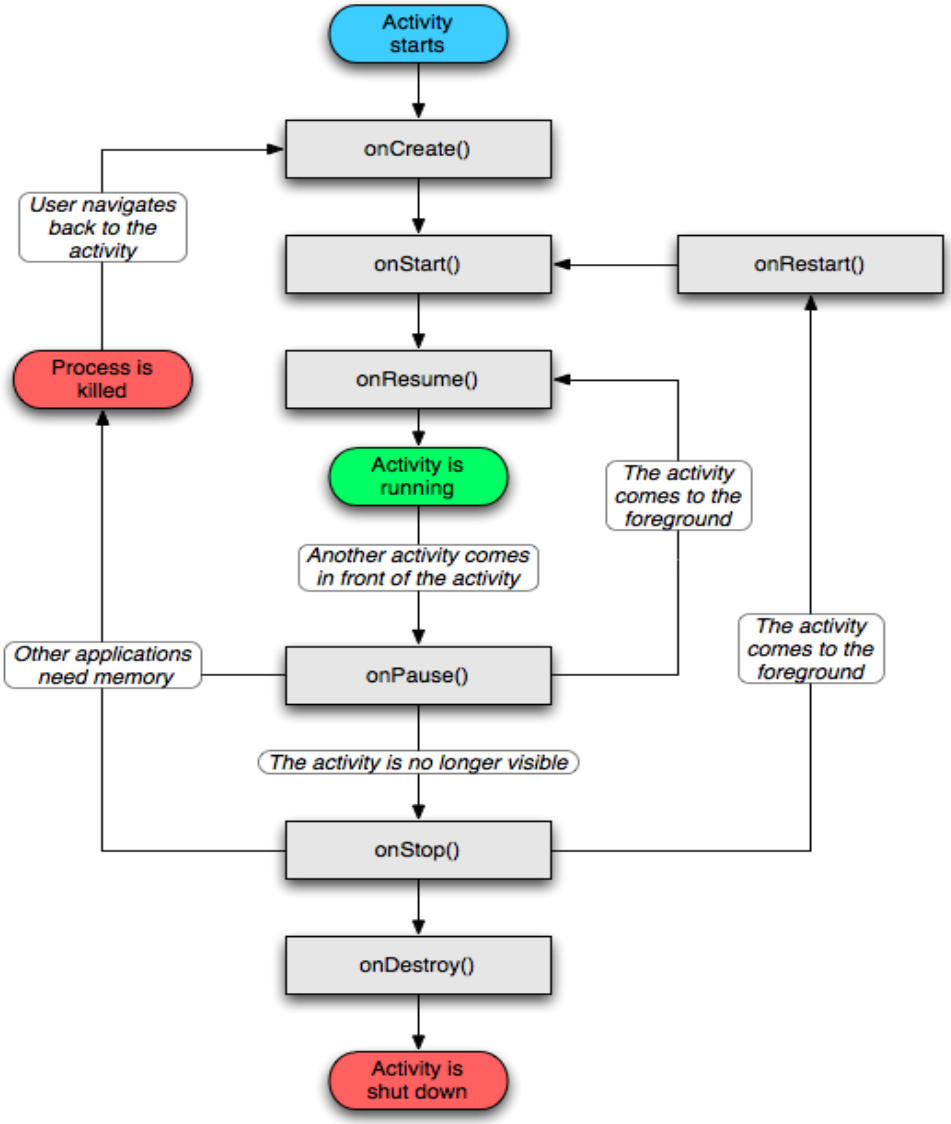

Figura 12. Ciclo de vida de aplicación Andoid

1.7 Android

Android puede en cualquier momento pausar, parar, destruir nuestra aplicación según las necesidades del momento o el uso de las aplicaciones por parte del usuario; será el programador el que deba tener en cuenta todos estos eventos para hacer una aplicación robusta. Los siguientes métodos son llamados, dentro de una actividad, automáticamente por el sistema cuando se produce una determinada circunstancia o evento. El programador podrá implementar las acciones a realizar para cada uno de estos métodos en función de la actividad donde se produzca el evento:

- **onCreate()**: Llamado cuando la actividad es llamada por primera vez. Es donde se debe crear la inicialización normal de la aplicación, crear vistas, etc. Este método da acceso al estado de la aplicación cuando se cerró. Después de esta llamada siempre se llama al método onStart().
- **onRestart()**: Llamado cuando la actividad ha sido parada y tiene que volver a ser lanzada. Es llamado siempre después onStart().
- **onStart()**: Llamado antes de que la actividad sea visible por el usuario. A continuación, se llama a onResume() si la actividad va a ser visible o a onStop() si se oculta por otra actividad.
- **onResume()**: Llamado cuando la actividad va a empezar a interactuar con el usuario, en este punto es el último punto antes de que el usuario vea la actividad y pueda empezar a interactuar con ella. Siempre después de onResume() viene onPause().
- **onPause()**: Llamado cuando el sistema va a empezar una nueva actividad. La actividad anterior permanece en segundo plano aún visible por el usuario, pero oculta por otra actividad. Ésta necesita parar animaciones, y parar todo lo que esté ejecutándose. Hay que intentar que esta llamada dure poco tiempo, porque hasta que no se ejecute este método no arranca la siguiente actividad. Después de esta llamada puede venir un onResume() si la actividad vuelve a primer plano o un onStop() si se hace invisible totalmente para el usuario.
- **onStop()**: Llamado cuando la actividad ya no es visible al usuario, porque otra actividad ha pasado a primer plano. Desde aquí, el sistema puede llamar a onRestart() si vuelve a primer plano o a onDestroy() si se elimina completamente la actividad.
- **onDestroy()**: Llamado cuando se necesita destruir o eliminar completamente una actividad para liberar recursos en memoria.

Debido a que cada aplicación se considera un proceso, y cada proceso se ejecuta en un hilo, para decidir qué proceso debe eliminar Android los ordena por importancia en:

- **Foreground Process:** Es la aplicación que contiene la actividad que ahora mismo se está mostrando en pantalla (Se ha llamado al método onResume()). Habrá muy pocos

procesos Foreground ejecutándose a la vez en el sistema –por lo general sólo el que se esté ejecutando en primer plano– y estos procesos solamente se eliminarán si la memoria es tan baja que ni eliminando al resto de procesos tiene el sistema los recursos necesarios.

- **Visible process:** Es el que contiene una actividad que es visible, pero no en primer plano (se ha llamando al método onPause()). Por ejemplo, estoy leyendo un email, y hago clic en un enlace http y abre el navegador y me lleva a esa web. En ese momento el navegador sería *Foreground Process* y el cliente de mail sería un *Visible process*. Estos procesos son considerados importantes por el sistema operativo y normalmente no se eliminan.
- **Service process.** Se asigna a un servicio. Estos procesos se ejecutan en segundo plano que, normalmente, son importantes. El sistema nunca va a liquidar un servicio a menos que sea necesario para mantener todos los Visible y Foreground process.
- **Background process.** Es un proceso que contiene una actividad que actualmente no es visible por el usuario, estos procesos no tienen demasiada importancia, puede ser una aplicación que se ejecutó hace tiempo y no se ha vuelto a usar; pasa a estar en background. Por eso es importante que cuando nuestra aplicación pase a Background, liberar en la medida de lo posible todos los recursos que podamos.
- **Empty process.** Es un proceso que no alberga nada ya, lo usa Android como caché para cuando se crea un proceso nuevo.

BLOQUE 2
METODOLOGÍA PARA WEB SCRAPING EN ANDROID

2.1Android y web scraping

Con la evolución de las nuevas tecnologías y la gran expansión que ha sufrido el mercado de las aplicaciones móviles, se hace necesario en muchas ocasiones recurrir al web scraping para poder brindar al usuario la información que está solicitando.

Los dispositivos móviles cada vez son más fiables y potentes, y un sistema operativo ya maduro como es Android es capaz de soportar e implementar aplicaciones capaces de ello.

Por suerte Android es un sistema operativo "Open Source" por lo que la creación y utilización de APIs se hace más sencilla que en otros sistemas.

Actualmente para el caso que nos ocupa, que es, la comunicación con un servidor web, la extracción del código HTML de la página web para, posteriormente, presentar la información relevante de este, disponemos de una serie de "APIs", no todas son validas para la tarea de web scraping; existen determinadas APIs que son útiles solamente para establecer la comunicación con el servidor web, otras que son las indicadas realizar el web scraping y algunas pueden realizar las dos tareas, que veremos a continuación:

APIs para la comunicación con el servidor web

HttpClient y HttpEntity:

Se trata de la biblioteca HttpClient de Apache, disponible para Android y que está pensada para implementar clientes HTTP. La principal función dicha biblioteca Apache HttpClient es la creación y el envío de peticiones HTTP (GET, POST, PUT, HEAD,…) así como de obtener las respuestas enviadas por el servidor. También soporta conexiones mediante HTTPS, pero no soporta JavaScript.
El establecimiento de una comunicación HTTP implica el intercambio de peticiones HTTP y respuestas HTTP, que normalmente son realizadas internamente por HttpClient de manera transparente al programador, el cual crea un objeto de tipo HttpClient rellenando las cabeceras de la petición que considere oportunas y HttpClient se encarga de transmitir la petición y de recibir la respuesta a través de una conexión TCP, elevando una excepción si hay algún problema.

HttpEntity se utiliza para implementar la parte del cuerpo de la petición HTTP; si se va a crear una petición que requiera el envío de datos que no se puedan añadir en las cabeceras, como por ejemplo una petición POST, se van añadiendo los datos necesarios en el cuerpo de esta petición a través de esta clase. En caso de que la respuesta del servidor contenga datos a procesar que no son cabeceras se utiliza también HttpEntity para recoger el cuerpo de la respuesta; por ejemplo para recoger el código HTML de una página web.

Selenium Client API y Selenium WebDriver:

Son interfaces de programación de aplicaciones (API) usadas en la mayoría de casos para realizar pruebas de software sobre sitios web y aplicaciones web; sin embargo, la diferencia que existe con HtmlUnit es que utilizan un navegador web para ejecutar dichas pruebas.
Al hacer uso del navegador web soporta tanto JavaScript para la carga de páginas web dinámicas como el protocolo HTTPS.
Selenium Client API permite que las pruebas puedan implementarse en varios lenguajes de programación como Java, C#, Ruby y Python.
Selenium WebDriver es la herramienta que "recoge" las pruebas enviadas por Selenium Client API y las envía al navegador mediante un controlador específico para cada navegador que además recoge también el resultado de dichas pruebas que el servidor manda en la respuesta. Su uso es similar al de HtmlUnit con la diferencia que para poder realizar web scraping es necesario obtener la versión que viene con el API HtmlUnit integrado.

WebView:

WebView es un API diseñada para visualizar páginas web en dispositivos móviles. Replica el comportamiento de un navegador web y posee interfaz gráfica. Utiliza el motor de código abierto WebKit para interactuar con un servidor web, mostrar páginas web, descargar archivos y administrar plugins. Soporta JavaScript y HTTPS.

Se trata pues de un pseudo-navegador web que, a diferencia de otros navegadores, es utilizable desde código.

HttpURLConnection:

Es una API similar a HttpClient y su funcionalidad es la misma, sin embargo tiene menos peso en memoria y consume menos recursos en el sistema. Basta con crear una instancia HttpURLConnection y abrir una conexión con el servidor mediante el método openConnection(URL url) para obtener un objeto HttpURLConnection con la conexión establecida, a partir de ahí, se pueden ir modificando las cabeceras de la petición así como

la propia petición (GET, POST, etc.) con los métodos setRequestProperty(String <nombre cabecera>, String <valor>) y setRequestMethod(String <petición>) respectivamente. La respuesta enviada por el servidor se obtiene mediante la creación de un objeto BufferedReader que contenga el inputStream del objeto HttpURLConnection en cuestión.

Esta API soporta comunicaciones mediante HTTPS, sin embargo carece de soporte para ejecutar JavaScript.

APIs que facilitan la extracción de datos

Actualmente existen numerosas herramientas que automatizan la realización de web scraping y son válidas para diferentes lenguajes de programación, sin embargo, la inmensa mayoría de ellas no son "Open Source" y no son gratuitas.

Por otra parte, existen APIs de código abierto y gratuito que, aunque no realizan la tarea de web scraping en sí, su principal función es facilitar y hacer semitransparente dicha tarea al programador que es el que se encarga de implementar el código para ello. A continuación, se destaca una de ellas:

Jsoup:

Es una interfaz para programación de aplicaciones Java diseñada para permitir a los usuarios trabajar con HTML. Proporciona una API muy conveniente para la extracción y manipulación de datos, utilizando lo mejor de los métodos DOM, CSS y jQuery. Jsoup está diseñado para hacer frente a todas las variedades de HTML que se encuentran en la web; desde los HTML perfectos y válidos a los marcados con formatos inválidos; es capaz de crear un árbol DOM a partir de ellos. También implementa la especificación de HTML5. Es posible, además, procesar código HTML desde una URL, un archivo o desde una cadena de caracteres.

Permite establecer una comunicación con el servidor para extraer un documento HTML del mismo, sin embargo, no soporta autenticaciones en servidores web, por lo que su uso para este campo es limitado.

Se usa para realizar un parseo de un documento HTML a un árbol DOM desde el que poder trabajar más fácilmente.

APIs para la comunicación con el servidor y además facilitan la extracción de datos

HtmlUnit

HtmlUnit es una API diseñada para replicar el comportamiento de un navegador web, sin embargo carece de la interfaz gráfica del navegador y está implementado en Java. Permite la manipulación de alto nivel de sitios web a partir de otro código Java, incluyendo el relleno y envío de formularios y pinchar hiperenlaces. También proporciona acceso a la estructura y los detalles de las páginas web recibidas. HtmlUnit emula el comportamiento de las partes del navegador, incluyendo los aspectos de bajo nivel de los protocolos TCP/IP y HTTP. Una secuencia como getPage(url), getLinkWith("Click here"), click() permite al usuario navegar a través de hipertexto y obtener páginas web, incluyendo HTML, JavaScript, AJAX y *cookies*. También puede tratar con seguridad HTTPS, autenticación HTTP básica, la redirección automática de páginas y otras cabeceras HTTP. Permite al código Java de testeo (probador) examinar las páginas devueltas en forma de texto, un DOM XML, o como colecciones de formularios, tablas y enlaces. El uso más común de HtmlUnit es como herramienta de testing de páginas web, pero a veces se puede utilizar como herramienta de web scraping o para descarga de contenido del sitio web.

2.2 Metodología

2.2.1 Metodología: definición y conceptos

Una metodología es un conjunto integrado de técnicas y métodos que permite abordar de forma homogénea y abierta cada una de las actividades del proyecto de desarrollo.

Las metodologías se basan en una combinación de los modelos de proceso genéricos (cascada, incrementa, etc.). Definen artefactos, roles y actividades, junto con prácticas y técnicas recomendadas.

La metodología para el desarrollo de software es un modo sistemático de realizar, gestionar y administrar un proyecto para llevarlo a cabo con altas posibilidades de éxito. Una metodología para el desarrollo de software comprende los procesos a seguir sistemáticamente para idear, implementar y mantener un producto software desde que surge la necesidad del producto hasta que se logra el objetivo por el cual fue creado.

Una definición estándar de metodología puede ser el conjunto de procedimientos que se utilizan en una determinada actividad con el fin de formalizarla y optimizarla. Determina los pasos a seguir y cómo realizarlos para finalizar una tarea.

Si esto se aplica a la ingeniería del software, cabe destacar que una metodología:

- Optimiza el proceso y el producto software.

- Define una serie específica de pasos o métodos que guían en la planificación y en el desarrollo del software.

- Define qué hacer, cómo y cuándo durante todo el desarrollo y mantenimiento de un proyecto.

Una metodología de desarrollo de software o metodología de desarrollo de sistemas es un marco de trabajo que se usa para estructurar, planificar y controlar el proceso de desarrollo de un sistema de información.

El **marco de trabajo** de una metodología de desarrollo de software consiste en:

- Una filosofía de desarrollo de software, con el enfoque o enfoques del proceso de desarrollo de software.

- Múltiples herramientas, modelos y métodos para ayudar en el proceso de desarrollo de software.

El uso de herramientas de apoyo para la aplicación de alguna metodología en el desarrollo del proyecto permite realizar un control de todo el proceso, obtener ayuda en la elaboración de documentación y facilitar los análisis de pruebas y las verificaciones del producto final.

2.2.2 Ventajas del uso de metodologías para desarrollo de software

Son muchas las ventajas que puede aportar el uso de una metodología. A continuación se van a exponer algunas de ellas, clasificadas desde distintos puntos de vista.

Desde el punto de vista de gestión:

- Facilitar la tarea de planificación.
- Facilitar la tarea del control y seguimiento de un proyecto.
- Mejorar la relación coste/beneficio.
- Optimizar el uso de recursos disponibles.
- Facilitar la evaluación de resultados y cumplimiento de los objetivos.
- Facilitar la comunicación efectiva entre usuarios y desarrolladores.

Desde el punto de vista de los ingenieros del software:

- Ayudar a la comprensión del problema.
- Independiza el desarrollo del proyecto de las habilidades del equipo de trabajo.
- Optimizar el conjunto y cada una de las fases del proceso de desarrollo.
- Facilitar el mantenimiento del producto final.
- Permitir la reutilización de partes del producto.

Desde el punto de vista del cliente o usuario:

- Garantía de un determinado nivel de calidad en el producto final.
- Confianza en los plazos de tiempo fijados en la definición del proyecto.
- Definir el ciclo de vida que más se adecue a las condiciones y características del desarrollo.

2.2.3 Desventajas del uso de metodologías para desarrollo de software

El uso de una metodología para desarrollo de software tiene también una serie de inconvenientes, aunque, dada la gran mayoría de ventajas que posee es bastante más recomendable su uso.

Algunas de las posibles desventajas pueden ser:

- Establecer un excesivo trabajo para proyectos de poca entidad.
- El uso de una metodología poco flexible puede dar lugar a dificultades para una conseguir una buena adaptación a determinadas situaciones.
- Requiere un tiempo de aprendizaje de la metodología, si se producen actualizaciones de los procedimientos.

2.2.4 Metodologías tradicionales y ágiles

Desarrollar un buen software depende de un gran número de actividades y etapas, donde el impacto de elegir la metodología para un equipo en un determinado proyecto es trascendental para el éxito del producto.

Según la filosofía de desarrollo se pueden clasificar las metodologías en dos grupos. Las metodologías tradicionales, que se basan en una fuerte planificación durante todo el desarrollo, y las metodologías ágiles, en las que el desarrollo de software es incremental, cooperativo, sencillo y adaptado.

Metodologías tradicionales

Las metodologías tradicionales son denominadas, a veces, de forma peyorativa, como metodologías pesadas.

Centran su atención en llevar una documentación exhaustiva de todo el proyecto y en cumplir con un plan de proyecto, definido todo esto, en la fase inicial del desarrollo del proyecto.

Estas metodologías tradicionales imponen una disciplina de trabajo sobre el proceso de desarrollo del software, con el fin de conseguir un software más eficiente. Para ello, se hace

énfasis en la planificación total de todo el trabajo a realizar y una vez que está todo detallado, comienza el ciclo de desarrollo del producto software.

Se centran especialmente en el control del proceso, mediante una rigurosa definición de roles, actividades, artefactos, herramientas y notaciones para el modelado y documentación detallada. Además, las metodologías tradicionales no se adaptan adecuadamente a los cambios, por lo que no son métodos adecuados cuando se trabaja en un entorno, donde los requisitos no pueden predecirse o bien pueden variar.

Una de las metodologías tradicionales más usadas son:

- **RUP**: *Rational Unified Process* es una de las metodologías más usadas para el análisis, implementación y documentación de sistemas orientados a objetos. Define claramente quién, cómo, cuándo y que debe hacerse en el proyecto. Se basa en 6 procesos: Adaptar el proceso, equilibrar prioridades, demostrar valor iterativamente, colaboración entre equipos, elevar el nivel de abstracción y enfocarse en la calidad.

Pero sin dudas adaptarse a la agitada sociedad actual implica ser "ágil", es decir, tener la capacidad de proveer respuestas rápidas y ser adaptables al cambio. Ambas cualidades siempre han sido deseables, pero en el entorno de negocio actual resultan indispensables. Este requerimiento de agilidad en las empresas, gobiernos y cualquier otra organización provoca que el software también deba ser desarrollado de manera ágil.

Las necesidades de un cliente pueden sufrir cambios importantes del momento de contratación de un software al momento de su entrega; y es mucho más importante satisfacer estas últimas que las primeras. Esto requiere procesos de software diferentes que en lugar de rechazar los cambios sean capaces de incorporarlos.

Metodologías ágiles

Las metodologías ágiles son una buena elección cuando se trabaja con requisitos desconocidos o variables. Si no existen requisitos estables, no existe una gran posibilidad de tener un diseño estable y de seguir un proceso totalmente planificado, que no vaya a variar ni en tiempo ni en dinero. En estas situaciones, un proceso adaptativo será mucho más efectivo que un proceso predictivo.

Las metodologías ágiles proporcionan una serie de pautas y principios junto a técnicas pragmáticas que puede que no curen todos los males pero harán la entrega del proyecto

menos complicada y más satisfactoria tanto para los clientes como para los equipos de entrega.

Estas metodologías ponen de relevancia que la capacidad de respuesta a un cambio es más importante que el seguimiento estricto de un plan.

Algunas de las metodologías más usadas por las empresas de desarrollo de software son:

- **XP**: *Extreme Programming*, se basa en la creencia que ser capaz de adaptarse a los cambios de requisitos en cualquier punto de la vida del proyecto es una aproximación mejor y más realista que intentar definir todos los requisitos al comienzo del proyecto e invertir esfuerzos después en controlar los cambios en los requisitos. Además, se promueve la mayor simplificación del código posible y la integración de cliente en el desarrollo del proyecto.

- **Scrum**: Define una serie de prácticas y roles como punto de partida para el proceso de desarrollo de software de un proyecto. En Scrum un proyecto se realiza en bloques temporales planificados por los miembros del equipo. Al final de cada bloque, se tiene que proporcionar un resultado del producto final que sea susceptible de ser entregado al cliente.

¿Metodologías ágiles o metodologías tradicionales?

Tener metodologías diferentes para aplicar de acuerdo con el proyecto que se desarrolle resulta una idea interesante. Estas metodologías pueden involucrar prácticas tanto de metodologías ágiles como de metodologías tradicionales. De esta manera podríamos tener una metodología por cada proyecto, la problemática sería definir cada una de las prácticas, y en el momento preciso definir parámetros para saber cuál usar.

En las metodologías tradicionales el principal problema es que nunca se logra planificar bien el esfuerzo requerido para seguir la metodología. El no poder predecir siempre los resultados de cada proceso no significa que estemos frente a una disciplina de azar.

Es importante tener en cuenta que el uso de un método ágil no vale para cualquier proyecto. Sin embargo, una de las principales ventajas de los métodos ágiles es su peso inicialmente ligero y por eso las personas que no estén acostumbradas a seguir procesos encuentran estas metodologías bastante agradables.

En la tabla que se muestra a continuación aparece una comparativa entre estos dos grupos de metodologías.

Metodologías ágiles	Metodologías tradicionales
Basadas en heurísticas provenientes de prácticas de producción de código	Basadas en normas provenientes de estándares seguidos por el entorno de desarrollo
Especialmente preparados para cambios durante el proyecto	Cierta resistencia a los cambios
Impuestas internamente (por el equipo)	Impuestas externamente
Proceso menos controlado, con pocos principios	Proceso mucho más controlado, con numerosas políticas/normas
No existe contrato tradicional o al menos es bastante flexible	Existe un contrato prefijado
El cliente es parte del equipo de desarrollo	El cliente interactúa con el equipo de desarrollo mediante reuniones
Grupos pequeños (<10 integrantes) y trabajando en el mismo sitio	Grupos grandes y posiblemente distribuidos
Pocos artefactos	Más artefactos
Pocos roles	Más roles que en las ágiles
Menos énfasis en la arquitectura del software	La arquitectura del software es esencial y se expresa mediante modelos

Figura 13. Metodologías ágiles y tradicionales

2.3 Propuesta de metodología para Web Scraping: WSM, Web Scraping Methodology

Existen múltiples metodologías, tantas como se quieran definir, ya que para cada proyecto se pueden crear o modificar metodologías existentes con el fin de mejorarlas y adaptarlas de forma única y específica a las necesidades de proyecto.

WSM se puede considerar como una metodología que tiene como objetivo el poder realizar Web Scraping correctamente. Se trata de una metodología cerrada puesto que no es adecuada para realizar una tarea distinta a la extracción y presentación de datos relevantes de un documento HTML.

WSM posee algunas características de la metodología XP como:

- Desarrollo iterativo e incremental: añadiendo pequeñas mejoras al código unas tras otras.
- Pruebas unitarias continuas frecuentemente repetidas.
- Programación en parejas, de esta manera el código es revisado y discutido mientras se escribe.
- Integración del equipo de programación con el usuario.
- Corrección de todos los errores antes de añadir una nueva funcionalidad.
- Propiedad del código compartida, así cualquier miembro del equipo puede corregir y extender cualquier parte del proyecto.
- Simplicidad en el código.

Cabe destacar que todas estas características se combinan con una que posee la gran mayoría de metodologías tradicionales: el establecimiento de una serie de procesos o fases para que el producto final cumpla con el requisito principal, que es poder obtener y presentar información ubicada en Internet relevante para un usuario, en el sistema operativo Android.

WSM establece una serie de fases para poder lograr su objetivo:

- Fase 1: Análisis de la comunicación entre cliente y servidor.
- Fase 2: Comunicación y extracción de la respuesta del servidor.

- Fase 3: Parseo y extracción de la información relevante al usuario.
- Fase 4: Presentación de la información al usuario.

Estas cuatro fases se detallarán a continuación.

Es necesario recalcar que Andoid es un sistema operativo para dispositivos móviles y está basado, en gran parte, en Java, por tanto, la mayoría de aplicaciones que se ejecutan en él están implementadas en Java aunque existen módulos de adaptación a otros lenguajes de programación como Ruby o C++ para poder crear aplicaciones con ellos. Esta metodología se centra exclusivamente en crear una aplicación, para una determinada tarea, utilizando Java.

2.4 Fase 1: Análisis de la comunicación entre cliente y servidor

En esta primera fase se concentra todo lo relativo al análisis de la comunicación que se va a establecer entre el cliente y el servidor.

Se va a tratar de replicar el comportamiento de un navegador web, por tanto, es necesario conocer previamente cómo se comunica el navegador web con el servidor donde se encuentran los datos relevantes para realizar las peticiones correctamente.

Como se ha mencionado en el bloque 1, un navegador web esta implementado de acuerdo al modelo cliente-servidor y, por tanto, su objetivo es ejercer como cliente web para establecer una comunicación con un servidor web; dicha comunicación estará formada por el intercambio de una serie de mensajes de tipo petición-respuesta y será iniciada siempre por el cliente con una o varias peticiones.

Los mensajes intercambiados durante la comunicación están definidos por el protocolo HTTP, es decir, están formados a partir un conjunto de normas y definiciones establecidas en este protocolo para que sea posible una correcta interpretación por parte de cliente y servidor.

Así pues, construir las peticiones HTTP en el cliente antes de mandarlas al servidor se convierte en fundamental para poder obtener la información relevante para el usuario. Para poder realizar correctamente estas peticiones es necesario analizar previamente como están formadas, para poder implementarlas igual que lo haría un navegador web. Por tanto, esta fase es de vital importancia para que el producto final sea totalmente funcional.

Existen numerosas herramientas que permiten visualizar la estructura de las peticiones HTTP enviadas al servidor y de las respuestas HTTP enviadas por este al cliente. Algunas se presentan como extensiones o aplicaciones para un determinado navegador web y otras están ya integradas en el mismo. A continuación se detallará algunas de las más usadas:

Mozilla Firebug

Es un paquete de utilidades para analizar, editar, monitorizar y depurar el código fuente de una página web. Se presenta como extensión o complemento para el navegador Firefox de Mozilla.

Es una herramienta destinada especialmente a desarrolladores web y empresas del sector. Se centra en los lenguajes CSS, HTML y JavaScript aunque tiene una opción con la que se puede visualizar los mensajes HTTP que se envían y se reciben en el navegador. Es una extensión "Open Source" y gratuita.

Firebug tiene un uso fácil e intuitivo, basta con descargar e instalar la herramienta e iniciar el navegador para que aparezca un icono a la derecha de la barra de dirección. Pulsando sobre ese icono se visualizará una ventana en la parte inferior del navegador con una serie de etiquetas en su parte superior.

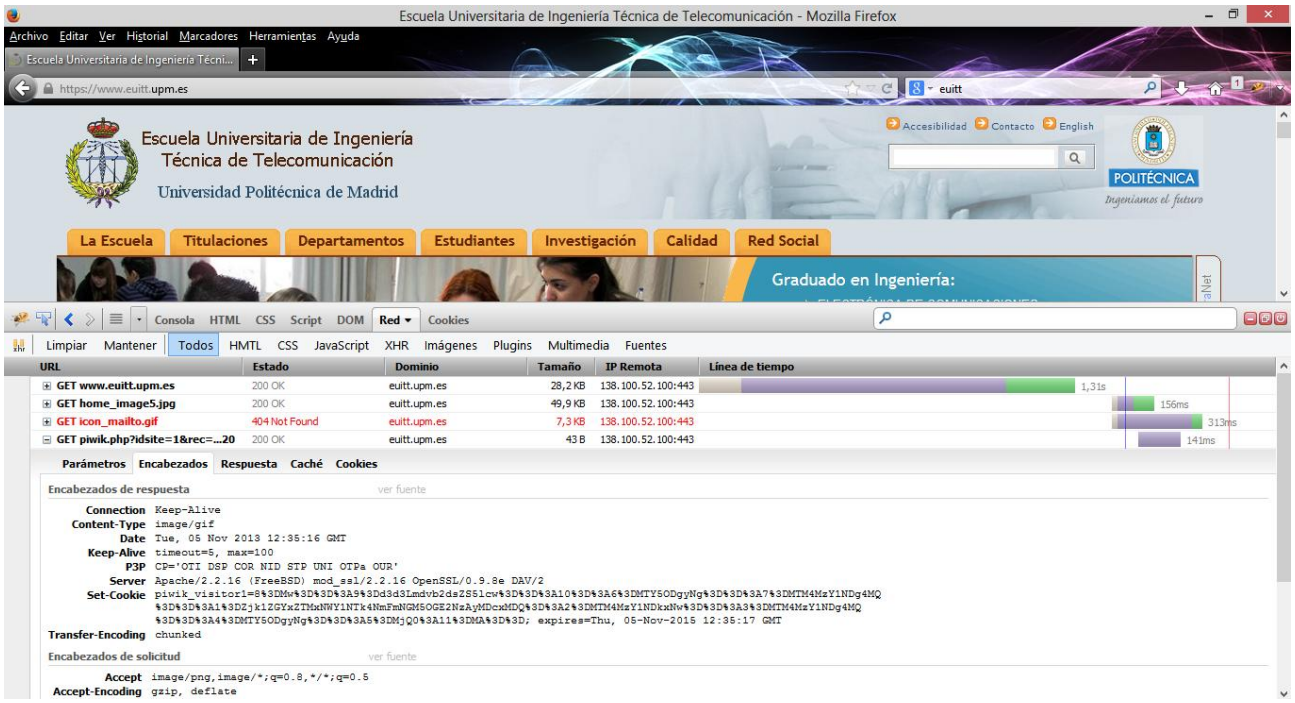

Figura 14. ***Utilización de Mozilla Firebug***

2.4 Fase 1: Análisis de la comunicación entre cliente y servidor

Mediante la pestaña *Red* se puede visualizar los mensajes HTTP enviados y recibidos con sus correspondientes cabeceras, el contenido de dichas cabeceras (sub-pestaña *Encabezados*) y el contenido del cuerpo del mensaje en caso de que hubiera (sub-pestaña *Respuesta)*.

HTTP Fox

HTTP Fox es una herramienta de monitorización y análisis de todo el tráfico HTTP de entrada y salida del navegador. Se presenta como extensión o complemento del navegador Firefox de Mozilla.

Presenta información de las cabeceras de petición y respuesta con su correspondiente valor, cookies enviadas y recibidas, cadenas Query enviadas como parámetro, parámetros de peticiones POST y el contenido del cuerpo de la respuesta en caso de haberlo.

Es una utilidad sencilla de usar; para ello basta con descargarse el complemento para Mozilla, instalarlo e iniciar el navegador. Una vez instalada, se ejecuta pulsando sobre *HTTP Fox,* a través de la opción *Desarrollador Web* de la pestaña *Herramientas* del navegador, y se elige *Open in Own Window.* Pulsando sobre la pestaña *Start* de la ventana que se ha abierto empezará a capturar el tráfico HTTP. Una vez capturados los mensajes relevantes, pulsando sobre *Stop* finalizará la captura de tráfico HTTP.

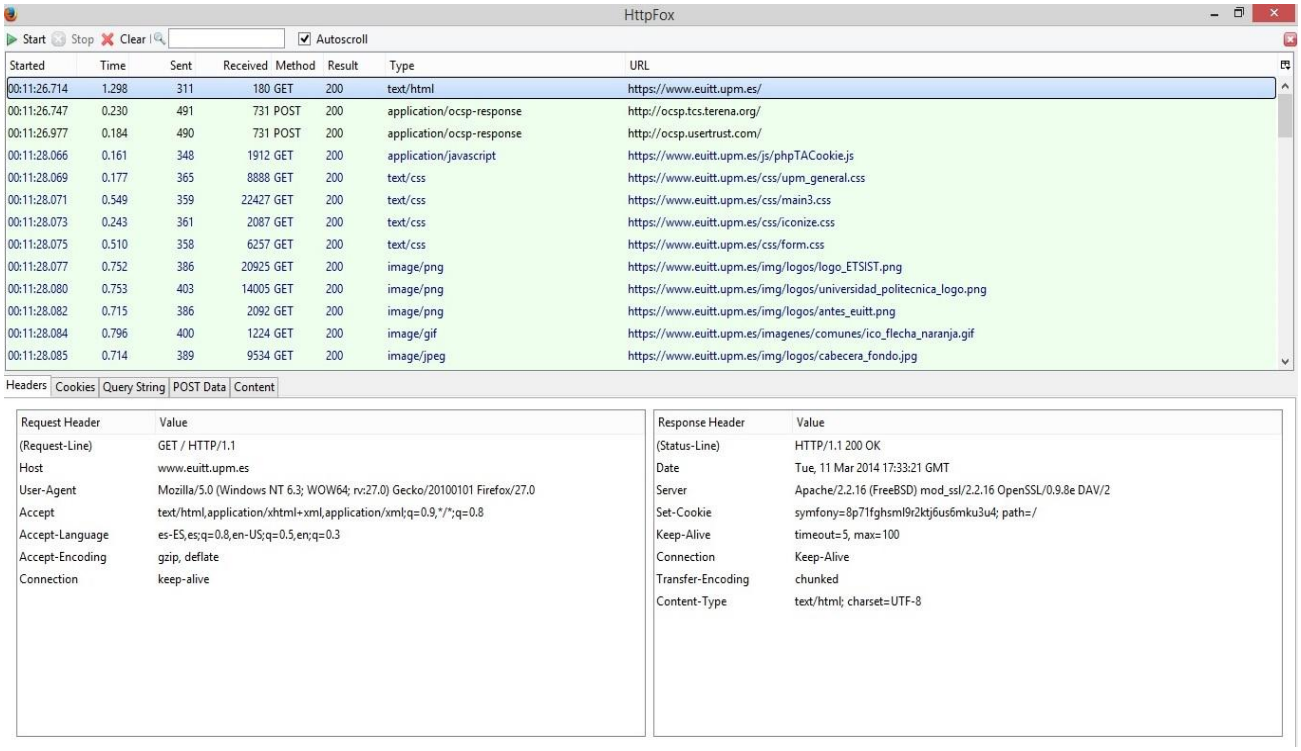

Figura 15. ***Captura del tráfico HTTP con HTTP Fox***

En la parte inferior de la ventana se puede visualizar los diferentes valores para cada petición pulsada en la lista de la parte superior.

Herramienta para Google Chrome

Esta herramienta viene integrada en el navegador Chrome de Google y tiene prácticamente la misma funcionalidad que la extensión Firebug de Mozilla: analizar, editar y monitorizar el código fuente de una página web.

A diferencia de Firebug y HTTP Fox, esta utilidad no se presenta como extensión o complemento del navegador sino que forma parte del mismo.

El uso de esta herramienta es sencillo, basta con pulsar con el botón derecho del ratón, en cualquier parte de la página web que se está visualizando, y seleccionar la opción *inspeccionar elemento* para que se abra una ventana desde la que se puede analizar el código HTML, el tráfico HTTP de entrada y salida, etc.

Para visualizar las peticiones HTTP es necesario seleccionar la opción *Network* en la parte superior de la ventana.

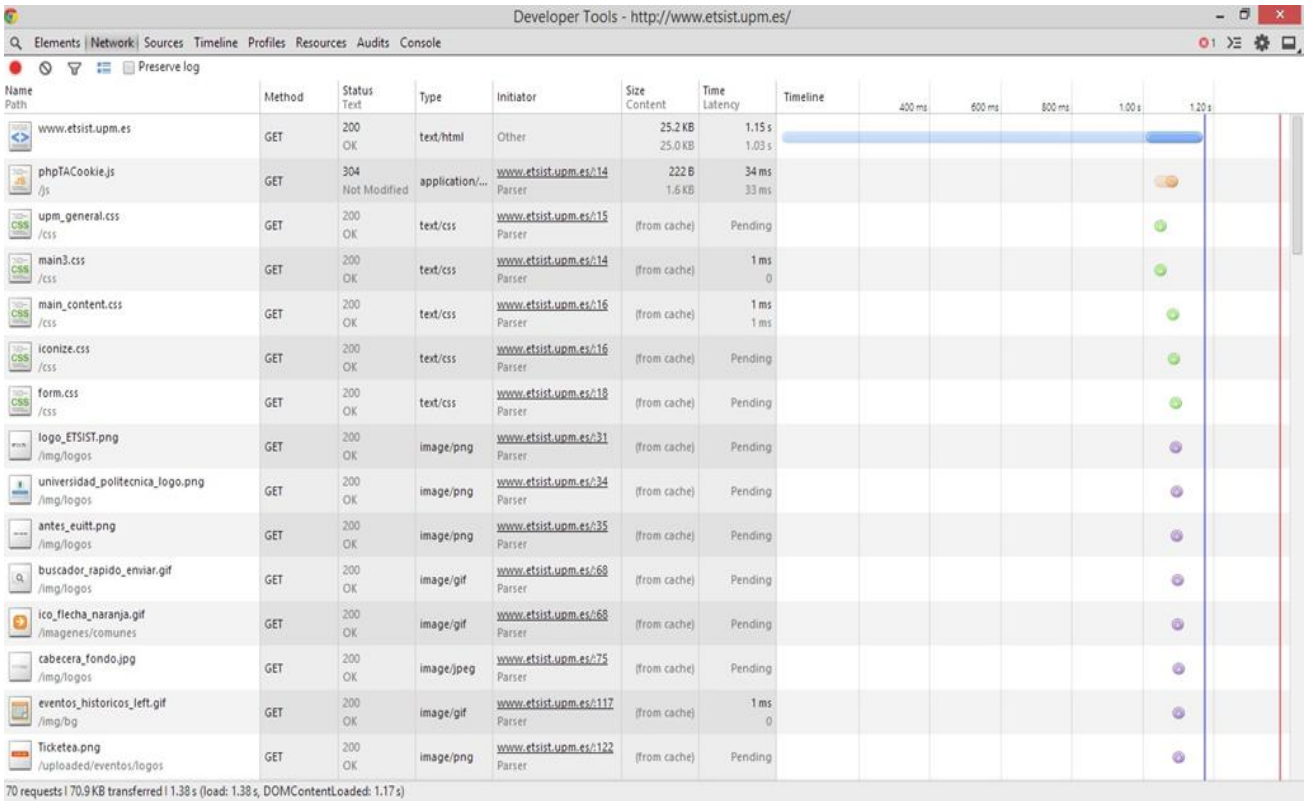

Figura 16. ***Captura del tráfico HTTP con la herramienta de Google Chrome.***

2.4 Fase 1: Análisis de la comunicación entre cliente y servidor

A continuación, si se quiere visualizar en detalle cada petición basta con pulsar sobre el mensaje deseado y aparecerá en la parte derecha de la ventana las cabeceras, contenido del mensaje (en caso de haberlo), etc.

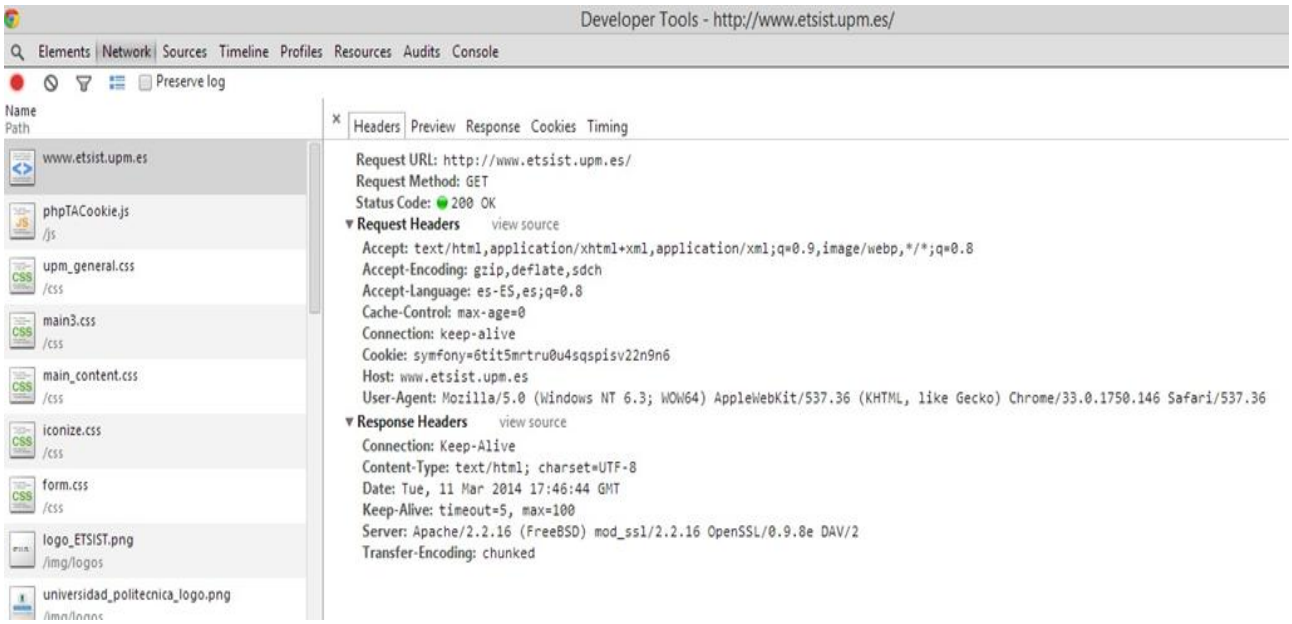

Figura 17. ***Detalle de una petición HTTP.***

Mediante este tipo de herramientas se analiza la estructura de las peticiones que se tienen que enviar al servidor para poder implementarlas posteriormente. Para ello es necesario utilizarlas contra el servidor del que se quiere obtener la información deseada. De esta manera se obtendrán los mensajes HTTP que se deben implementar.

Es importante tener en cuenta no sólo las cabeceras que se tienen que replicar sino, además, los datos que sean necesarios enviar en el cuerpo de la petición (en caso de que se requiera) así como el formato en que se tienen que enviar los mismos.

Una vez analizadas las peticiones HTTP y teniendo claro que campos de las cabeceras hay que añadir en cada petición, se avanzará a la fase 2 de esta metodología que es la implementación de la comunicación y la extracción de la respuesta enviada por el servidor.

2.5 Fase 2: Comunicación y extracción de la respuesta del servidor.

Esta fase se centra en la implementación de las peticiones HTTP analizadas en la fase 1. Para ello se utilizaran algunas APIs de las mencionadas en el apartado *Android y Web Scraping*.

El primer paso se centra en la implementación de las peticiones HTTP que se van a enviar al servidor. Para la realización de dicha tarea se van a usar las APIs *HttpURLConnection* y WebView dependiendo de si el objetivo es extraer la información de una página web estática o dinámica.

2.5.1 Comunicación para webs estáticas

Se considera que una web es estática cuando está implementada en HTML, no permite ejecutar funcionalidades escritas en otros lenguajes de programación que no sean HTML ni grandes libertades a la hora de crear efectos. Una web estática es básicamente informativa y está enfocada a mostrar la misma información permanentemente, el cliente se limita a obtener esa información sin poder interactuar con la web. Este tipo de webs carecen de un sistema de bases de datos.

Para crear la comunicación con servidores que utilicen este tipo de webs se pueden utilizar varias APIs en Android de las mencionadas en el apartado 2.1 como *HttpClient-HttpEntity*, *Selenium Client-Selenium WebDriver*, *WebView*, *HttpURLConnection*, *Jsoup* o *HtmlUnit*. Sin embargo el API más adecuada para esta tarea es *HttpURLConnection* puesto que es más sencilla de usar, consume menos recursos del sistema y ocupa un menor espacio que la mayoría de APIs mencionadas anteriormente.

HttpClient y *HttpEntity* consumen más recursos y ocupan más espacio en el sistema; con *Selenium Client* y *Selenium WebDriver* se dificulta el manejo de esta librería y consume todavía más recursos que *HttpClient* ya que se hace imprescindible la utilización del navegador web además de la aplicación; WebView solo se puede ejecutar desde el hilo principal de la aplicación y no permite ejecutarlo en un hilo secundario además de tener

interfaz gráfica; Jsoup no soporta las autenticaciones en páginas web al no almacenar datos de sesión y *HtmlUnit* no es compatible con Android.

HtmlUnit sería la más indicada para establecer una comunicación con el servidor y extraer los datos de la respuesta ya que es el API más sencillo y rápido, sin embargo, su incompatibilidad con Android por el momento la convierte en un API descartada para conseguir el objetivo final.

Selenium Client API y *Selenium WebDriver* son APIs que si bien pueden utilizarse para realizar Web Scraping, su complejo manejo y el uso obligatorio del navegador del sistema además de ser parcialmente incompatibles con Android las convierten en APIs descartadas para esta tarea en este sistema operativo.

Así pues para establecer la comunicación con el servidor y extraer la respuesta de este se recomienda el uso de HttpURLConnection o HttpsURLConnection para comunicaciones con HTTPS.

A continuación se detalla cómo establecer la comunicación haciendo uso de estas clases:

El primer paso es implementar una petición HTTP a partir de los datos analizados en la fase 1. Si se van a extraer datos de una página web estática basta con realizar una petición GET indicando la URL del sitio web donde está la información y rellenando los valores necesarios en las cabeceras. Esto se puede implementar mediante los siguientes métodos de la clase HttpURLConnection o HttpsURLConnection:

1º Crear un objeto URL con:

```
URL url = new URL(String <url>);
```

2º Crear la conexión con esta web mediante:

```
HttpURLConnection conn = (HttpURLConnection) url.openConnection( );
```

3º Crear la petición GET:

```
conn.setRequestMethod("GET");
```

4º Rellenar cabeceras y parámetros necesarios conforme al análisis de la fase 1:

- Parámetros de la conexión:

```
conn.setDoInput(true);      para poder recoger la respuesta del servidor

conn.setUseCaches(boolean)      para permitir (true) o no (false) el uso de memoria caché

HttpsURLConnection.setFollowRedirects(boolean);     para permitir (true) o no (false) redirecciones
```

- Las cabeceras se modifican con este método; es necesario llamar a este método cada vez que se requiera modificar una cabecera:

```
conn.setRequestProperty(String <Nombre cabecera>, String <Valor cabecera>);
```

Una vez completados estos cuatro pasos ya se ha enviado la petición GET al servidor.

2.5.2 Extracción de la respuesta

Para obtener la respuesta basta con seguir una serie de pasos:

1º Obtener el objeto InputStream donde está la respuesta del servidor:

2.5 Fase 2: Comunicación y extracción de la respuesta del servidor.

```
BufferedReader lector = new BufferedReader(new
InputStreamReader(conn.getInputStream( ), Charset.forName("<codificación>")),
<longitud del buffer>);
```

Nota: "<codificación>" es un objeto String con la codificación de texto del código HTML de la página web, esta codificación se obtiene en la fase 1 analizando el código HTML en la etiqueta *HEAD*; las codificaciones más comunes son ISO-8859-15 y UTF-8.

Nota: <longitud del buffer> es un entero con la longitud del buffer de lectura expresada en número de caracteres.

2º Se lee progresivamente, línea a línea, del buffer de entrada creado y se va uniendo en una única cadena de caracteres hasta que ya no hay más datos que leer:

```
StringBuilder  constructor = new StringBuilder();
String html_completo = null;
String linea = null;
        while ((linea = lector.readLine()) != null) {
                constructor.append(linea);
        }
html_completo=constructor.toString();
lector.close();
conn.disconnect( );
```

Nota: La variable lector es el objeto BufferedReader creado anteriormente. Una vez obtenido el contenido de la respuesta cerramos la conexión con el servidor llamando al método *disconnect( ).*

Una vez realizados estos pasos, en función de las peticiones que sean necesarias enviar al servidor, se obtendrá el código HTML donde se encuentra la información relevante que será extraída en la fase 3.

2.5.3 Comunicación para webs dinámicas

En una página web dinámica la información presentada es generada por el servidor a partir de una petición realizada por el cliente. Dependiendo del tipo de petición y del tipo de usuario se mostrará una información u otra distinta. Este tipo de webs están asociadas a una o varias bases de datos desde las que se extrae la información que se va a enviar al cliente; la extracción de esta información se realiza a través de una o varias aplicaciones implementadas en lenguajes de programación web como PHP, Perl CGI, JSP o ASP entre otros. La página generada y enviada es un documento HTML que puede contener o no scripts.

En esta metodología se tratarán por separado las webs dinámicas que contengan scripts de las que no los contengan ya que la manera de proceder será diferente debido a las limitaciones de Android para obtener y extraer webs que contengan scripts.

Webs que no contengan scripts

Las páginas web que no contengan scripts se tratarán de una manera similar a las webs estáticas salvo en los casos en que sea necesario autenticarse para acceder al recurso solicitado. Al igual que con las webs estáticas se recomienda el uso del API HttpURLConnection o HttpsURLConnection para conexiones mediante HTTPS.

En los casos en que para acceder al documento HTML sea necesario autenticarse previamente mediante un *log in* habrá que realizar una petición de tipo POST para enviar los datos de autenticación así como obtener la/-s cookies de sesión que devuelva el servidor. El mecanismo para realizar una petición POST es prácticamente igual que para una petición GET cambiando algunos parámetros y adjuntando en el cuerpo los datos necesarios:

1º Crear un objeto URL con:

```
URL url = new URL(String <url>);
```

2º Crear la conexión con esta web mediante:

2.5 Fase 2: Comunicación y extracción de la respuesta del servidor.

```
HttpURLConnection conn = (HttpURLConnection) url.openConnection( );
```

3º Crear la petición POST:

```
conn.setRequestMethod("POST");
```

4º Rellenar cabeceras y parámetros necesarios conforme al análisis de la fase 1:

- Parámetros de la conexión:

```
conn.setDoInput(true);      para poder recoger la respuesta del servidor

conn.setUseCaches(boolean);      para permitir (true) o no (false) el uso de memoria caché

conn.setDoOutput(true);    para indicar que vamos a enviar datos al servidor. Necesario que el valor sea true si se va a realizar un POST
HttpsURLConnection.setFollowRedirects(boolean);    para permitir (true) o no (false) redirecciones
```

- Las cabeceras se modifican con este método; es necesario llamar a este método cada vez que se requiera modificar una cabecera:

```
conn.setRequestProperty(String <Nombre cabecera>, String <Valor cabecera>);
```

Con esto estaría casi lista la petición POST, únicamente faltaría adjuntar los datos a enviar en el cuerpo del mensaje; para ello creamos una cadena de caracteres que contenga los datos necesarios codificados de acuerdo al formato que acepte el servidor (UTF-8, ISO 8859-15, etc.). Un ejemplo sería:

2.5 Fase 2: Comunicación y extracción de la respuesta del servidor.

```
String output =
"accion="+URLEncoder.encode("validar",HTTP.UTF_8)+"&user="+URLEncoder.encode(u
suario, HTTP.UTF_8)+"&pass="+
URLEncoder.encode(password,HTTP.UTF_8)+"&conectar="+URLEncoder.encode("Conec
tar", HTTP.UTF_8);
```

Con esto se enviaría la siguiente cadena de caracteres, codificada en UTF-8, al servidor en el cuerpo del mensaje:
acción=&user=<usuario>&pass=<contraseña>&conectar=Conectar

Nota: la cadena de caracteres se tiene que ser exactamente igual a la que envía el navegador y que es analizada en la fase 1.

A continuación se crea un objeto DataOutputStream que contenga la salida de la conexión y se envía la cadena que se ha creado anteriormente al servidor:

```
DataOutputStream dataout = new DataOutputStream(conn.getOutputStream( ));
dataout.writeBytes(output);
```

Nota: Al llamar al método writeBytes es cuando se realiza la petición POST.

El siguiente paso es obtener los datos que se buscan, para ello habrá que realizar alguna petición GET o POST más dependiendo del servidor; sin embargo, para poder avanzar es necesario primero comprobar si la respuesta del servidor a la petición POST realizada es la correcta o ha ocurrido algún tipo de problema y si esa respuesta es la correcta será necesario obtener las cookies de sesión ya que tendrán que ser enviadas al servidor en cada petición realizada a continuación.

Para comprobar si la operación POST es correcta se utiliza el método que devuelve un entero con el código HTTP que envía el servidor en la respuesta: conn.getResponseCode( );

Las cookies de sesión se obtienen en varios sencillos pasos:

2.5 Fase 2: Comunicación y extracción de la respuesta del servidor.

Primero se obtienen todas las cabeceras de la respuesta y acto seguido se obtiene la cabecera “Set-Cookie” donde se encuentran las cookies que envía el servidor al cliente; una vez obtenida la cabecera se extrae la/-s cookie/-s de sesión. A continuación se muestra un ejemplo que recoge una cookie:

```
Map<String, List<String>> cabeceras = conn.getHeaderFields();
        String cookie = "";
        if(cabeceras!=null){
                List<String> lista_cookies = cabeceras.get("Set-Cookie");
                        if((lista_cookies != null)&&(lista_cookies.size()>0)){
                        cookie = lista_cookies.get(<posición de la
                        cookie>).substring(0,lista_cookies.get(<posición de la
                        cookie).indexOf("path")).trim();
                        }
        }
```

Nota: <posición de la cookie> es un entero con el valor de la posición de la cookie deseada, esta posición se obtiene en la fase 1 de análisis, de lo contrario puede ser que se obtenga una cookie que no es la que se quiere. Con el método substring(<inicio>,<offset>) se deja únicamente la cookie con su valor ya que lo que viene a continuación, que es el *path*, no es necesario enviarlo al servidor.

Cuando se han realizado todas las peticiones necesarias para llegar al recurso deseado, la extracción del código HTML de la página web se realiza de la misma manera a como se especifica para las webs estáticas.

Webs que contengan scripts

Debido a limitaciones que tiene Android para ejecutar Scripts, el tratamiento de este tipo de webs será totalmente diferente en esta fase ya que la información buscada dependerá de la ejecución en el cliente de los Scripts que contenga el documento HTML. Por tanto se utilizará un API que permita la ejecución de JavaScript en el cliente.

En un principio se puede considerar que el API *HmtlUnit* es el más indicado para este tipo de operaciones dado que replica a un navegador y por tanto soporta la ejecución de JavaScript; además carece de interfaz gráfica por lo que puede ser utilizado en un hilo secundario no teniendo que hacerlo en el principal. Sin embargo, a pesar de ser el API más

sencillo de usar de los mencionados en el apartado *2.1 Android y Web Scraping* está formado por una serie de librerías que son incompatibles con Android por el momento ya que vienen incorporadas como librerías nativas de este sistema operativo y por tanto entran en conflicto con las de *HtmlUnit* que aunque tengan el mismo nombre carecen de la misma funcionalidad.

Selenium Client API y *Selenium WebDriver* presentan un caso similar, en algunas de sus librerías, al de *HtmlUnit* siendo, además, no tan sencillas de utilizar como otras APIs.

Así pues la única alternativa restante que presente una relativa sencillez de uso sería el API *WebView*, que replica el comportamiento de un navegador web con la problemática, para esta tarea, de que posee interfaz gráfica y el sistema no permite que sea ejecutado en un hilo secundario teniendo que hacerlo obligatoriamente en el hilo principal de la aplicación.

Dado que, por el momento, no existe una alternativa mejor, el uso de este API es el recomendado para este tipo de webs.

El primer paso a realizar con este API, basándose en los datos obtenidos en la fase 1 de análisis, es comprobar la página web a la que se quiere acceder y analizar si requiere autenticación de usuario para poder avanzar o de lo contrario se puede acceder directamente.

Para aquellas webs en las que se puede acceder al documento HTML directamente bastará con cargar la URL de la página web y extraer el documento. La extracción del mismo no resulta tan sencilla ya que WebView es un API de visualización de páginas web y carece de métodos de extracción del código HTML visualizado. Sin embargo, al permitir la ejecución de JavaScript, se podrá extraer dicho documento mediante esta opción que se detallará más adelante.

El acceso a la página web se realiza de la siguiente forma:

1º Es necesario crear un *Layout* en XML que será una vista de la aplicación. Esta vista contendrá un *Webview* al que se le asigna un identificador:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"
   android:layout_height="match_parent">

   <WebView          android:id="@+id/webView1"
                android:layout_width="match_parent"
```

```
            android:layout_height="match_parent"/>

</LinearLayout>
```

2º En el *Activity* donde se quiera utilizar se creará una variable de tipo WebView que será la que recibirá la vista de *Layout* y ejecutará los métodos necesarios para cargar la página web. Como la página web tiene scripts, será necesario habilitar la ejecución de JavaScript que viene desactivada por defecto; además para recoger el código HTML de la web habrá que implementar una clase llamada "MyJavaScriptInterface" y añadirla como interfaz de JavaScript mediante el método *addJavaScriptInterface(Object object, String nombre)*, más adelante se detallará la implementación de esta clase:

```
WebView navegador;
String documento;

protected void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            setContentView(R.layout.gestor_pv);
            navegador = (WebView) findViewById(R.id.webView1);

            navegador.getSettings().setJavaScriptEnabled(true);
            navegador.getSettings().setUserAgentString("Mozilla/5.0 (Windows NT
            6.3; WOW64; rv:26.0) Gecko/20100101 Firefox/26.0");          //Para
            que no nos muestre la version móvil

            navegador.addJavascriptInterface(new MyJavaScriptInterface(this),
            "INTERFAZ");
```

3º A continuación se accede a la web deseada y se gestionan posibles eventos que puedan ocurrir durante la carga, para ello se crea un objeto WebViewClient y se asocia al WebView, acto seguido dentro de este WebViewClient se gestionan los eventos:

2.5 Fase 2: Comunicación y extracción de la respuesta del servidor.

```
        navegador.setWebViewClient(new WebViewClient() {
                @Override
                //SE SOBREESCRIBE ESTE METODO PARA QUE CARGUE LAS NUEVAS URL
                EN EL WEBVIEW Y NO EN OTRO NAVEGADOR
                public boolean shouldOverrideUrlLoading(WebView view, String url) {
                        view.loadUrl(url);
                        return true;
                }
                @Override
                //SE SOBREESCRIBE ESTE METODO PARA GESTIONAR POSIBLES ERRORES
                AJENOS A LA COMUNICACIÓN (RED, TIEMPO DE ESPERA, CONEXIÓN, ETC);
                NO GESTIONA ERRORES HTTP
                public void onReceivedError(WebView view, int errorCode, String
                description, String failingUrl) {
                        super.onReceivedError(view, errorCode, description, failingUrl);
                                //ACCIONES…
                 }
                @Override
                // SE SOBREESCRIBE ESTE METODO PARA GESTIONAR LAS ACCIONES A
                REALIZAR CUANDO SE HA TERMINADO DE CARGAR LA PAGINA
                public void onPageFinished(WebView view, String url) {
                        super.onPageFinished(view, url);
                //SE OBTIENE EL CODIGO HTML DE LA PAGINA MEDIANTE JAVASCRIPT
                        view.loadUrl("javascript:window.INTERFAZ.processHTML('<html>'+
                        document.getElementByTagName('html').innerHTML+'</html>');");
                }
        } );  //FIN WEBVIEWCLIENT
        navegador.loadUrl(“URL”);//SE CARGA LA WEB DESEADA
}//FIN ONCREATE
```

Nota: El método *processHTML(String texto)* que sirve para obtener el código HTML se detallará en la implementación de la clase MyJavaScriptInterface.

2.5 Fase 2: Comunicación y extracción de la respuesta del servidor.

4º Para recoger correctamente el documento HTML es necesario implementar la clase MyJavaScriptInterface mencionada anteriormente:

```
class MyJavaScriptInterface {
        Context mContext;

        MyJavaScriptInterface(Context c) {
            mContext = c;
        }

        @JavascriptInterface
        public void processHTML(String html)
        {
            documento = html ;
        }
}
```

Nota: Es necesario pasar un objeto *Context* cuando se llama al constructor; dado que se tiene que ejecutar en la Actividad donde se encuentra el WebView en la llamada desde esta misma se pasa como parámetro *this*. La variable *documento* es una cadena de caracteres creada en la Actividad junto con la variable WebView.

Con todo esto implementado, ya se habría conseguido el documento HTML de la página web y solo restaría extraer la información relevante y presentarla, pero eso pertenece a las fases 3 y 4 respectivamente.

Los pasos que se acaban de detallar hacen referencia a la obtención del código HTML de la página web para aquellas que no requieran de autenticaciones previas. Para los casos en que el acceso a dicho código HTML requiera de una autenticación previa con usuario y contraseña el proceso a seguir sería el mismo, con la única diferencia que se tendría que modificar el método *onPageFinished* del objeto WebViewClient para adaptarlo a las condiciones impuestas.

En aquellos casos en los que sea necesario *loguearse* para poder seguir adelante, existen dos maneras de introducir los datos de autenticación dependiendo de si se permite la interacción del usuario con el WebView o no.

2.5 Fase 2: Comunicación y extracción de la respuesta del servidor.

En el caso en que se permita dicha interacción del usuario, bastará con que este introduzca sus datos en los campos correspondientes que muestre el WebView y vaya avanzando (pinchando los enlaces que corresponda) hasta dar con la vista de la que quiere extraer la información. En tal caso, el método *onPageFinished* quedaría de la siguiente manera:

```
@Override
// SE SOBREESCRIBE ESTE METODO PARA GESTIONAR LAS ACCIONES A
REALIZAR CUANDO SE HA TERMINADO DE CARGAR LA PAGINA
public void onPageFinished(WebView view, String url) {
        super.onPageFinished(view, url);
        if(url.contains("URL donde se encuentra la información deseada")){
        //SE OBTIENE EL CODIGO HTML DE LA PAGINA MEDIANTE
        JAVASCRIPT
        view.loadUrl("javascript:window.INTERFAZ.processHTML('<html>'+
        document.getElementByTagName('html').innerHTML+'</html>');");
        }
}
```

Nota: Mediante la condición **if** se comprueba que la URL sea donde se encuentra la información relevante; se podría utilizar el método *url.equals("URL buscada")* en lugar del método *url.contains("URL buscada")*.

Esta metodología no recomienda esta opción dado que el WebView sería visible y, por tanto, el usuario vería la información que busca, como en un navegador normal, antes de extraerla y presentarla en la aplicación con lo que no tendría mucho sentido la creación de la aplicación.

Por tanto, lo recomendable sería que el usuario no pudiera ver el WebView ni interactuar con él. Para ello, sería necesario modificar el Layout creado al principio de la siguiente manera:

2.5 Fase 2: Comunicación y extracción de la respuesta del servidor.

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"
   android:layout_height="match_parent">

   <WebView          android:id="@+id/webView1"
                  android:layout_width="match_parent"
                  android:layout_height="match_parent"
                  android:visibility="invisible"/>

   <RelativeLayout   android:layout_width="match_parent"
                     android:layout_height="match_parent">

      <ProgressBar
         android:id="@+id/progressBar1"
         style="?android:attr/progressBarStyleLarge"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:layout_centerHorizontal="true"
         android:layout_centerVertical="true" />

      <TextView
         android:id="@+id/textView1"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:layout_below="@+id/progressBar1"
         android:layout_centerHorizontal="true"
         android:text="@string/conectando"
         android:textAppearance="?android:attr/textAppearanceLarge" />

   </RelativeLayout>

</FrameLayout>
```

De esta forma, añadiendo el atributo “invisible” al WebView y superponiendo otro Layout encima con el texto anterior, el usuario vería esto:

Figura 18. ***Vista de la Actividad.***

Una vez hecho esto, los casos anteriores quedarían igual, salvo el que requiere autenticación previa que cambiaría ya que el usuario no puede introducir los datos de autenticación en el WebView.

Nótese que para que todo esto sea posible, tanto en el uso del API HttpURLConnection como de WebView, es necesario pedir previamente los datos de autenticación al usuario en aquellos casos en que se requieran.

Así pues, en el caso en que sea necesario autenticarse previamente y dado que el usuario no puede interactuar con el WebView, los datos solicitados se introducirán a través de JavaScript en el método *onPageFinished* que quedaría de la siguiente manera:

2.5 Fase 2: Comunicación y extracción de la respuesta del servidor.

```
@Override
// SE SOBREESCRIBE ESTE METODO PARA GESTIONAR LAS ACCIONES A REALIZAR
CUANDO SE HA TERMINADO DE CARGAR LA PAGINA
public void onPageFinished(WebView view, String url) {
        super.onPageFinished(view, url);
        if(!login){                // SI NO NOS HEMOS LOGUEADO, LO HACEMOS
                view.loadUrl("javascript:{" +
                "document.getElementById('user').value = '"+usuario +"';" +
                "document.getElementById('pass').value = '"+password+"';" +
                "document.getElementById('form_login_enviar').click();" +
                " };");
                //SE BUSCAN LOS CAMPOS DEL FORMULARIO DE LOGIN, SE RELLENAN Y
                SE HACE EL LOGIN
                login= true;
        }
         if(url.contains("URL donde se encuentra la información deseada")&& (login){
        //SE OBTIENE EL CODIGO HTML DE LA PAGINA MEDIANTE JAVASCRIPT
        view.loadUrl("javascript:window.INTERFAZ.processHTML('<html>'+document.ge
        tElementByTagName('html').innerHTML+'</html>');");
        }
}
```

Nota: Es necesario el uso de flags en los casos en los que se carguen más de una página en el proceso de búsqueda de la página deseada ya que cada vez que se termina de cargar una página se llama automáticamente al método *onPageFinished*. Los flags que se necesiten **deben declararse** como variables booleanas **fuera del WebViewClient**, o bien dentro del método *onCreate* o bien en la declaración de variables de la Actividad. En este caso *login* es una variable booleana que indica si se ha realizado el login o no.

Las variables usuario y password son los datos que se han solicitado previamente al usuario. En los métodos *document.getElementById('elemento')* la cadena *'elemento'* es el campo del formulario de autenticación que se tendrá que rellenar o enviar según proceda y se obtienen en la fase 1 de análisis.

En aquellos casos en que sea necesario ir pinchando enlaces , una vez se ha producido la autenticación, para llegar a la página deseada, se tendrá que realizar también a través de JavaScript mediante:

2.5 Fase 2: Comunicación y extracción de la respuesta del servidor.

```
view.loadUrl("URL del enlace o JavaScript que se ejecute al pinchar el enlace");
```

Además, se tendrán que gestionar las acciones correspondientes, en el método *onPageFinished*, para cuando termine de cargar el enlace pinchado.

A partir de este punto se tendrán que gestionar en el método *onPageFinished* los eventos que surjan de autenticaciones incorrectas, errores en los datos de autenticación, etc. mediante el uso de flags, lanzamiento de excepciones, etc.

Una vez obtenido el código HTML de la página web, el proceso pasa a la fase 3 que se centra en la extracción de la información relevante.

2.6 Fase 3: Parseo y extracción de la información relevante al usuario.

Esta fase se centra en la conversión del documento HTML obtenido en la fase 2 en un árbol DOM para, posteriormente, extraer del mismo la información importante para el usuario.

Para realizar esta tarea se dispondría de dos APIs capaces de extraer únicamente los datos relevantes del documento HTML: HtmlUnit y Jsoup.

Como ya se comentó anteriormente, la utilización de HtmlUnit es, por el momento, incompatible con el sistema operativo Android, por tanto, se recomienda el uso de Jsoup, ya que es de gran versatilidad y pone a disposición de los desarrolladores suficiente documentación como para que su utilización sea cómoda y eficiente.

Nótese que Jsoup no es una herramienta que realice automáticamente Web Scraping, como otras que puedan existir en el mercado, por tanto, será el desarrollador el que "automatice" esta tarea mediante la implementación de la misma haciendo uso de este API.

El primer paso para ello es la verificación de que el documento HTML extraído de la web se encuentra almacenado en una variable de tipo String. A continuación se comprobará que dicho documento coincide con el documento HTML fijado como objetivo en la fase 1 de la metodología. Si no se cumple alguno de estos requisitos iniciales para esta fase, será necesario volver a fases anteriores hasta que se satisfagan dichos requisitos.

En el caso en que se hayan cumplido con estas condiciones, se podrá avanzar hacia el parseo de dicho documento. Para ello Jsoup dispone de una serie de clases y métodos que permiten simplificar al máximo esta tarea.

Para realizar la conversión del documento HTML a un árbol DOM bastaría con declarar un objeto de tipo *Document* y llamar al método estático *parse* de la siguiente forma:

```
//====================== PARSEO DE LA PAGINA HTML A UN DOCUMENTO DOM
Document documento = Jsoup.parse(html);
```

Donde *html* es la variable String donde está almacenado el código HTML extraído de la web.

2.6: Fase 3: Parseo y extracción de la información relevante al usuario.

A partir de aquí ya se tendría un árbol DOM en el que el elemento raíz sería la etiqueta <html> y los nodos descendientes serían la etiqueta <head> y la etiqueta <body> donde se encontraría la información buscada. Los nodos descendientes de <head> y <body> serían las etiquetas que se encontrasen dentro de cada uno de ellos respectivamente.

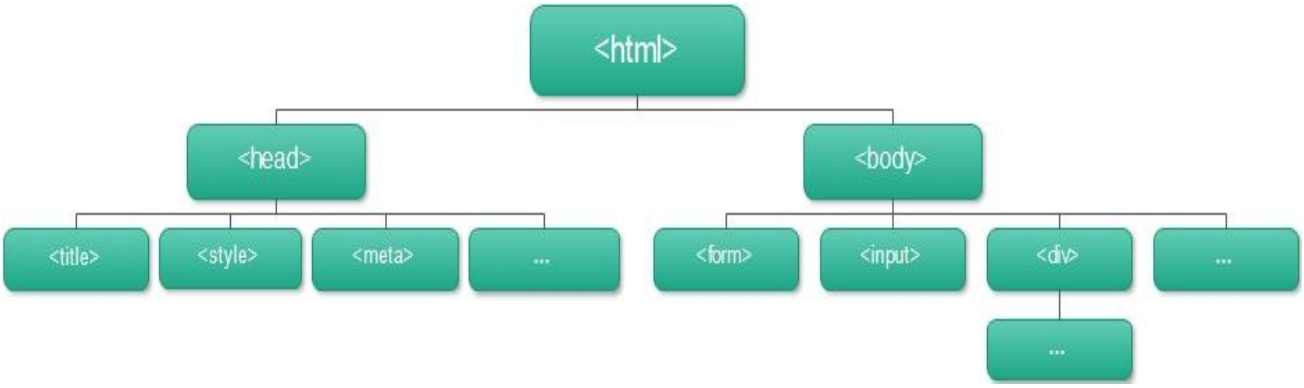

Figura 19. ***Ejemplo de árbol DOM creado a partir de un documento HTML.***

Una vez obtenido el árbol DOM, el siguiente paso consiste en extraer sólo la información que interesa; esta información se encontrará dentro de uno o varios nodos descendientes de <body>, por tanto, la tarea es encontrar esos nodos y obtener la información que contienen. El nombre de los nodos que se tienen que buscar tendrá que obtenerse en la fase 1 de análisis.

Para la búsqueda de dichos nodos Jsoup pone a disposición dos clases llamadas *Element*[2] y *Elements*[3] que mediante el uso de sus diferentes métodos se conseguirá llegar a los nodos deseados.

Existen numerosas formas de obtener la información deseada con estas dos clases, por tanto, deberá ser el desarrollador el encargado de implementar dicha tarea, de la forma que mejor se adapte a los criterios de búsqueda que establezca.

También es el desarrollador el encargado de decidir el tipo de variable en la que almacenarán los datos obtenidos en función de cuanta información se busque y del tipo de información que sea.

En el momento en que se han extraído todos los datos buscados lo único que restaría sería presentarlos al usuario.

[2] La documentación de la clase *Element* se encuentra en: http://jsoup.org/apidocs/org/jsoup/nodes/Element.html

[3] La documentación de la clase *Elements* se encuentra en: http://jsoup.org/apidocs/org/jsoup/select/Elements.html

2.7 Fase 4: Presentación de la información al usuario.

Esta es la fase final de la metodología y como su nombre indica se centra en la presentación de la información al usuario. En ella, los datos que se han extraído en la fase 3 se presentan al usuario de manera que este sólo visualiza los datos que quiere ver sin información ambigua y repetitiva.

Esta fase se centra más en el desarrollo de una Vista y de una Actividad en una aplicación Android que en la tarea de Web Scraping, sin embargo, posee igual o mayor importancia que todas las fases anteriores porque es el usuario final el que decide si realmente el producto creado cumple sus expectativas o no en función de la información que le presente y de cómo se la presente.

Así pues, el desarrollador tendrá que cuidar al máximo la presentación de los datos, estableciendo un criterio de presentación de los mismos en función del tipo de información solicitada. La claridad y la sencillez en la presentación elevarán la calidad del producto.

Se recomienda iniciar una nueva Actividad para presentar los datos extraídos en la que se puede hacer uso de cualquier clase y utilidad de las que dispone el sistema Android para definir e implementar una interfaz gráfica sencilla e intuitiva para el usuario.

BLOQUE 3
APLICACIÓN UPMdroid

3.1 Introducción

UPMdroid es una aplicación para dispositivos móviles con el sistema operativo Android.

Dicha aplicación nace como ejemplo de la metodología WSM detallada en el Bloque 2 y como resultado a la necesidad de los alumnos de centralizar las consultas que se realizan a través de las diferentes plataformas que ofrece la Universidad Politécnica de Madrid.

Está orientada a los alumnos de la Universidad Politécnica de Madrid y su principal funcionalidad, como ya se ha mencionado, es la de centralizar en una sola aplicación las diferentes plataformas que ofrece la UPM: Intranet y Politécnica Virtual. Su objetivo es que el alumno pueda consultar las calificaciones de las asignaturas en las que se encuentre matriculado y consultar su horario de clases tan sólo introduciendo sus datos de autenticación y pulsando un solo botón.

UPMdroid se basa en la metodología WSM (Web Scraping Methodology) de manera que para poder obtener la información relevante para el alumno, accede a las diferentes plataformas y mediante la técnica de web scraping obtiene los datos solicitados. Es una aplicación del lado del cliente que usa la infraestructura que ofrecen los servidores de la Universidad Politécnica de Madrid.

Durante la fase 1 de esta metodología se ha accedido mediante una de las herramientas de análisis, mencionadas en la fase 1 del bloque 2, al código HTML de las plataformas de las que se va a extraer la información y se ha determinado cual es la información que se quiere obtener:

3.1 Introducción

Información relevante en la plataforma de Intranet:

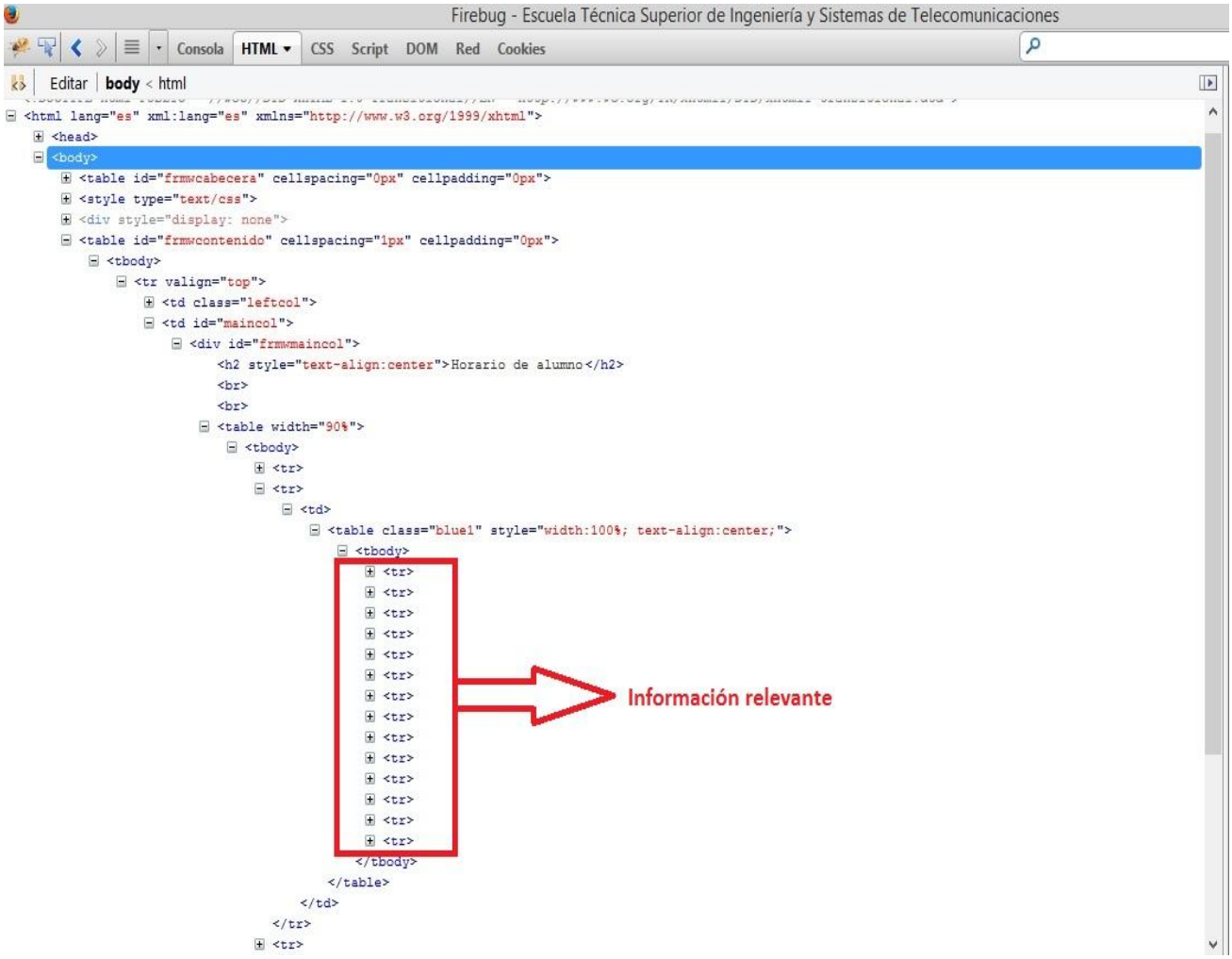

Figura 20. ***Extracto del código HTML de la Web de Intranet de la ETSIST.***

3.1 Introducción

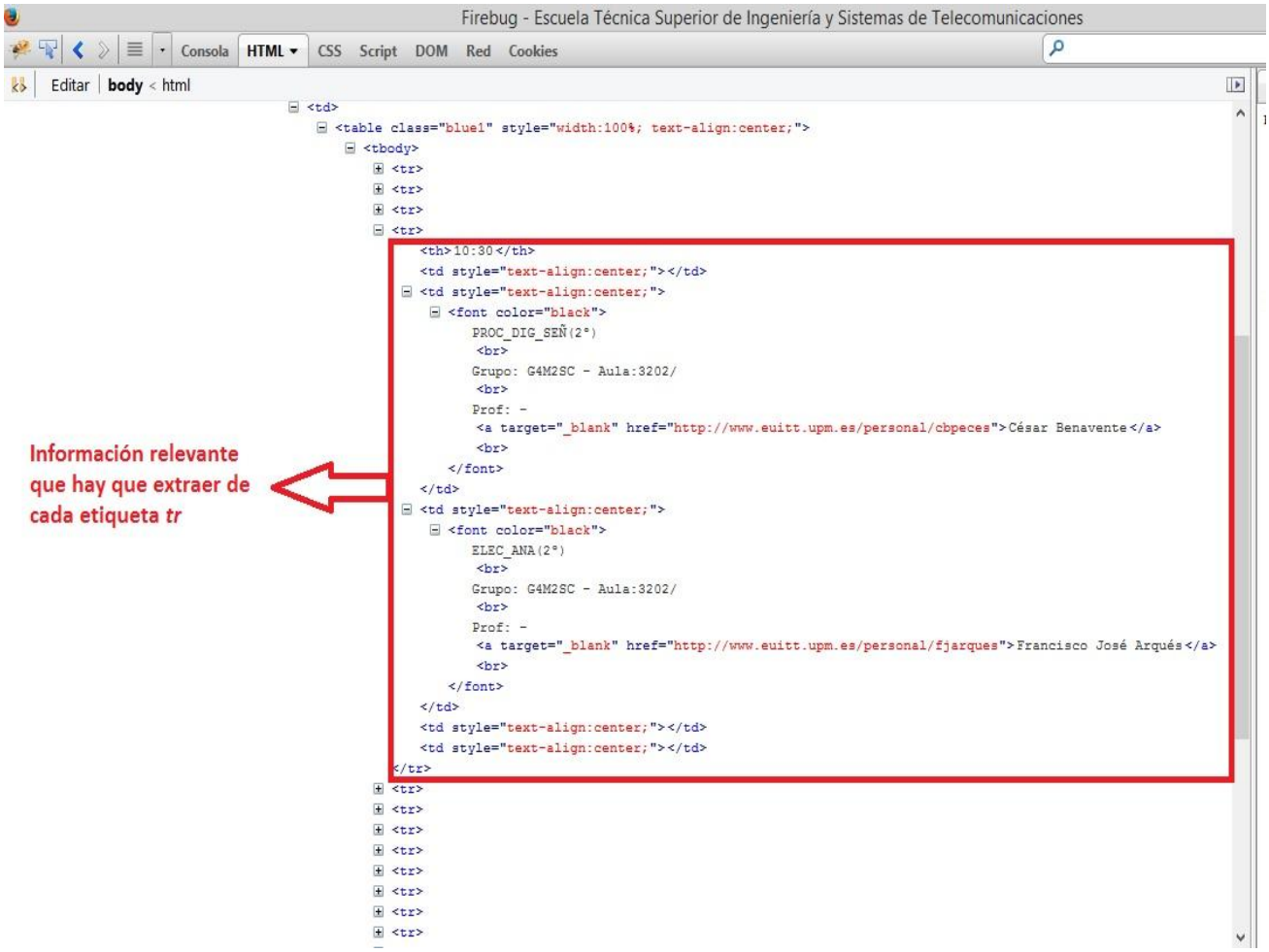

Figura 21. ***Extracto del código HTML de la Web de Intranet de la ETSIST.***

3.1 Introducción

Información relevante en la plataforma Politécnica Virtual:

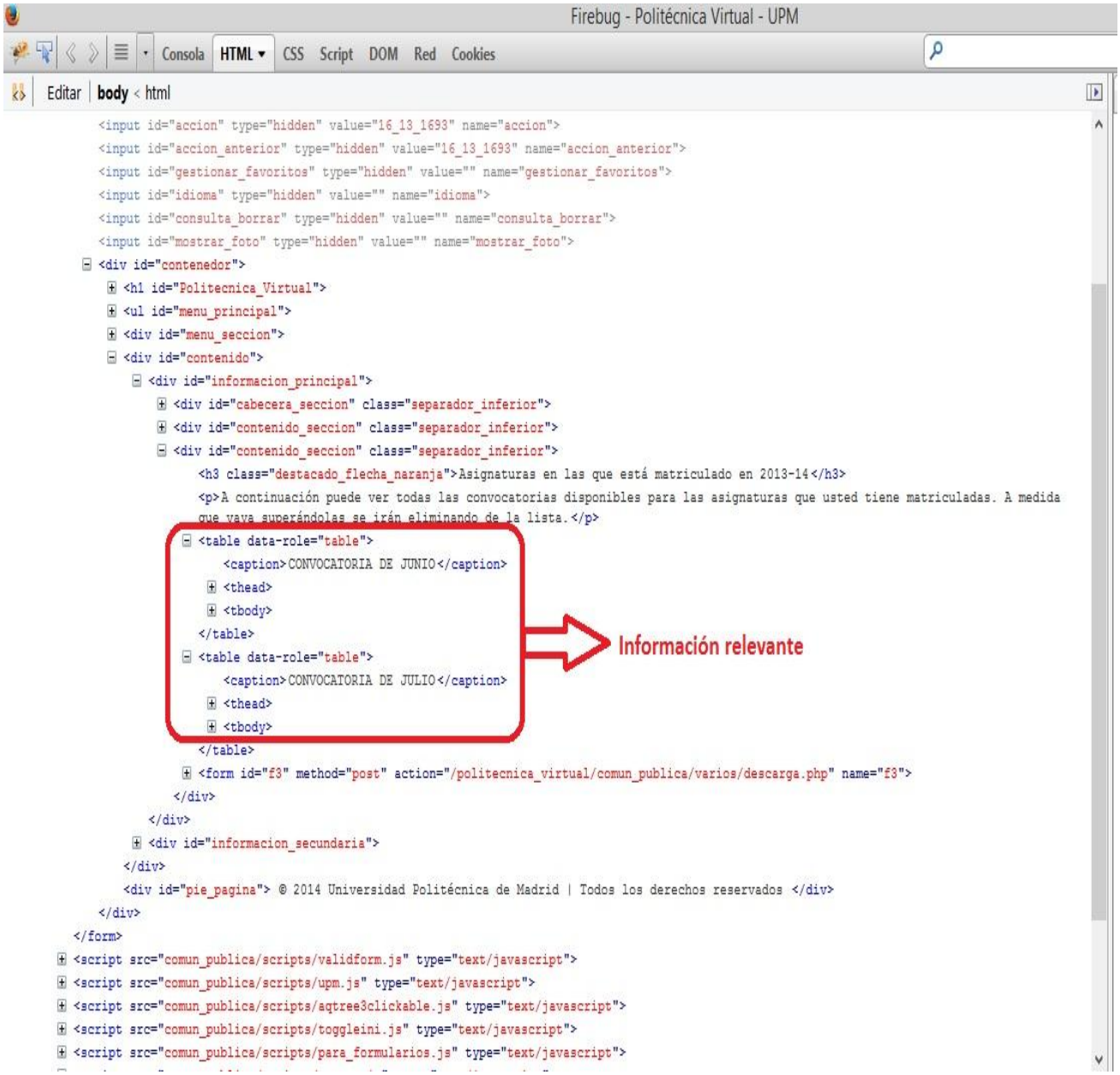

Figura 22. ***Extracto del código HTML de la Web de Politécnica Virtual de la UPM.***

3.1 Introducción

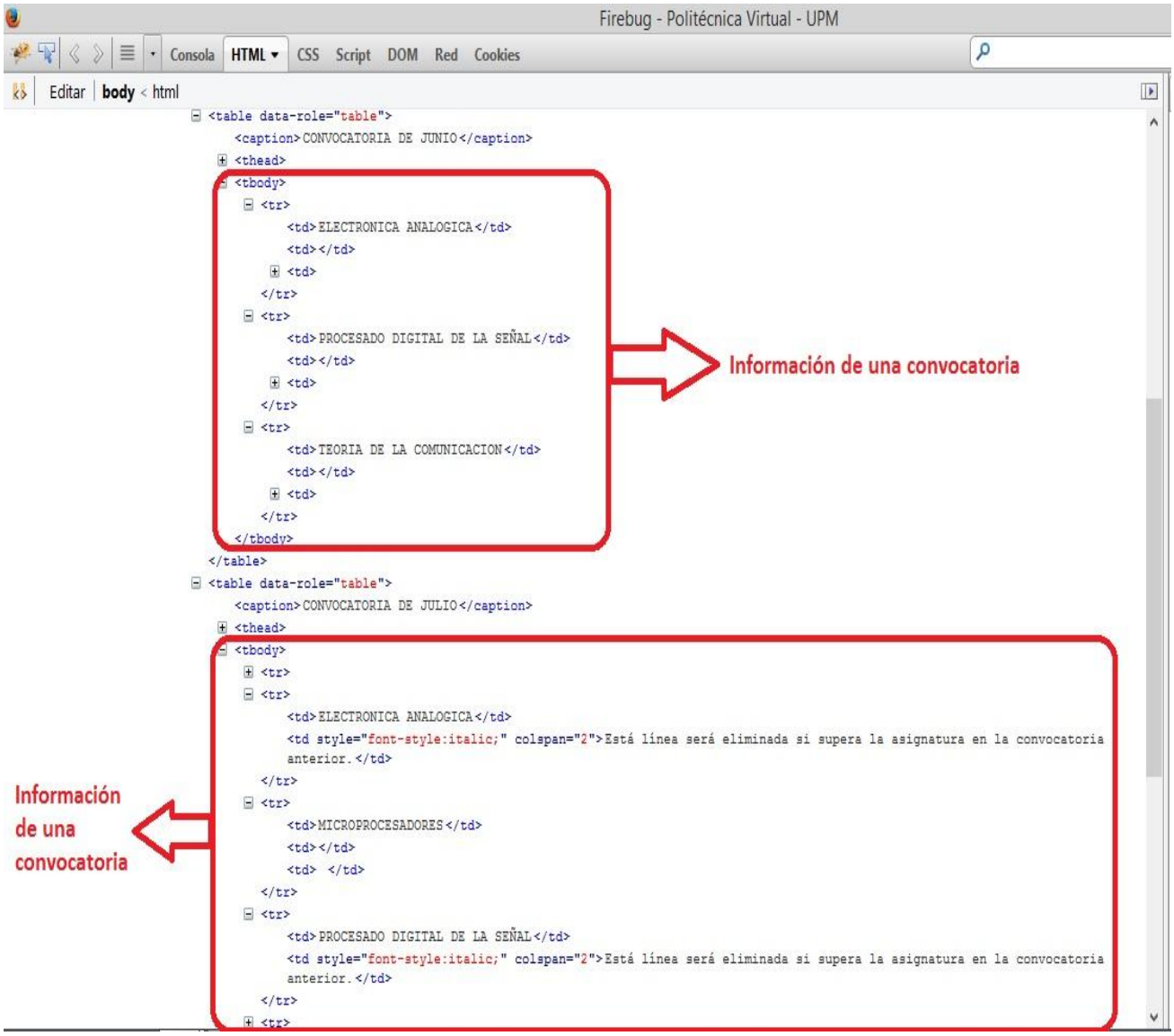

Figura 23. ***Extracto del código HTML de la Web de Politécnica Virtual de la UPM.***

3.1 Introducción

Así pues, la aplicación estará enfocada a obtener esta información y presentarla al usuario.

Se extraerá la información correspondiente al horario de clases de Intranet y la información correspondiente a las calificaciones de exámenes de Politécnica Virtual.

Además se tendrá especial cuidado a la hora de tratar con los datos de sesión: usuario, contraseña, cookies, etc. ya que juegan un papel crucial si se quiere obtener esta información.

3.2 Análisis de requisitos

Puesto que se va a implementar una aplicación para Android no se puede empezar a programar sin establecer una serie de requisitos previos que el producto final tiene que cumplir. Además, existe una serie de requisitos fundamentales sin los cuales no se podría empezar a implementar la aplicación.

Generalmente, en esta etapa de diseño se concierta una cita entre el analista y el cliente para establecer los requisitos que tiene que cumplir el producto final para satisfacer las necesidades del cliente.

Existen diversos mecanismos que permiten la detección de especificación de requisitos que un sistema software debe satisfacer. Según el estándar IEEE 610.12-1990 se define un requisito como una condición o capacidad que un sistema software debe alcanzar para satisfacer un contrato, especificación o cualquier otro documento formal que se imponga (necesidades del cliente).

Así pues, se pueden definir los requisitos en: **funcionales y no funcionales**, donde los requisitos funcionales tratan de describir la forma en la que el usuario va a utilizar el sistema y los requisitos no funcionales especifican las propiedades del sistema que tienen que ver con el rendimiento, velocidad, uso de memoria, plataforma, etc.

Los requisitos no funcionales son más difíciles de detectar, y por ello son evaluados subjetivamente.

3.2.1 Requisitos Funcionales

Autenticación:

R1. *Al realizar una consulta sobre cualquier plataforma de la UPM, si es la primera vez, se le pedirá al usuario que se autentique.*

R2. *El usuario se identificará introduciendo el nombre de usuario y el password que tenga en la plataforma en la que quiera realizar la consulta.*

R3. *Dado que las dos plataformas utilizan el mismo nombre de usuario y password, estos quedarán almacenados después de realizar la primera consulta correctamente*

en cualquiera de ellas. De esta forma el usuario no tendrá que volver a introducirlos para posteriores consultas.

R4. *Si la autenticación no ha sido correcta se volverá a la página principal informando al usuario que dicha autenticación ha sido fallida.*

R5. *Debe haber una opción en la página principal que muestre los datos de autenticación almacenados y ofrezca la posibilidad de cambiarlos o borrarlos.*

Generales:

R6. Debe haber una opción de ayuda en la página principal que describa el comportamiento y la manera en que se utiliza la aplicación.

R7. Las operaciones de consulta no podrán realizarse en el hilo principal de la aplicación, habilitando uno secundario para dichas operaciones y cerrándolo al terminarla.

3.2.2 Requisitos no Funcionales

R1. *La aplicación debe tener una interfaz gráfica sencilla e intuitiva.*

R2. *Los usuarios interactuarán con la aplicación mediante un dispositivo móvil que incorpore el sistema operativo Android (Smartphone, Tablet, ordenador, etc.) y soporte conexión a Internet.*

R3. *La aplicación estará creada para soportar todas las versiones de Android desde la versión 2.2 Froyo (API 8) hasta la versión 4.4 (API 19) ambas incluidas.*

R4. *El acceso a las diferentes plataformas se realizará vía Internet con conexiones seguras (SSL y HTTPS).*

R5. *Para poder implementarla será necesario tener algún entorno de programación Java (Eclipse, NetBeans, Android Studio, etc.), así como tener instalado la máquina virtual de Java (JRE) y Android Software Developement Kit (SDK).*

R6. *Las pruebas de funcionamiento se podrán realizar en un dispositivo virtual con Android (AVD o Genymotion) ó en un Smartphone o Tablet con Android conectada por USB al equipo donde se esté desarrollando la aplicación.*

3.2.3 Conclusiones

Con esta serie de requisitos se puede resumir todo en los siguientes aspectos a tener en cuenta:

- ✓ **Fácil de usar.** Se debe promover la utilización de la aplicación y para ello hay que lograr que la aplicación no resulte muy compleja de usar, de esta manera, los usuarios no mostrarán su rechazo.
- ✓ **Interfaz sencilla e intuitiva.** La interfaz debe ser sencilla, es decir, clara, concisa y sin demasiados elementos visuales ya que la pantalla de los dispositivos está limitada en cuanto a dimensiones se refiere. De esta manera se evitarán ralentizaciones innecesarias en la carga de la aplicación.
- ✓ **Útil para el usuario.** La aplicación será diseñada teniendo especial cuidado en lo que a funcionalidad se refiere ya que lo que se busca es dar una correcta respuesta a una necesidad del usuario y, por tanto, además de sencilla y fácil tiene que cumplir el objetivo principal por el que es creada.
- ✓ **Robustez.** Es necesario que la aplicación sea robusta manejando los distintos eventos que puedan surgir y dándoles respuesta lo más rápidamente posible.

3.3 Entorno

En el diseño de la aplicación se ha tenido en cuenta el entorno del sistema teniendo especial cuidado en que se pueda utilizar la aplicación independientemente del lugar en el que se encuentre el usuario. El concepto “always on” (siempre conectado) que proporciona 4G, UMTS y GPRS con toda la infraestructura que posee cada operador de telefonía móvil es un factor importante para las aplicaciones móviles que posibilitan el uso de las mismas en cualquier momento. Por supuesto, cabe resaltar que el usuario también puede usar la aplicación cuando esté conectado a una red Wi-Fi.

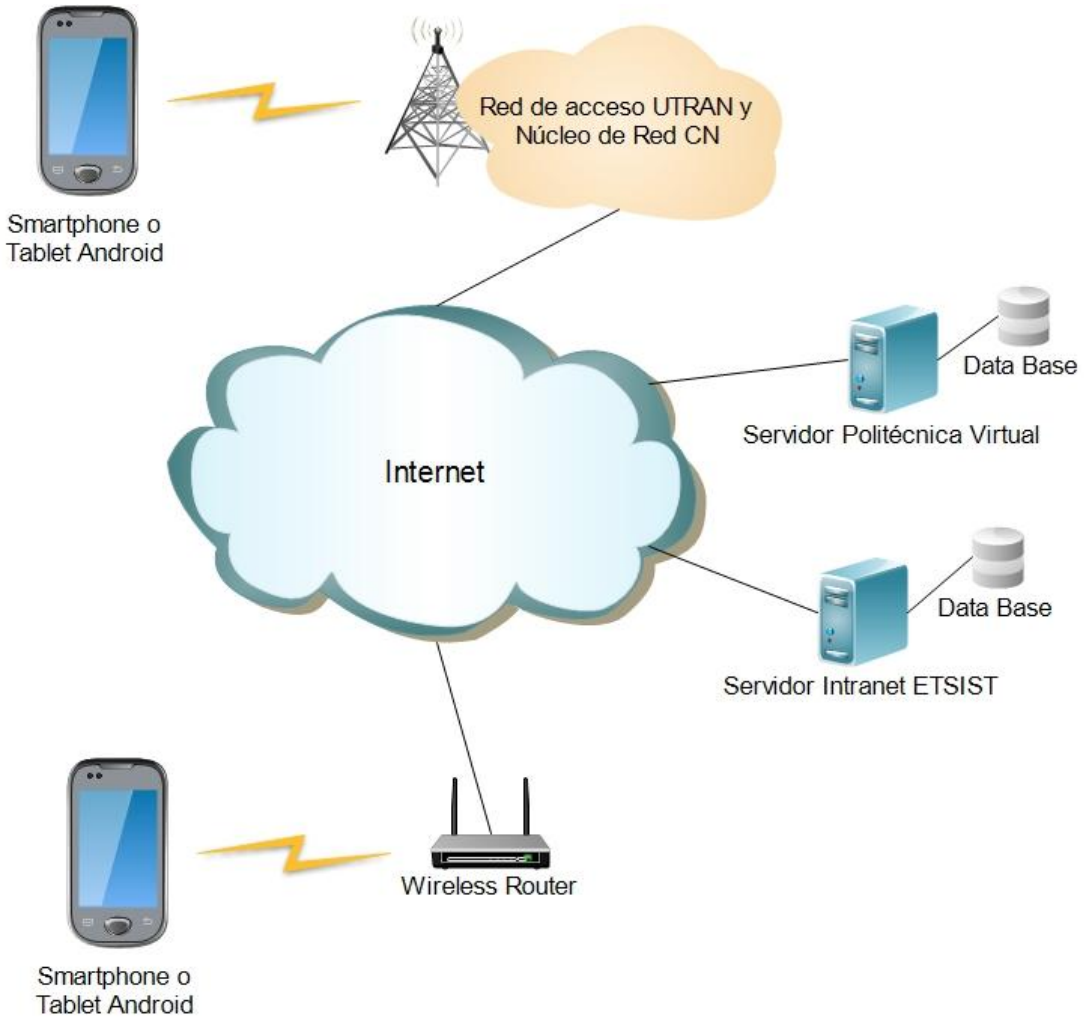

Figura 24. ***Entorno del sistema.***

Como se observa en la figura, existen dos posibles escenarios en los que utilizar la aplicación:

3.3 Entorno

- **Uso de la Red de Telefonía Móvil**. Este dominio está íntegramente implementado por las operadoras de telefonía móvil. El terminal móvil se ocupará de conectarse a la estación Base que se encuentre en su zona y después, internamente, realizarán los procedimientos necesarios con la arquitectura propia de las operadoras de telefonía móvil. Únicamente hay que tener en cuenta que estas tecnologías permiten el acceso a Internet desde casi todo el territorio nacional, dependiendo de la cobertura que cada operadora brinde a sus clientes.
- **Uso de Router Wi-Fi**. Mediante un Router Wireless con conexión a Internet se podrá conectar la aplicación al servidor web de destino que corresponda. Este tipo de escenario puede darse en un entorno doméstico, en el que el router inalámbrico está gestionado por el propio usuario, o bien en un entorno público el cual escapa a su control y gestión.

3.4 Diagrama de casos de uso

Antes de empezar a implementar una aplicación es importante realizar un diseño pormenorizado, intentando dar solución a todos los problemas que puedan surgir. De esta manera se evitan posibles reestructuraciones con todo lo que ello supone (peor resultado al esperado, retraso en la implementación y plazos de entrega, etc.), contratiempos que se pueden eludir si se realiza un diseño óptimo.

3.4.1 Casos de uso general

En la figura siguiente se muestra el caso de uso general de todo el sistema.

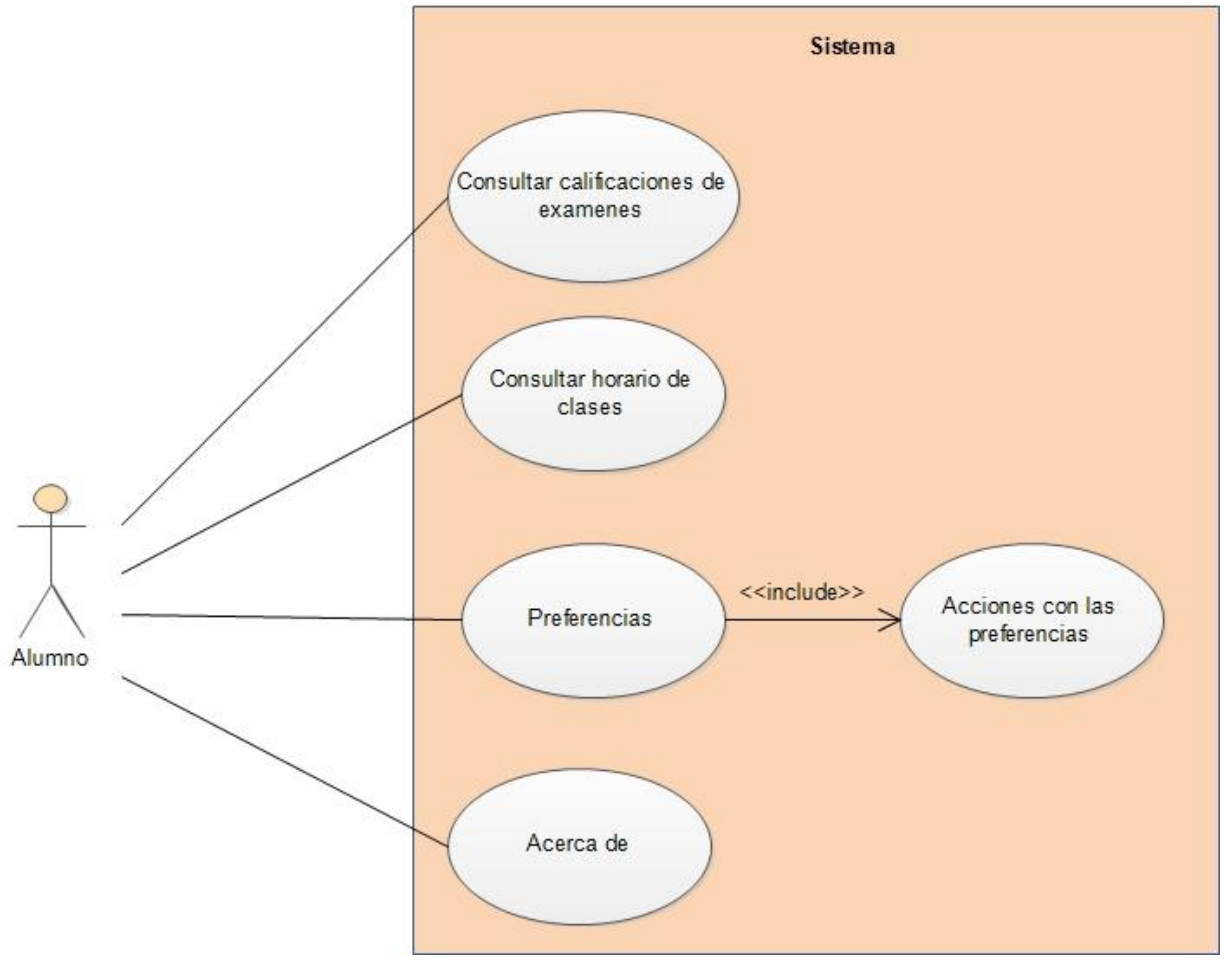

Figura 25. ***Caso de uso general de la aplicación***

El Actor es el alumno que es el que interactuará con el sistema con su dispositivo móvil. El alumno, al acceder al sistema, dispondrá de cuatro posibles casos de uso. Mediante los

casos de *Consulta* se obtendrán los datos que solicite el alumno. Estos casos se detallarán a continuación.

Cuando realice el caso de uso de *Preferencias* el alumno interactuará con el sistema mediante el caso de uso *Acciones* con el cual podrá visualizar, modificar y/o borrar sus datos de autenticación almacenados, así como, introducirlos si es que no hay datos almacenados.

Con el caso de uso *Acerca de* obtendrá una breve explicación del funcionamiento básico de la aplicación.

3.4.1.1 Caso de uso Consultar calificaciones de exámenes

A continuación se detalla el caso de uso para consultar las calificaciones de exámenes.

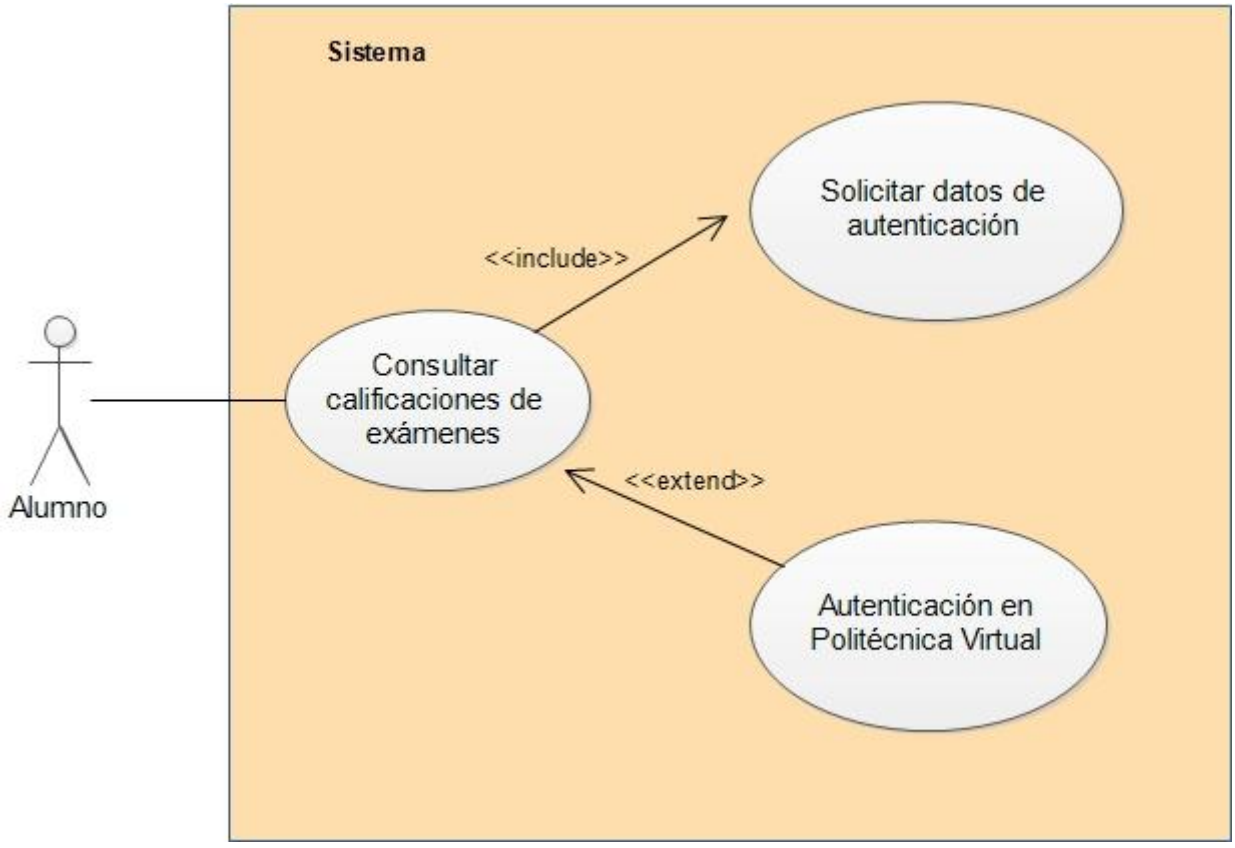

Figura 26. ***Caso de uso para consultar calificaciones de exámenes.***

Cuando el alumno realiza este caso de uso, si es la primera vez y no ha introducido antes sus datos de autenticación, se le solicitan. Una vez introducidos se realiza la autenticación en la plataforma Politécnica Virtual. Si la autenticación es correcta y no se ha realizado previamente alguno de los otros casos de uso de consulta, se almacenan sus datos de

autenticación y ya no se piden en las siguientes veces, para cualquier caso de consulta, a no ser que sean borrados desde el caso de uso *Preferencias*.

3.4.1.2 Caso de uso Consultar horario de clases

A continuación se detalla el caso de uso para consultar las calificaciones de exámenes.

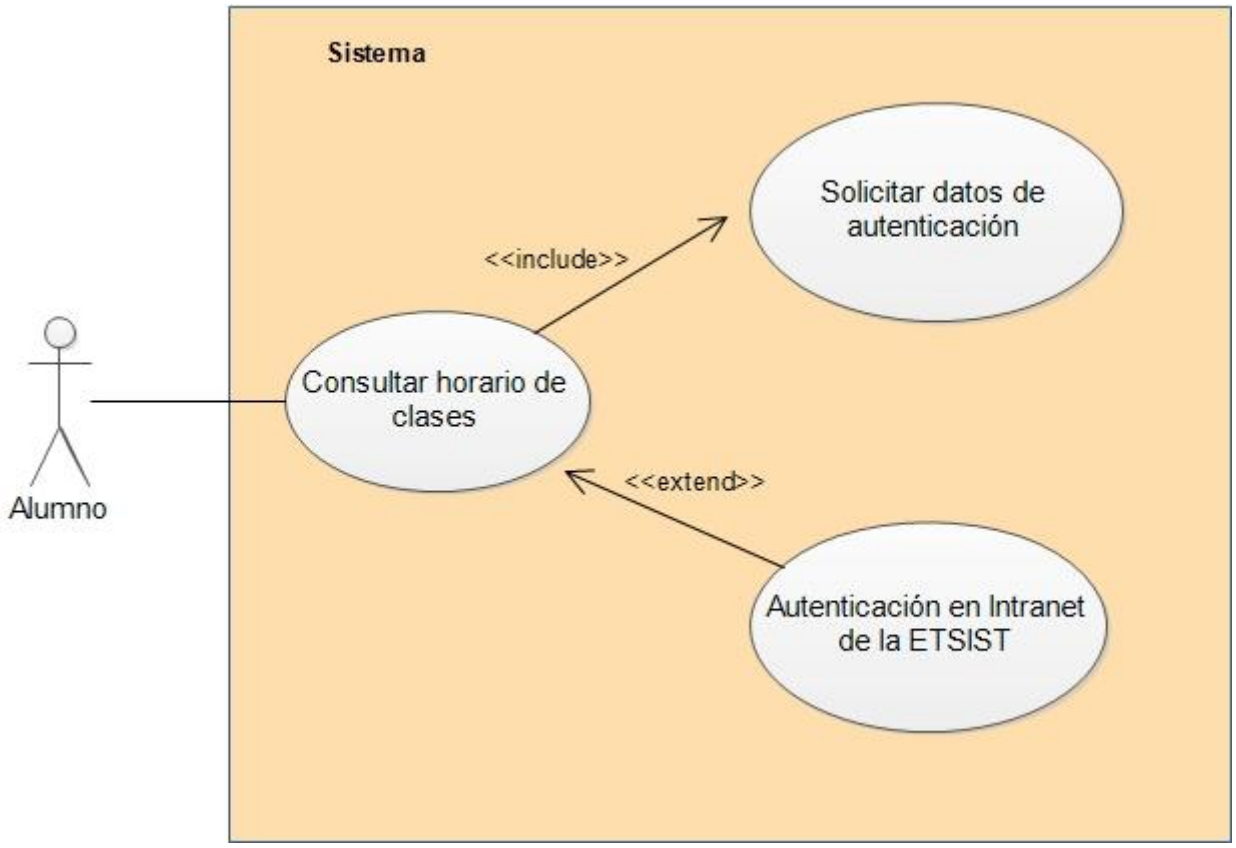

Figura 27. ***Caso de uso para consultar el horario de clases.***

Cuando el alumno realiza este caso de uso, si es la primera vez y no ha introducido antes sus datos de autenticación, se le solicitan. Una vez introducidos se realiza la autenticación en la plataforma de Intranet de la ETSIST. Si la autenticación es correcta y no se ha realizado previamente alguno de los otros casos de uso de consulta, se almacenan sus datos de autenticación y ya no se piden en las siguientes veces, para cualquier caso de consulta, a no ser que sean borrados desde el caso de uso *Preferencias*.

3.5 Diagramas de actividad

El comportamiento del sistema es un factor muy importante en el diseño de la aplicación ya que se quiere que sea un sistema robusto y de respuestas lo más rápidamente posible a los eventos que vayan apareciendo durante su ejecución.

A continuación se detallará el comportamiento del sistema para cada caso de uso.

3.5.1 Comportamiento para el caso de uso Consultar calificaciones de exámenes

La consulta de las calificaciones de exámenes se realizará sobre la plataforma Politécnica Virtual que proporciona la UPM, para ello será necesaria la previa autenticación del usuario que desea consultarlas. La aplicación solicita los datos al usuario en el caso en que no se encuentren almacenados previamente y realiza la autenticación en el servidor. Acto seguido busca y obtiene los datos solicitados y si son los correctos, almacena los datos de autenticación si no lo estaban ya y presenta la información al usuario.

A continuación se esquematiza el comportamiento de la aplicación para esta petición.

3.5 Diagramas de actividad

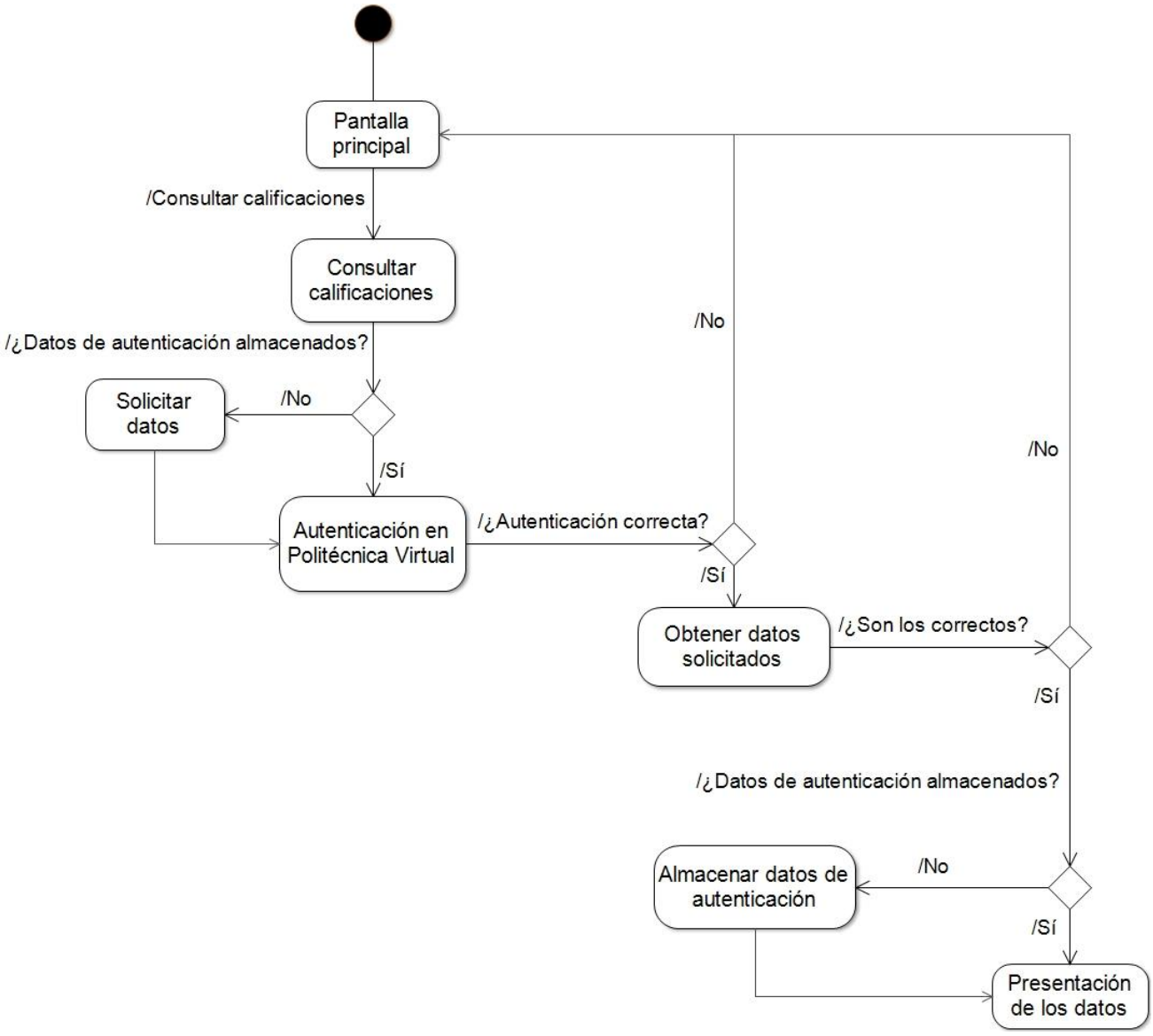

Figura 28. ***Diagrama de actividad para la consulta de calificaciones.***

3.5.2 Comportamiento para el caso de uso Consultar horario de clases

La consulta de las calificaciones de exámenes se realizará sobre la plataforma Intranet de la ETSIST que proporciona la UPM, para ello será necesaria la previa autenticación del usuario que desea consultar su horario de clases. La aplicación solicita los datos al usuario en el caso en que no se encuentren almacenados previamente y realiza la autenticación en el servidor. Acto seguido busca y obtiene los datos solicitados y si son los correctos, almacena los datos de autenticación si no lo estaban ya y presenta la información al usuario.

A continuación se esquematiza el comportamiento de la aplicación para esta petición.

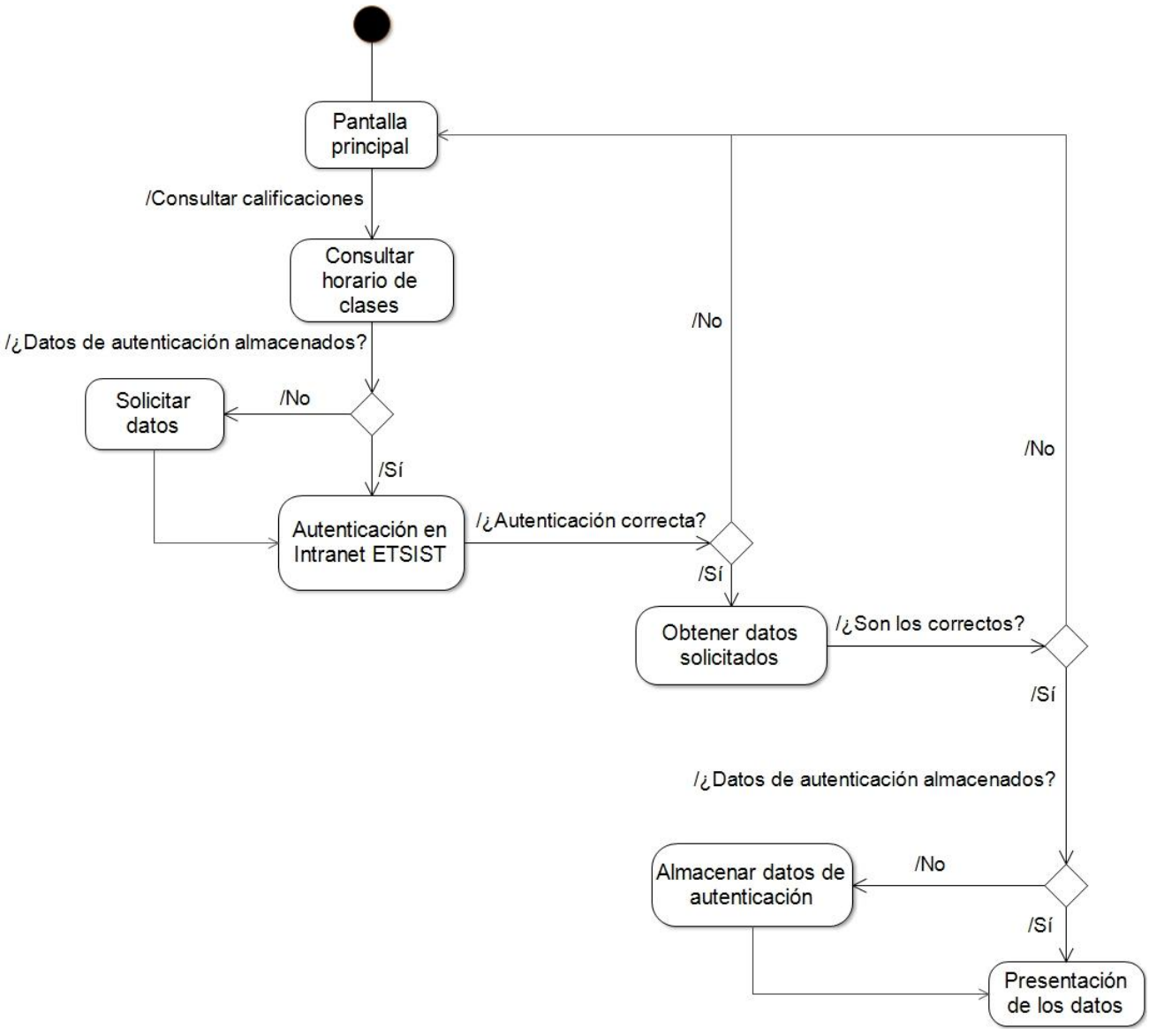

Figura 29. ***Diagrama de actividad para la consulta del horario de clases.***

3.6 Diagrama de clases

En este apartado se desarrollará el diagrama de clases general de la aplicación. Dada la gran importancia que adquiere la fase diseño de software, conviene aclarar en todo lo posible este aspecto ya que de esta manera se simplifica el posterior desarrollo de la aplicación y facilita la comprensión de su estructura tanto para otros desarrolladores como para el propio cliente.

3.6 Diagrama de clases

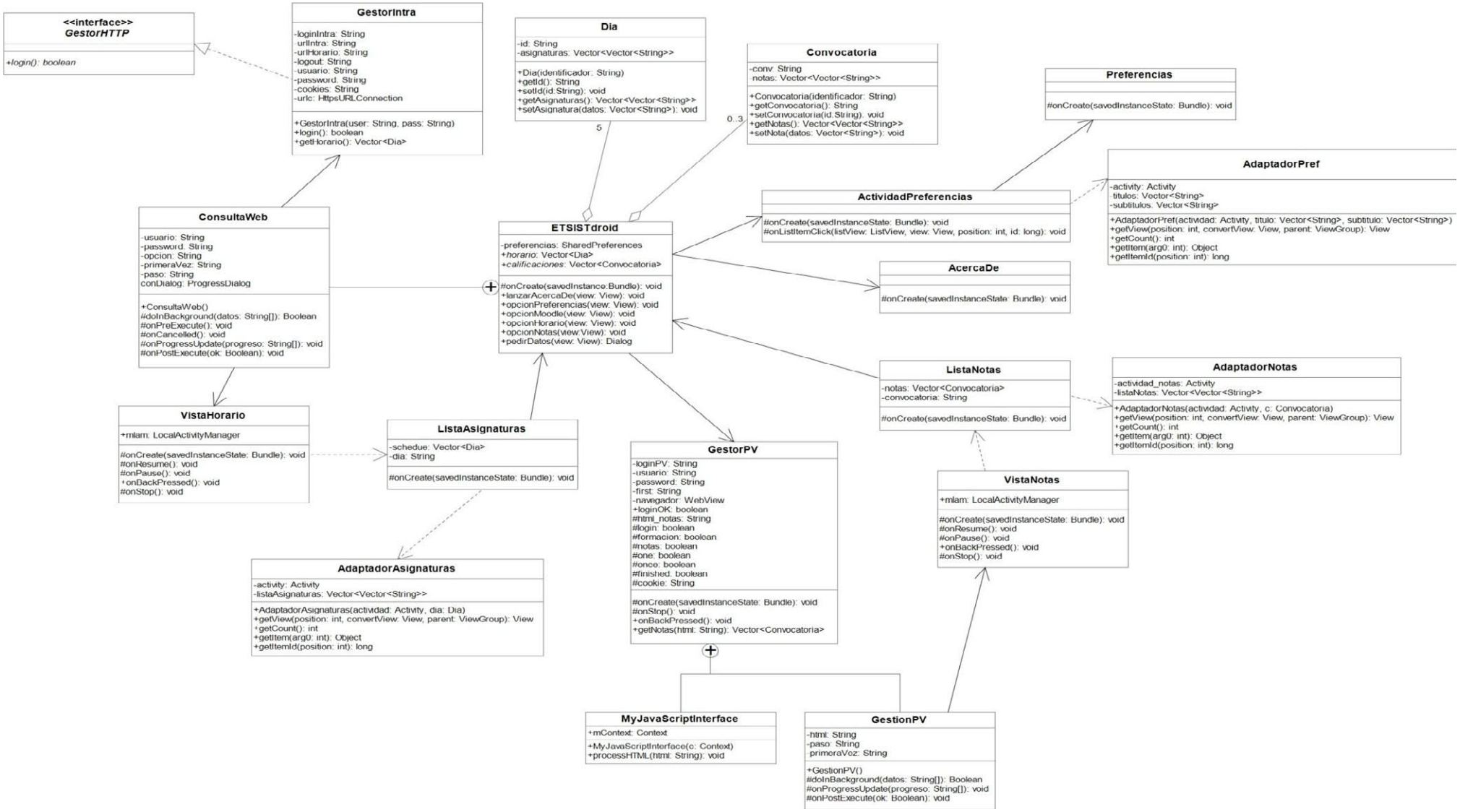

Figura 30. Diagrama de clases

3.6 Diagrama de clases

Como se puede observar en la figura, la clase *UPMdroid* se convierte en el eje de la aplicación, desde la que se podrá acceder a las diferentes opciones existentes, es, por tanto, la clase principal de la aplicación. Existen, además, clases como *GestorPV, VistaHorario, VistaNotas,* que compondrán, ayudados de otras clases, las distintas "vistas" que tendrá la aplicación.

La clase *UPMdroid* posee dos partes claramente diferenciadas. Por un lado se encuentran los atributos y por otro los métodos. En los atributos se encuentran las variables que serán usadas por esta clase. La variable *preferencias* será utilizada cuando sea necesario almacenar los datos de autenticación del usuario. Las variables *horario* y *calificaciones* serán las que contengan almacenados los datos relativos al horario de las clases y a las calificaciones de los exámenes respectivamente, además estarán declaradas como variables globales ya que es necesario que la clase que se encarga de presentar los datos al usuario, tenga acceso a los datos de estas variables. En la parte correspondiente a los métodos figurarán aquellos que contemplen los eventos a realizar para cada opción; además, la clase *UPMdroid* tendrá una clase interna que será la encargada de realizar y ejecutar, en un hilo secundario, las operaciones necesarias para la opción de consulta del horario de clases.

3.7 Diagramas de secuencia

Los diagramas de secuencia describen la secuencia de sucesos que transcurren en la ejecución concreta de un sistema, define, además, detalles de implementación del mismo.

El diagrama contiene los objetos que intervienen y la interacción que llevan a cabo entre ellos, mostrando el flujo del sistema. Generalmente se representa un diagrama de secuencia para cada caso de uso.

A continuación se detallan los diagramas de secuencia para los dos casos de uso más importantes del sistema.

3.7 Diagramas de secuencia

Diagrama de secuencia para la consulta de horarios de clases

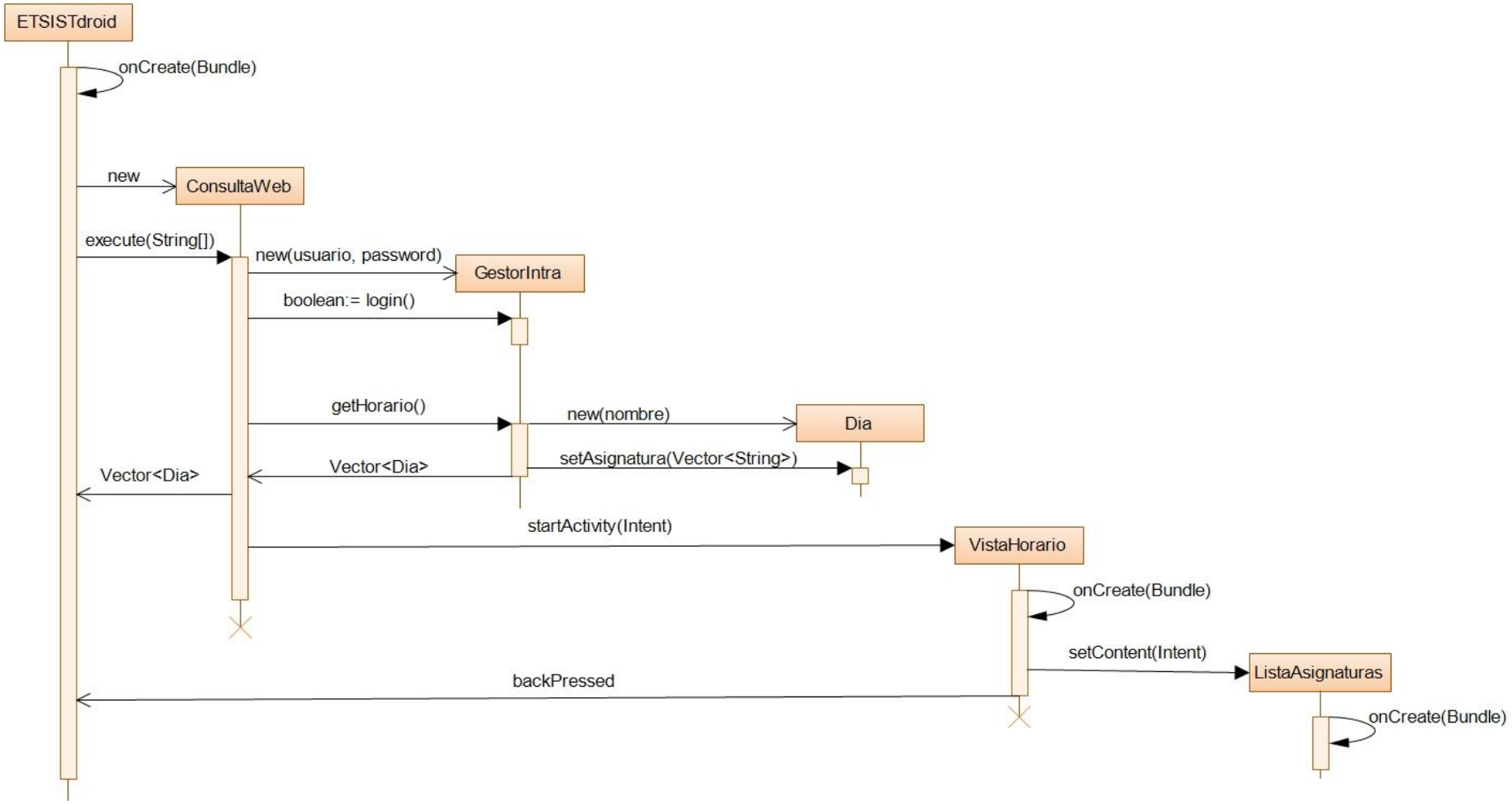

Figura 31. ***Diagrama de secuencia de consulta de horario de clases.***

3.7 Diagramas de secuencia

Como se observa en la figura anterior, el objeto principal *ETSISTdroid* crea *ConsultaWeb* y llama al método *execute*. Mediante esta llamada se consigue que tanto la búsqueda, la obtención y la extracción de la información se realicen en un hilo secundario de manera que no afecte al hilo principal de la aplicación. Una vez que es llamado el método *startActivity* desde *ConsultaWeb* el resto de operaciones son ejecutadas en el hilo principal.

ETSISTdroid y *VistaHorario* son vistas de la aplicación; una es la vista principal y la otra la vista donde se visualiza el horario. *ListaAsignaturas* es una vista, asociada a *VistaHorario,* donde se muestra el horario de clases de un día, por tanto, *VistaHorario* contendrá cinco *ListaAsignaturas*, una por cada día. El método *setContent* será llamado una vez por cada *ListaAsignaturas*.

Si se pulsa *atrás* en *VistaHorario* se volverá a la vista principal y se destruirá esa vista.

Diagrama de secuencia para la consulta de calificaciones

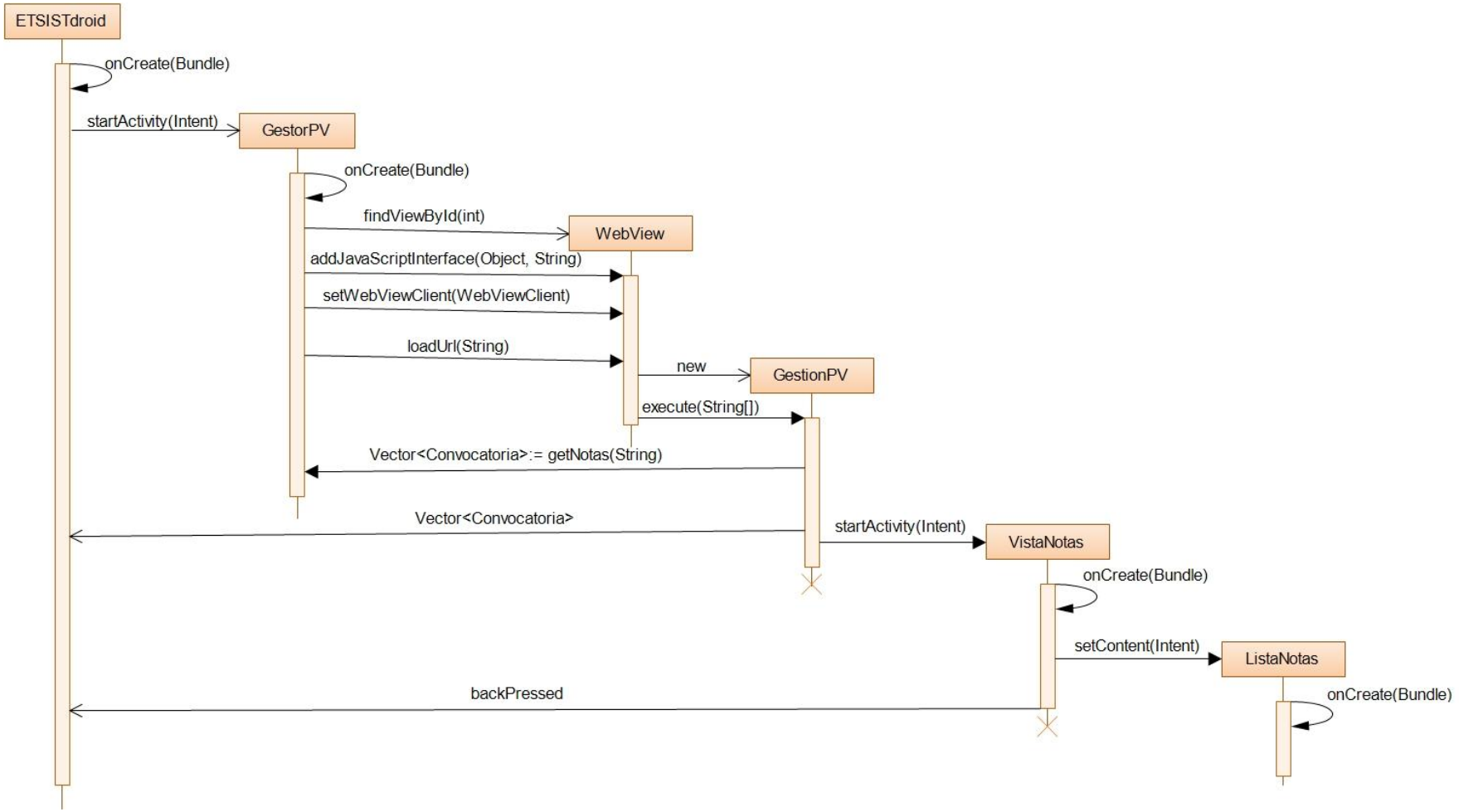

Figura 32. ***Diagrama de secuencia para la consulta de calificaciones.***

3.7 Diagramas de secuencia

Como se observa en la figura anterior, la vista principal *UPMdroid* crea *GestorPV* el cual se encarga del acceso, la obtención y la extracción de la información buscada. Dado que para el acceso y obtención de la información es necesaria la ejecución de JavaScript y la única opción que lo permite (*WebView*) no se puede ejecutar en segundo plano, se realiza en el hilo principal de la aplicación.

La extracción de la información se realiza en un hilo secundario cuando se llama al método *execute* desde el *WebView* y cuando se completa esta operación se llama a *VistaNotas* que se ejecutará ya en el hilo principal para mostrar las calificaciones.

Al igual que para el caso del horario de clases, *VistaNotas* tiene asociada la vista *ListaNotas* que mostrará la lista de asignaturas con sus calificaciones para una convocatoria. Por tanto, *VistaNotas* podrá tener hasta tres *ListaNotas* asociadas. El método *setContent* será llamado una vez por cada *ListaNotas*.

Si se pulsa *atrás* en *VistaNotas* se volverá a la vista principal y se destruirá esa vista.

3.8 Implementación

Como ya se vio en el apartado *1.8.2 ¿Qué se necesita para programar?* del bloque 1, es necesario cumplir una serie de requisitos antes de ponerse a implementar una aplicación para Android. Una vez cumplidos y ya que se ha hablado de utilizar la librería *Jsoup*[4], se tendrá que importar dicha librería al proyecto creado en Eclipse dado que es una librería externa que no pertenece al paquete principal de Java o Android.

Una vez hecho esto se ha procedido a la implementación de la aplicación desarrollando las diferentes clases en Java descritas en el diseño así como la interfaz de usuario, todo ello usando el entorno de desarrollo Eclipse.

Una vez concluida la implementación se han realizado una serie de pruebas de funcionamiento para comprobar el correcto funcionamiento de la aplicación.

3.8.1 Prueba de la aplicación sobre la plataforma Intranet:

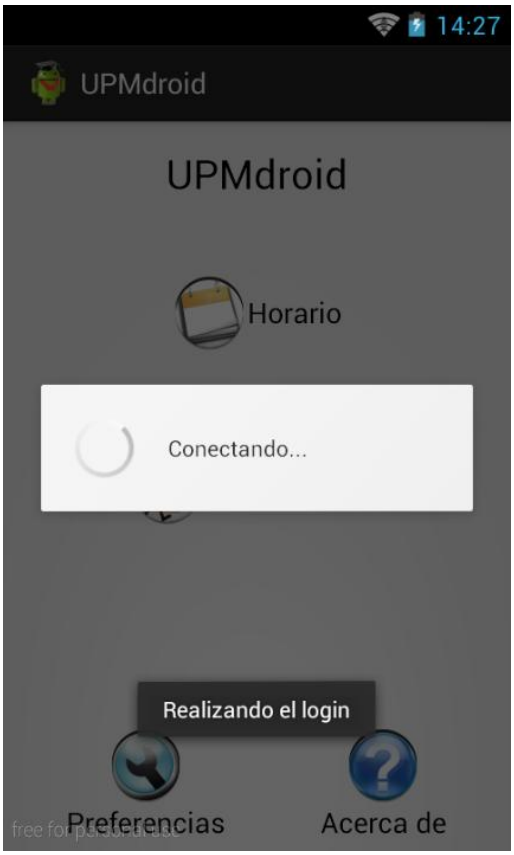

Figura 33. ***Primer paso.***

[4] Se puede descargar Jsoup desde la siguiente web: http://jsoup.org/download

3.8 Implementación

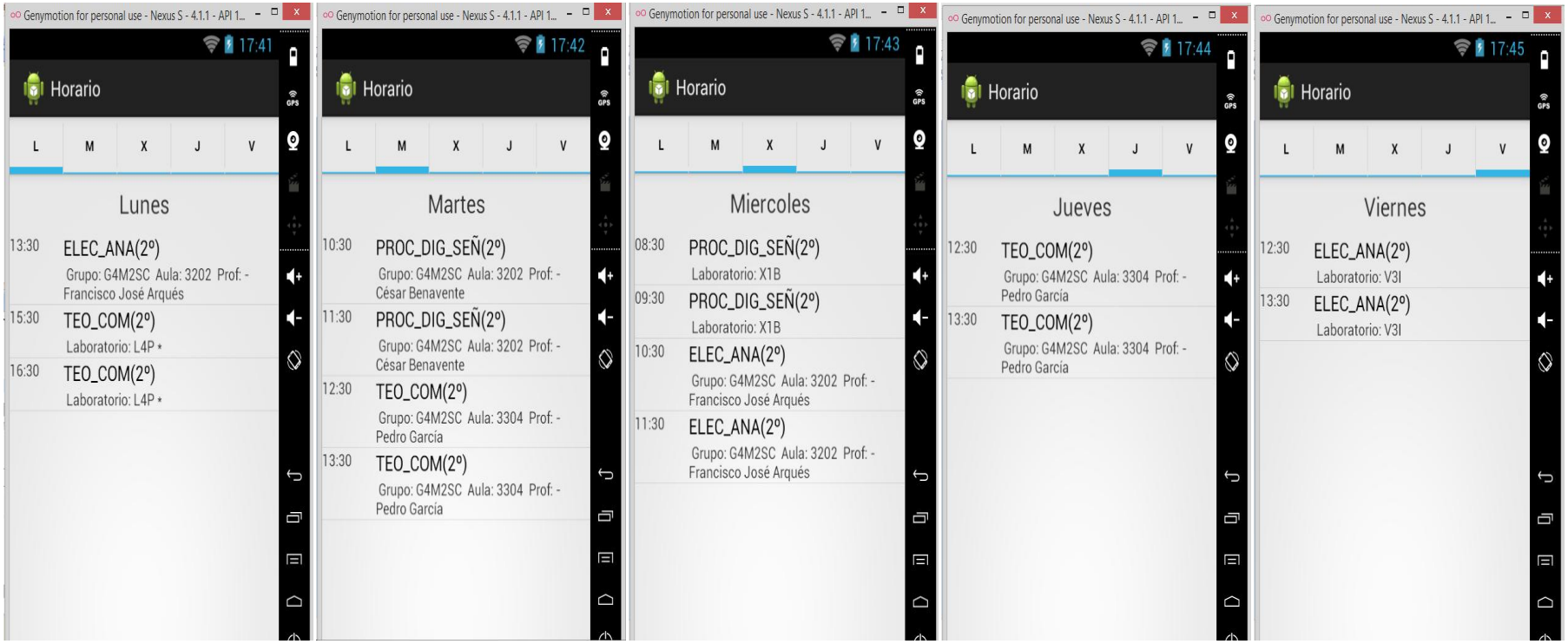

Figura 34. ***Resultado final.***

Se ha dividido la prueba en dos pasos: en el primero se establece la conexión, se obtiene la información y se extrae la relevante; en el segundo se presenta la información extraída al usuario.

3.8.2 Prueba de funcionamiento sobre la plataforma Politécnica Virtual:

Figura 35. ***Primer paso.***

3.8 Implementación

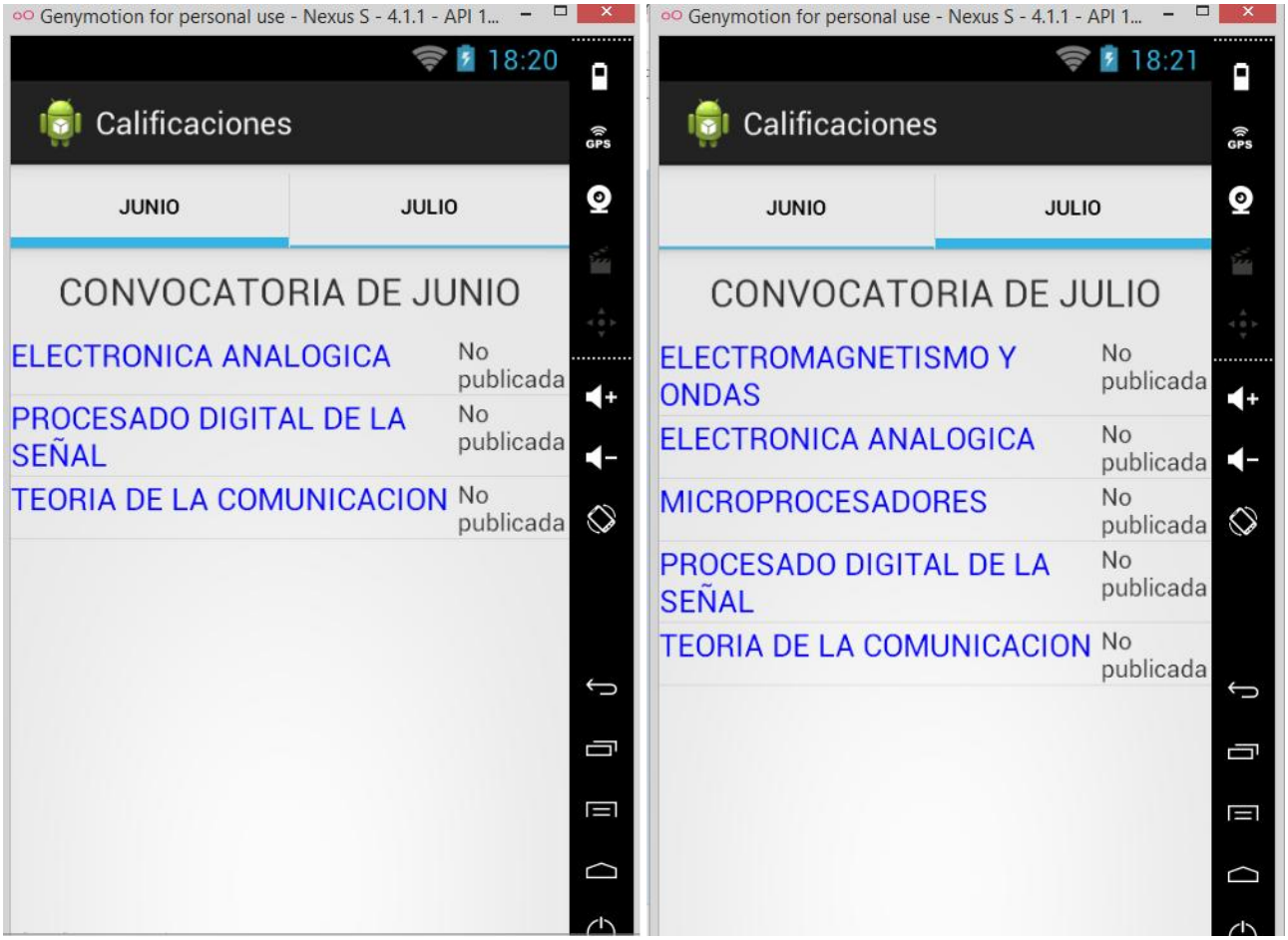

Figura 36. ***Resultado final.***

Al igual que en el caso de consulta del horario de clases, se ha dividido la prueba en dos pasos: en el primero se establece la conexión, se obtiene la información y se extrae la relevante y en el segundo se presenta esta información al usuario.

4. Conclusiones

Con la realización de este proyecto se ha pretendido básicamente proponer una posible solución al problema de integración de la técnica de web scraping con el sistema operativo Android al carecer de APIs que realicen esta tarea. Como se tenía cierto interés en crear una aplicación móvil en la que se pudieran consultar las tareas más importantes durante el curso, surgió también la necesidad de definir una metodología para ello.

En este proyecto se han cumplido varios objetivos. Los principales son la adquisición de conocimientos sobre la técnica de Web Scraping, así como el desarrollo de una estrategia a seguir para afrontar y solucionar el problema. Además se han adquirido una serie de conocimientos secundarios sin los cuales habría sido imposible poder llevar a la práctica el proyecto. Los más importantes son la adquisición de conocimientos sobre Android así como el ciclo de vida y la implementación de aplicaciones para su uso en este sistema operativo.

No hay que olvidar que también se han cumplido los objetivos para los que cobró vida este proyecto: Definir una metodología para realizar Web Scraping en Android así como crear una aplicación totalmente operativa que resultara tremendamente útil para los alumnos de esta escuela y que se basara en la metodología detallada anteriormente.

Se ha conseguido integrar dos plataformas web ofrecidas por la UPM en una sola aplicación que permita consultar lo más importante con tan sólo pulsar un botón.

A lo largo de la realización de este proyecto se han encontrado algunas dificultades no previstas; la más compleja ha sido la extracción de la información de webs que usaran scripts, sin embargo se encontró una solución al problema cuando las opciones estaban prácticamente agotadas, lo que se tradujo en un gran grado de satisfacción y en la sensación de que nunca hay que rendirse.

4.1 Trabajos futuros

Este proyecto ha cumplido su objetivo, sin embargo la constante evolución de Android así como de las metodologías para desarrollo de software dejan ventanas abiertas a la mejora tanto de la metodología planteada como de la aplicación implementada.

Para la parte de la aplicación se podría introducir nuevas funcionalidades como solicitar revisión en el caso de la consulta de las calificaciones en Politécnica Virtual, introducir nuevas consultas sobre la plataforma Moodle como visualizar fechas de entrega de prácticas de laboratorio o poner un mensaje en el foro de la asignatura matriculada así como mejorar la interfaz de usuario.

Además si Android lo permitiera en el futuro, la parte de Web Scraping sobre páginas que contengan scripts se podría volver a implementar. Además, en caso de que el API *HtmlUnit* permitiera su integración en Android, se podría implementar a aplicación usando esta librería en vez de *HttpURLConnection* o *WebView* dado que sería más sencillo de realizar.

5. Anexo I. Manual de usuario

Este manual pretende ser una sencilla guía que permita al usuario entender toda la funcionalidad de la aplicación así como los pasos a seguir para una correcta utilización.

Para la elaboración de esta guía se ha utilizado el emulador Genymotion con la versión 4.1.1 de Android.

Antes de utilizar la aplicación es necesario instalarla, para ello bastará con realizar la instalación del archivo *UPMdroid.apk*. Una vez instalada ya se podrá acceder a ella.

5.1 Acceder a la aplicación

Cuando esté instalada la aplicación simplemente con pulsar sobre el icono de la misma se accederá a la pantalla principal.

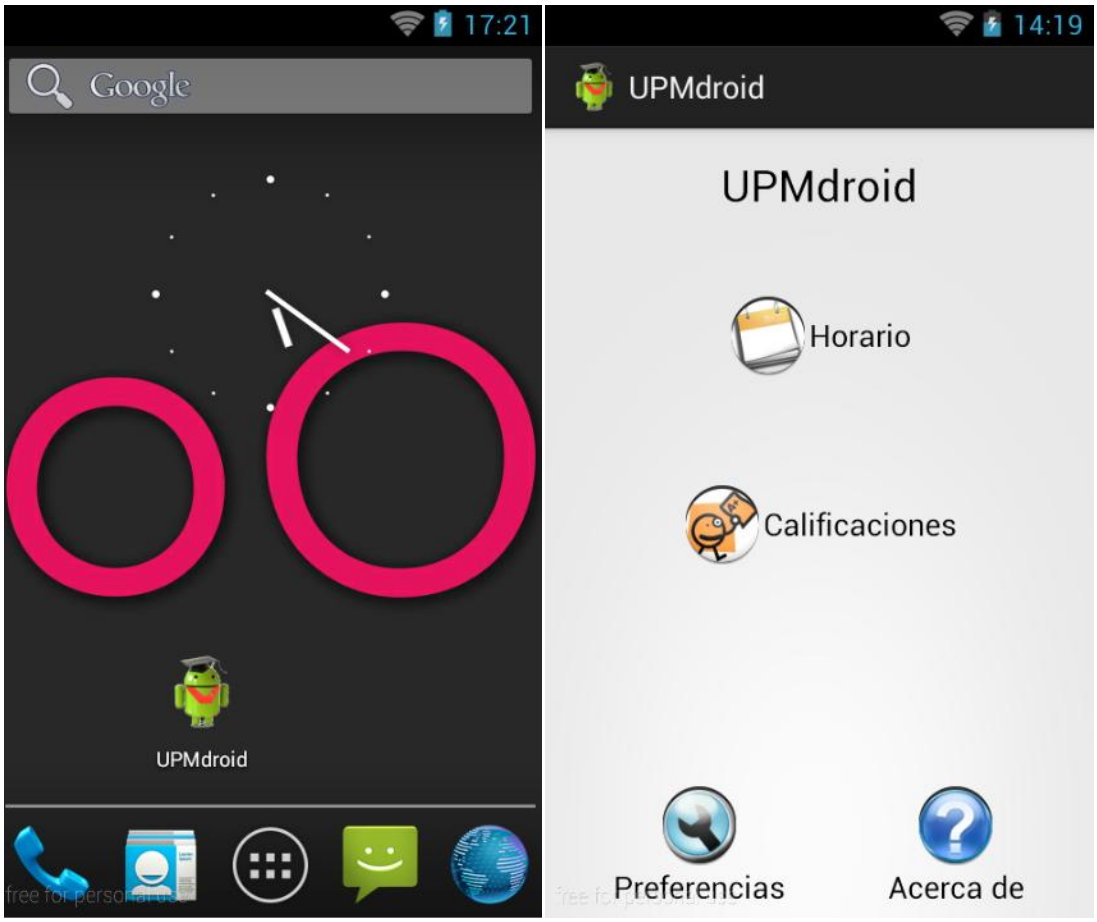

Figura 37. ***Escritorio del dispositivo y pantalla principal de la aplicación.***

En la pantalla principal de la aplicación se muestran cuatro posibilidades diferentes; entre ellas se destacan en el centro de la pantalla las dos más importantes que son *Horario* y *Calificaciones*. Las otras dos, *Preferencias* y *Acerca de*, se han añadido en la parte baja de la pantalla. Para salir de la aplicación, se deberá pulsar el botón *atrás*  estando en la pantalla principal, o bien, el botón *home*  en cualquier pantalla de la aplicación.

5.2 Realizar una consulta

Para realizar una consulta bastará con pulsar sobre cualquiera de las dos opciones, *Horario o Calificaciones*, dependiendo que tipo de consulta se desee realizar. Independientemente de cuál de las dos opciones se escoja, si es la primera vez que se va a realizar una consulta, aparecerá un cuadro donde el alumno deberá introducir sus datos de autenticación para enviarlos al servidor donde se realice la consulta. Actualmente tanto la plataforma de Intranet de la UPM como la plataforma Politécnica Virtual operan con el mismo formato de usuario y contraseña: **<*usuario*>@alumnos.upm.es** y **contraseña**. Cabe destacar que el alumno deberá introducir el usuario **completo** incluyendo **@alumnos.upm.es** ya que el sistema no autocompleta el nombre de usuario, sin embargo, sí que comprueba que tenga ese formato.

Figura 38. ***Solicitud de datos de autenticación.***

Una vez introducidos los datos de autenticación y tras unos segundos, dependiendo de la conexión, el alumno podrá ver el resultado de su consulta si ha introducido sus datos correctamente, en caso contrario, volverá a la pantalla principal de la aplicación informándole de que la autenticación en el servidor ha sido incorrecta.

Si se produce cualquier tipo de error de red o de conexión después de que el alumno introduzca sus datos, se volverá a la pantalla principal informando del problema aunque haya introducido correctamente sus datos de autenticación.

Una vez en la pantalla de visualización del resultado de la consulta, para volver a la pantalla principal bastará con que pulse el botón *atrás*  de su dispositivo móvil.

5.3 Preferencias

La opción *Preferencias* gestiona todo lo relacionado a los datos de autenticación del alumno. Si se pulsa sobre ella aparecerá la siguiente pantalla con estas tres opciones:

Figura 39. ***Pantalla principal de preferencias.***

Si se pulsa sobre la opción ***Modificar Preferencias*** aparecerá la siguiente pantalla:

Figura 40. ***Modificar preferencias.***

Desde esta pantalla se podrá cambiar el usuario y la contraseña pulsando sobre la opción deseada:

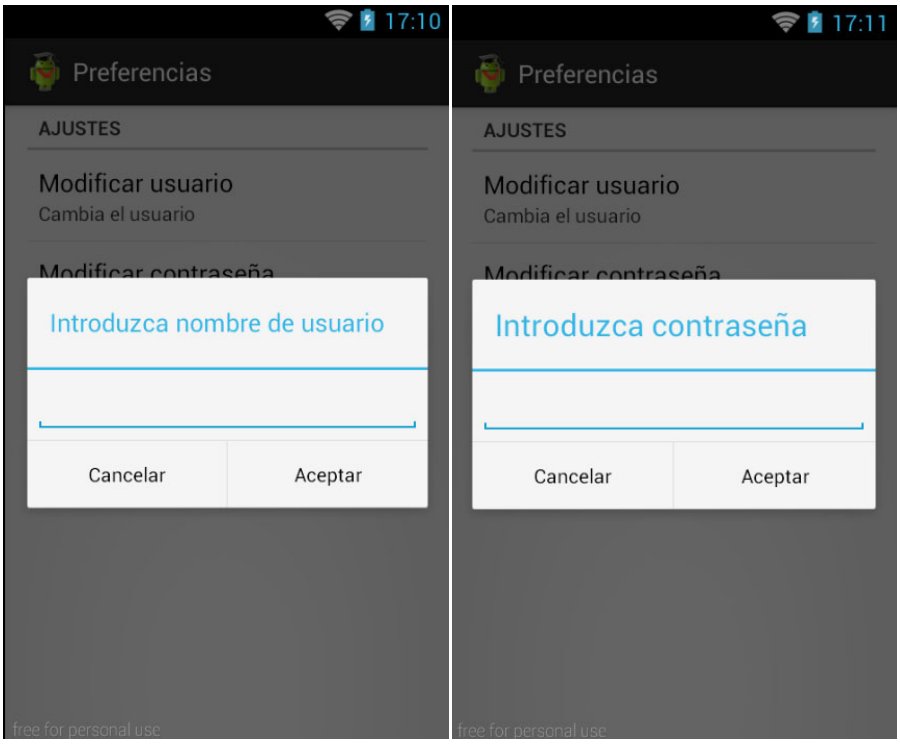

Figura 41. ***Modificar usuario y modificar contraseña***

Una vez modificadas las opciones deseadas bastará con pulsar atrás  para volver a la pantalla principal de las preferencias.

En la pantalla principal de las preferencias, si se pulsa sobre la opción ***Mostrar Preferencias*** aparecerá un cuadro en donde se podrá visualizar el nombre de usuario del alumno. La contraseña no se podrá visualizar por motivos de seguridad.

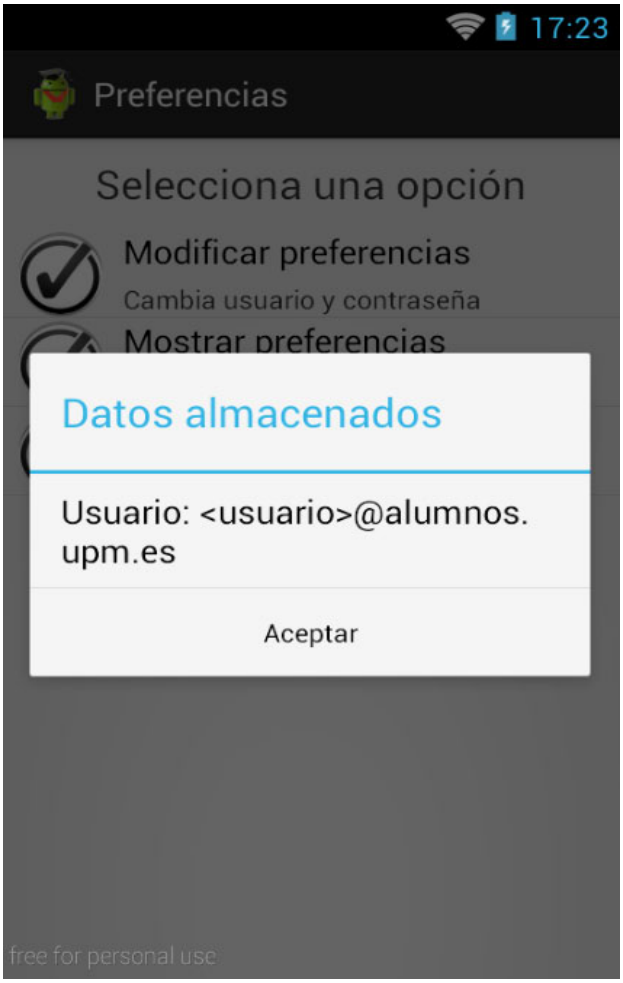

Figura 42. ***Mostrar preferencias.***

Asimismo, en la pantalla principal, si se pulsa sobre la opción ***Borrar Preferencias*** se procederá al borrado tanto del usuario como de la contraseña mostrando un aviso al final del borrado si se ha realizado correctamente.

5 Anexo I. Manual de usuario

Figura 43. ***Borrar preferencias.***

Pulsando *atrás*  en la pantalla principal de las preferencias se volverá a la pantalla principal de la aplicación.

5.4 Acerca de

La opción *Acerca de* muestra un cuadro con una pequeña ayuda para manejar la aplicación.

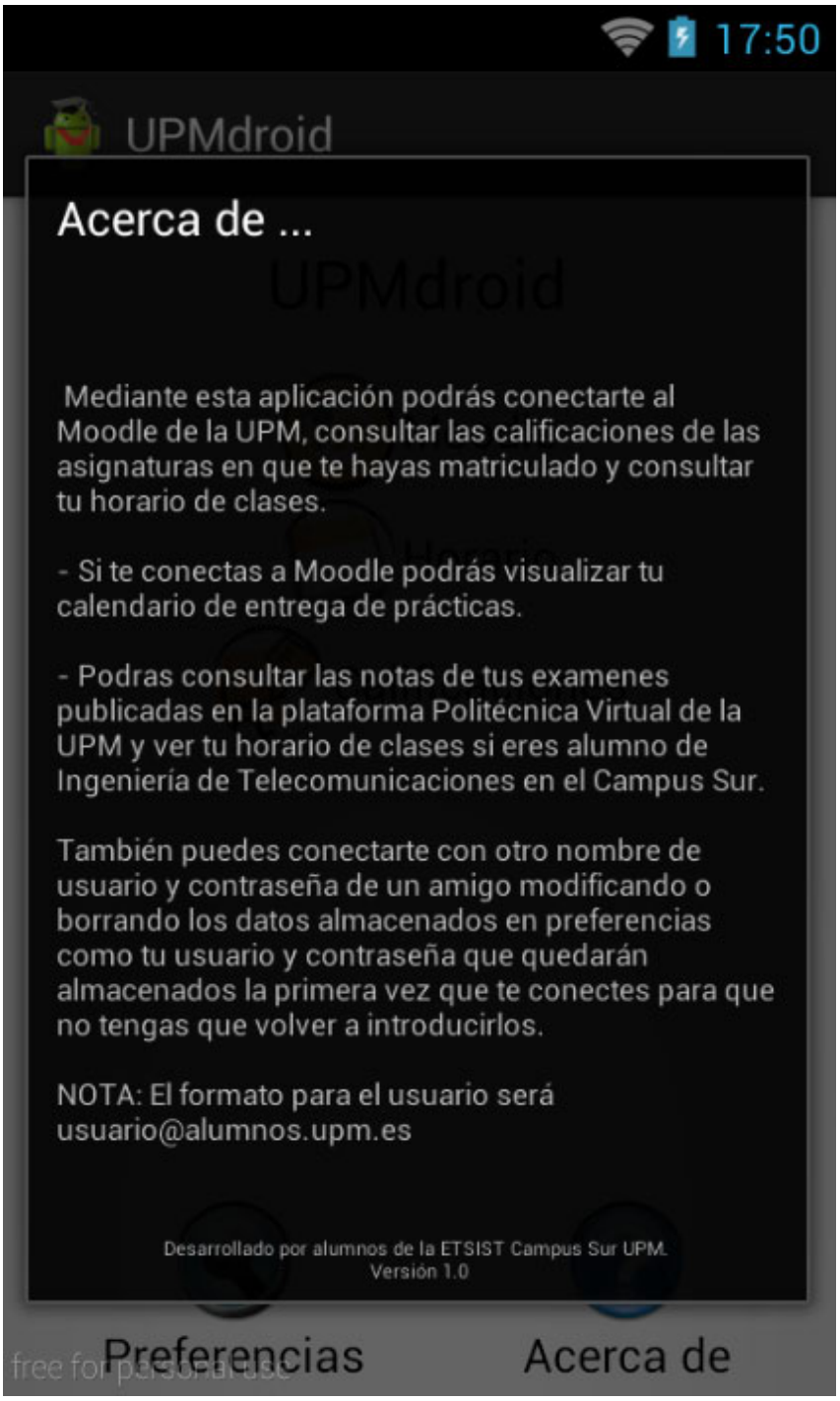

Figura 44. ***Opción Acerca de***

Para volver a la pantalla principal bastará con pulsar el botón *atrás*  del dispositivo móvil.

6. Glosario

- ✓ **Web Scraping** – Técnica búsqueda y extracción de una determinada información contenida en una página web.
- ✓ **Android** – Sistema operativo móvil desarrollado y distribuido por Google Inc.
- ✓ **Java** – Lenguaje de programación orientado a objetos desarrollado por Sun Microsystems y Oracle.
- ✓ **SGML** – Siglas de Standard Generalized Markup Language. Es un estándar para la organización y etiquetado de documentos.
- ✓ **HTML** – Lenguaje de etiquetado usado para programar el contenido de las páginas web. La inmensa mayoría de páginas web estructuran su contenido en un documento HTML.
- ✓ **CSS** – Lenguaje usado para definir la presentación de un documento HTML o XML.
- ✓ **XML** – Lenguaje de marcas extensible desarrollado por el W3C.
- ✓ **W3C** – Consorcio internacional que produce recomendaciones para la red informática global.
- ✓ **JavaScript** – Lenguaje de programación orientado a objetos utilizado principalmente en el lado del cliente e implementado como parte de un navegador web.
- ✓ **HTTP** – Protocolo del nivel de aplicación usado para el intercambio de peticiones y respuestas, entre cliente y servidor, por la red informática global.
- ✓ **DNS** – Siglas de Domain Name System. Es un sistema de nomenclatura jerárquica para equipos, servicios o cualquier recurso conectado a la red informática global.
- ✓ **IP** – Protocolo del nivel de red usado para el intercambio de datos bidireccional a lo largo de la red informática global.
- ✓ **Parsing** – Técnica para traspasar información de un modelo de datos a otro de manera que sea más sencillo su análisis.
- ✓ **API** – Interfaz de programación de aplicaciones. Conjunto de métodos y procedimientos que ofrece una librería para ser utilizado por otro software.
- ✓ **WSM** – Metodología para realizar Web Scraping, por aplicaciones para el sistema operativo Android.
- ✓ **Jsoup** – API para Java destinada a parsear documentos HTML desarrollada por Jonathan Hedley.
- ✓ **DOM** – Siglas de Document Object Model. Es un modelo de representación de datos presentes en documentos HTML y XML. Desarrollado por el W3C.

7. Bibliografía

[1] Roberto Augusto Cuenca Cuenca, *PFC: Android en el Hogar Digital usando UPnP y Redes Lonworks.,* Mayo 2013, UPM.

[2] Javier Galán Cardeñosa, *PFC: Aplicación móvil para control del Hogar Digital.,* Marzo 2013, UPM.

[3] John Duckett, *HTML & CSS: Design and Build Websites,* John Wiley & Sons. November 2011. ISBN: 978-1-118-00818-8. [Online]. http://proquestcombo.safaribooksonline.com/book/web-development/html/9781118206911.

[4] Wikipedia Foundation. Hypertext Transfer Protocol. [Online]. http://es.wikipedia.org/wiki/Hypertext_Transfer_Protocol.

[5] Elliotte Rusty Harold, *Processing XML with Java: A Guide to SAX, DOM, JDOM, JAXP, and TrAX*, Addison-Wesley Professional, November 2002. ISBN: 978-0-201-77186-2. [Online]. http://proquestcombo.safaribooksonline.com/book/xml/0201771861.

[6] Michael Schrenk, *Webbots, Spiders, and Screen Scrapers: A guide to Developing Internet Agents with PHP/CURL.* Ed. No Starch Press, 2nd Edition, March 15, 2012. ISBN: 978-1-59327-397-2. [Online]. http://proquestcombo.safaribooksonline.com/book/programming/php/9781593273972.

[7] Ken Bluttman; Wayne S. Freeze. *Access Data Analysis Cookbook.* O' Reilly Media, Inc., May 2007. ISBN: 978-0-596-10122-0. Chapter: *6.12. Scraping Web HTML.* [Online]. http://proquestcombo.safaribooksonline.com/book/databases-and-reporting-tools/0596101228/6dot-using-programming-to-manipulate-data/scraping_web_html_html?query=%28%28web+scraping%29%29#snippet.

[8] Ian F. Darwin, *Android Cookbook.* O' Reilly Media, Inc., April 2012. ISBN: 978-1-4493-8841-6. [Online]. http://proquestcombo.safaribooksonline.com/book/programming/android/9781449335786?bookview=overview.

[9] Wikipedia Foundation. Metodología de desarrollo de software. [Online]. http://es.wikipedia.org/wiki/Metodolog%C3%ADa_de_desarrollo_de_software .

7. Bibliografía

[10] Wikipedia Foundation. Programación extrema. [Online]. http://es.wikipedia.org/wiki/Programaci%C3%B3n_extrema.

[11] HtmlUnit. [Online]. http://htmlunit.sourceforge.net/ .

[12] Selenium WebDriver. [Online]. http://docs.seleniumhq.org/docs/03_webdriver.jsp .

[13] Jonathan Hedley. Jsoup: Java HTML parser. [Online]. http://jsoup.org/ .

[14] Pete Houston, *Instant Jsoup How-to.* Packt Publishing, June 2013. ISBN: 978-1-78216-800-3. [Online]. http://proquestcombo.safaribooksonline.com/book/programming/java/9781782167990/preface/pr06_html?query=%28%28web+scraping%29%29#snippet.

[15] Allen Holub's UML Quick Reference. [Online]. http://www.holub.com/goodies/uml/index.html .

[16] Sparx Systems, UML 2 Tutorial. [Online]. http://www.sparxsystems.com.au/resources/uml2_tutorial/ .

[17] Ryan Mitchell. *Instant Web Scraping with Java.* Packt Publishing, August 2013. ISBN: 978-1-84969-688-3. [Online]. http://proquestcombo.safaribooksonline.com/book/programming/java/9781849696883.

[18] Salvador Gómez Oliver, *Curso Programación Android*. [Online]. http://www.sgoliver.net/blog/?page_id=3011# .

[19] Universidad Politécnica de Valencia, *Fundamentos de Android.* [Online]. http://www.androidcurso.com/index.php/modulo-fundamentos/tutoriales-android-fundamentos .

[20] Android Developers. [Online]. http://developer.android.com/guide/index.html .

[21] Eclipse Foundation. Eclipse. [Online]. http://www.eclipse.org/ .

[22] Oracle Corporation. Class URLConnection. [Online]. http://docs.oracle.com/javase/6/docs/api/java/net/URLConnection.html#getHeaderFields%28%29 .

7. Bibliografía

[23] Stackoverflow. Android Web Scraping with a Headless Browser. [Online]. http://stackoverflow.com/questions/17399055/android-web-scraping-with-a-headless-browser.

[24] Stackoverflow. Fill fields in WebView automatically. [Online]. http://stackoverflow.com/questions/7961568/fill-fields-in-webview-automatically.