Gestor de contraseñas en un dispositivo móvil accesible por Bluetooth

Memoria del Proyecto de Fin de Carrera de Ingeniería en Informática

realizado por

Sergio Laguna García

y dirigido por

Helena Rifà Pous

Bellaterra, 18 de Setiembre de 2008.

Universitat Autònoma de Barcelona

etse
Escola Tècnica Superior d'Enginyeria

El sotasignat, **Helena Rifà Pous**

Professor/a de l'Escola Tècnica Superior d'Enginyeria de la UAB,

CERTIFICA:

Que el treball a què correspon aquesta memòria ha estat realitzat sota la seva direcció per en **Sergio Laguna García**

I per tal que consti firma la present.

Signat: **Helena Rifà Pous**

Bellaterra, 18 de Setembre de 2008

Agradecimientos

Este proyecto no habría sido posible sin la dirección de mi tutora de proyecto, Helena Rifà Pous.

No podía tampoco olvidarme del apoyo constante de mi hermano y mi madre, que han sabido animarme constantemente.

Finalmente mención especial para Melisa Pérez Zamora que ha sabido aguantarme durante este largo verano dándome su apoyo incondicional en todo momento.

Para ellos, **muchas gracias.**

Índice General

1. Introducción

El crecimiento exponencial de los usuarios y organizaciones conectadas a Internet (gran ejemplo de canal de comunicación no segura) hace que circule por la red información de todo tipo, desde noticias más o menos importantes, datos personales que no queremos que se conozcan públicamente y hasta la realización de gestiones económicas como podrían ser las transacciones bancarias, que requieren medidas específicas de seguridad que garanticen la confidencialidad, la integridad y la constatación del origen de los datos.

Este proyecto nace de la necesidad de dar un grado más de seguridad cuando nos encontramos delante de un formulario del tipo login/password mientras navegamos con uno de los navegadores de Mozilla, como podrían ser Firefox o ThunderBird.

1.1. Motivaciones

Una de las principales motivaciones que me han llevado a la realización de este proyecto ha sido el tema de la seguridad y la protección de datos, tema por el cual estoy muy interesado, pero no tengo mucha experiencia y estoy seguro que al acabarlo podré dar un grado más de seguridad a mis datos, lo que hace que me adentre en él con muchas ganas.

Otra motivación es la de trabajar con los navegadores de Mozilla, como el ThunderBird o el Firefox. Éste último en especial, que es sobre el cual se basa el proyecto y es el navegador que utilizo en la actualidad. Desde que salió al mercado, ha ofrecido a los usuarios una estabilidad notable y cuenta con muchísimas utilidades. Una de estas utilidades son las extensiones, que añaden más funcionalidad al navegador. En definitiva, me parece que el navegador Firefox es una más que buena alternativa a su principal competidor, el Internet Explorer de Microsoft.

Para finalizar, decir que este proyecto, lo podré aprovechar para mi mismo una vez esté finalizado. Se podrá continuar trabajando en él con las mejoras propuestas en el apartado mejoras futuras, dado que se tendrá mucho más tiempo para el desarrollo. Con esto, se podrá hacer una extensión más completa y acabar de dar más seguridad a los datos, sobre todo a mis transacciones bancarias, que todo sea dicho, son bastantes a lo largo del año.

1.2. Objetivos

El objetivo principal del proyecto es crear una extensión para el navegador Firefox, que sea capaz de capturar los formularios del tipo login/password, comunicarse con un dispositivo móvil, a través de Bluetooth, que servirá de gestor de contraseñas y rellenar los campos del formulario automáticamente con la contraseña proporcionada por el móvil. Estos objetivos los podríamos resumir en los siguientes puntos:

- Crear la parte de la interfaz gráfica.
- Crear un Midlet en el dispositivo móvil, que sea capaz de proporcionarnos la contraseña para un formulario conocido.
- Crear un componente XPCOM que se encargue de la comunicación Bluetooth con el dispositivo.
- Validar el formulario de login/password y rellenar los campos del formulario con la contraseña obtenida por el componente XPCOM.

1.3. Planificación

La planificación del proyecto se ha estructurado en varias fases que a continuación pasamos a resumir:

- **Fase 1 (Noviembre – Enero):** Se ha de llevar a cabo toda la recolecta de información. En lo referente al navegador, estaría el cómo crear una

extensión en Firefox, que lenguaje se ha de utilizar para la interfaz gráfica, como crear la conexión Bluetooth para comunicarnos con el dispositivo móvil y que herramientas tenemos para el desarrollo.

En cuanto al móvil, hay que buscar información sobre el lenguaje de programación Java para dispositivos móviles (J2ME) para crear sockets que nos permitan establecer la comunicación con el PC. También habría que ver qué herramientas de desarrollo tenemos, como podría ser NetBeans o Eclipse.

Al finalizar la recolecta de la primera fase, se tendrá que elaborar un estudio de viabilidad que será entregado el 14 de Enero.

- **Fase 2 (Febrero – 2 primeras semanas):** En las dos primeras semanas de febrero se creará el diseño del sistema y los casos de uso.

- **Fase 3 (Febrero – 2 segundas semanas):** Las dos semanas siguientes del mes de febrero se dedicarán a crear los módulos del navegador Firefox.

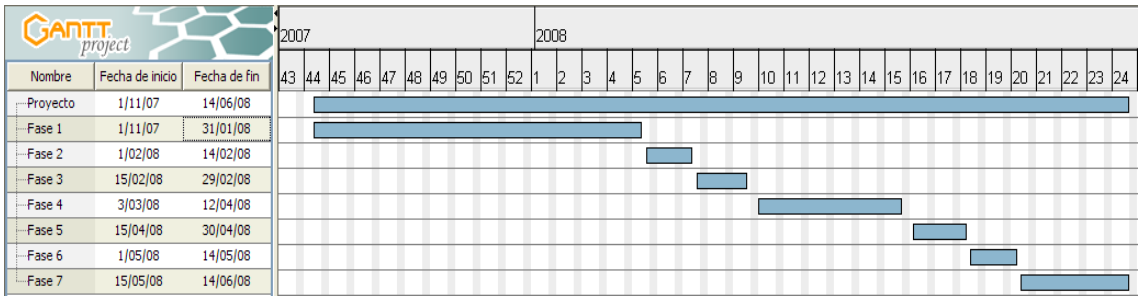
- **Fase 4 (Marzo – mediados Abril):** Las 6 semanas correspondientes al mes de Marzo y las 2 primeras de Abril, se hará la creación de un programa que sea capaz de realizar una comunicación Bluetooth basada en sockets y recibimiento de datos. También se creará la integración de dicho programa con el navegador Firefox. Dado que es un periodo de tiempo bastante largo, se avanzará, en la medida de lo posible, en la memoria del proyecto.

- **Fase 5 (mediados Abril – 2 segundas semanas):** En las 2 últimas semanas de Abril, se crearán los módulos del J2ME, es decir, la programación del MIDlet.

- **Fase 6 (Mayo – 2 primeras semanas):** Las 2 primeras semanas de Mayo se utilizarán para hacer la integración de todo junto y acabar la memoria.

- **Fase 7 (Mediados Mayo – Junio):** Las 2 últimas semanas de Mayo y las 2 primeras de Junio servirán para acabar de pulir la memoria ya que ésta será redactada mientras se va avanzando en la faena. También se acabará de pulir todo el programa. Finalmente se creará la presentación en PowerPoint del proyecto.

En la Figura 1 se observa un Diagrama de Gantt donde se representan las 7 fases explicadas anteriormente.



Nombre	Fecha de inicio	Fecha de fin
Proyecto	1/11/07	14/06/08
Fase 1	1/11/07	31/01/08
Fase 2	1/02/08	14/02/08
Fase 3	15/02/08	29/02/08
Fase 4	3/03/08	12/04/08
Fase 5	15/04/08	30/04/08
Fase 6	1/05/08	14/05/08
Fase 7	15/05/08	14/06/08

Figura 1. Diagrama de Gantt

El proyecto estaba previsto para ser entregado en la primera convocatoria, la de Junio, pero no se tuvo en cuenta la gran carga de trabajo que se tenía debido a las asignaturas que aun quedaban pendientes para el segundo semestre. Por esa razón principalmente, se abandonó un poco el proyecto dejándolo de lado hasta la finalización de los exámenes de Junio. Una vez concluidos los exámenes, a mediados de Junio, hubo una dedicación total al proyecto para poder presentarlo en la segunda convocatoria, correspondiente al mes de Setiembre, por eso la planificación sufrió unos ligeros cambios que serán presentados a continuación:

La primera fase queda tal y como se ha descrito anteriormente, es la única que se ha mantenido como se había dicho en la planificación inicial.

A partir de finales de febrero y hasta mediados de Junio se avanzó en la programación de los módulos del Firefox, creando la validación del formulario y

el rellenado automático. También se creó el diseño del sistema y los casos de uso.

Fue a principio de Julio cuando se creó el módulo J2ME, pero no en 2 semanas como estaba previsto en la planificación, sino en 1.

Después de esto, hubo muchos problemas para crear la comunicación desde el PC, como se describe en el apartado de los problemas encontrados y no se acabó dicha comunicación hasta finales de Agosto. También se creó la integración de dicho programa con el navegador. Habíamos planificado que iban a ser 6 semanas, pero se tardaron 7. Éstas 7 semanas se aprovecharon para adelantar gran parte de la memoria, dejando sin hacer los apartados de la implementación y las conclusiones.

Estos apartados de la memoria se terminarían en las 2 primeras semanas de Setiembre, junto con los detalles de la memoria y la integración de todo el programa junto.

La tercera semana se Setiembre se hará la presentación en PowerPoint del proyecto, que está previsto ser defendido para la última semana de Setiembre.

En la Figura 2 podemos ver el diagrama de Gantt correspondiente a los cambios efectuados en la planificación.

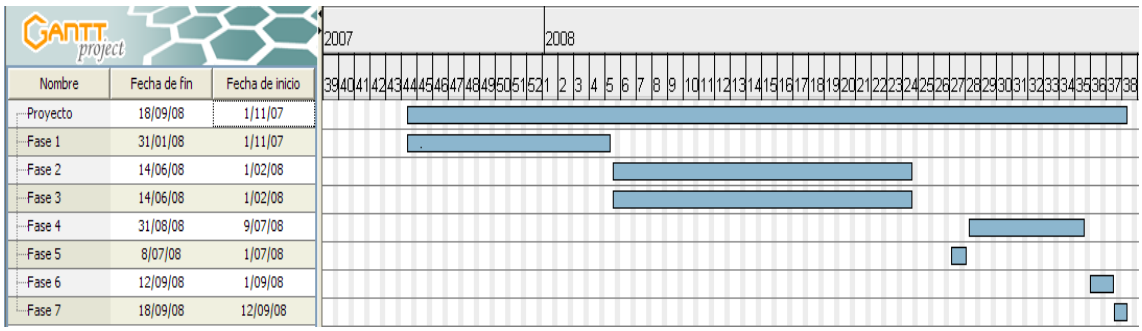


Figura 2. Diagrama de Gantt modificado

2. Tecnologías

En este capítulo hablaremos de las tecnologías usadas a lo largo de la creación del proyecto.

En un primer plano hablaremos de la tecnología **J2ME** desarrollada por Sun Microsystems y que es una variante del Lenguaje Java orientado para dispositivos móviles. Esta tecnología será usada para la creación de la aplicación del dispositivo móvil.

Por otro lado hablaremos de la tecnología **XUL** que es un lenguaje de interfaz de usuario basado en **XML**[1]. Es la tecnología usada en la creación de toda la interfaz gráfica de Firefox, por esa razón tendremos que hacer uso de ella, para variar dicha interfaz. Dentro de este apartado también se verá como dar funcionalidad a nuestras aplicaciones con Javascript y no solo eso, también se verá como aumentar aún más esa funcionalidad.

Finalmente se hablará de la tecnología **Bluetooth**, que permite la conectividad inalámbrica entre dispositivos remotos. Obviamente, es la tecnología que se usará para comunicar nuestro PC con el dispositivo móvil remoto.

[1] Extensible Markup Language (lenguaje de marcas extensible)

2.1. J2ME

J2ME es el acrónimo de Java 2 Micro Edition, la versión del lenguaje de programación Java desarrollada por Sun Microsystems y orientada al desarrollo de creación de aplicaciones para dispositivos móviles con pocas capacidades gráficas, de procesamiento y de memoria, como podrían ser los teléfonos móviles o las **PDA**'s[2].

La gran expansión de los teléfonos móviles en la última década ha hecho que las compañías telefónicas ofrezcan cada vez más prestaciones y servicios para sus terminales, como podría ser juegos u otras aplicaciones. Estos servicios de los terminales están desarrollados con la tecnología J2ME.

J2ME viene a ser la tecnología del futuro para la industria de los dispositivos móviles ya que proporciona una plataforma estándar para el desarrollo de aplicaciones y la facilidad de portar dichas aplicaciones entre diferentes dispositivos, sean o no del mismo fabricante.

Ya se están implantando los protocolos y los dispositivos necesarios para soportar la tecnología J2ME. Actualmente la mayoría de los terminales que salen al mercado ya están habilitados para usar esta tecnología.

2.1.1. Análisis de la plataforma Java 2

La versión de Java 2 de Sun se puede dividir en 3 ediciones distintas. **J2SE (Java Standard Edition)** orientada al desarrollo de aplicaciones independientes, **J2EE (Java Enterprise Edition)** orientada al entorno empresarial y **J2ME (Java Micro Edition)** orientada a dispositivos móviles.

[2] Personal Digital Assistant (Asistente Digital Personal)

La Figura 3 nos muestra la arquitectura de la plataforma Java 2. En la parte inferior de la Figura 3 se pueden ver las diferentes máquinas virtuales soportadas por las diferentes tecnologías, **JVM** y **KVM**, que se explicarán más adelante, como también se hará con los términos **CDC**, **CLDC** y **MIDP**.

Figura 3. Arquitectura de la plataforma Java 2 de Sun

Como se puede ver en la Figura 4, J2ME representa una parte simplificada de J2SE y a su vez ésta, representa una parte de J2EE.

Sun separó estas tecnologías por razones de eficiencia, ya que por ejemplo J2EE requiere unas características especiales de **E/S**[3] cosa que J2SE no. J2ME está pensado para dispositivos con pocas capacidades gráficas y de proceso, cosa que no sucede en J2SE, por eso también existe una separación bien clara entre estas dos tecnologías.

[3] Entrada/Salida

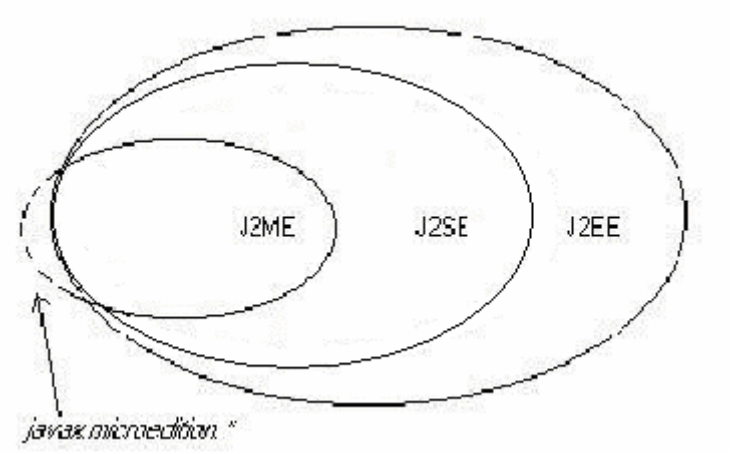


Figura 4. Relación entre las APIs de la plataforma Java.

2.1.2. Componentes J2ME

En este apartado vamos a ver cuáles son los componentes que forman parte de esta tecnología.

- **Máquinas virtuales de Java** (JVM), es un programa que se encarga de interpretar código precompilado por un programa Java. Gracias a las máquinas virtuales, los programas escritos en Java tienen independencia de la máquina donde han sido ejecutados.

- **Configuraciones**, son un conjunto de clases básicas en una categoría de dispositivos. Las categorías se miden por las prestaciones del dispositivo, capacidad de procesamiento o capacidad gráfica. Digamos que la configuración define la familia de dispositivos, según las capacidades de los dispositivos, se incluirán en una familia o en otra.

- **Perfiles**, son conjunto de clases que complementan a una configuración para unos dispositivos específicos. Los perfiles definen las características de un dispositivo. Son más específicos que las configuraciones.

Podemos ver en la Figura 5 la arquitectura de un entorno de ejecución. A continuación detallaremos un poco más cada uno de los tres componentes presentados anteriormente.

2.1.2.1. Máquinas Virtuales

Como ya se ha explicado antes, una máquina virtual de Java (JVM) es un programa encargado de interpretar código precompilado por un programa Java. La tecnología J2ME define varias máquinas virtuales, adecuándose a las dos configuraciones existentes en el mercado, la **Configuración de dispositivos limitados con conexión, CLDC** (Connected Limited Device Configuration) y la **Configuración de dispositivos con conexión, CDC** (Connected Limited Configuration). Nosotros solo nos centraremos en la primera de ellas, que es la que se adapta mejor a las características de nuestro dispositivo móvil. Estas configuraciones serán explicadas más profundamente en el apartado de configuraciones.

Como se ha comentado anteriormente, existe una máquina virtual para cada configuración, ya que éstas tienen características muy diferentes entre sí. Para la configuración CLDC, que es la que nos concierne, la máquina virtual se denomina **KVM (Kilo Virtual Machine)**, por su reducida memoria para ser ejecutada. Mientras que para la configuración CDC, la máquina virtual se denomina **CVM (Compact Virtual Machine)**.

A continuación veremos las características principales de la KVM.

- **KVM**

Es una máquina virtual orientada para dispositivos con pocas capacidades de procesamiento y de memoria. Fue diseñada para ser:

- Pequeña
- Alta portabilidad.
- Modulable.
- Lo más completa y rápida posible.

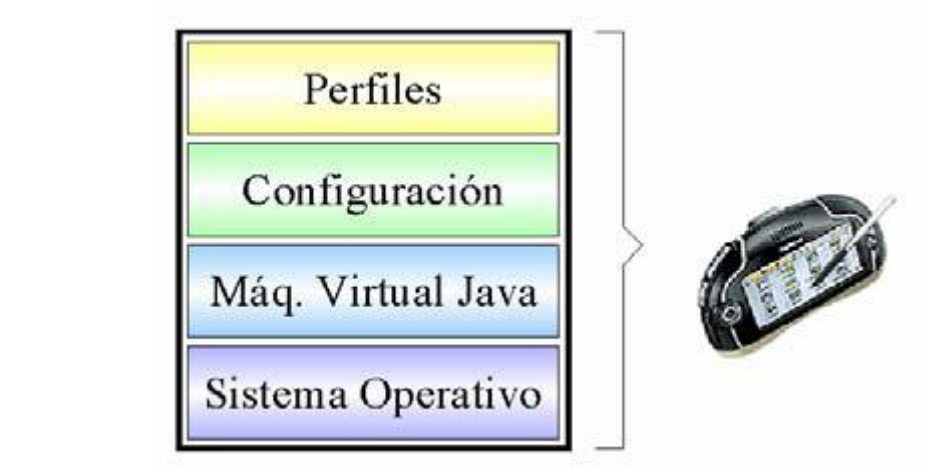


Figura 5. Entorno de ejecución

2.1.2.2. Configuraciones

Las configuraciones son el conjunto básico de **APIs**[4] que permiten desarrollar aplicaciones para una familia de dispositivos. Como ya se ha mencionado anteriormente, existen dos configuraciones en J2ME: CLDC, orientada a dispositivos con limitaciones de proceso y de memoria y CDC, orientada a dispositivos con mayores capacidades. Ahora veremos un poco más en profundidad estas configuraciones.

- **Configuración de dispositivos limitados con conexión, CLDC** (Connected Limited Device Configuration), un buen ejemplo de estos dispositivos con capacidades limitadas son los teléfonos móviles o las PDAs. Incluye las librerías `java.lang`, `java.util`, `java.io` y `javax.microedition.io`. Estas dos últimas librerías las necesitaremos para crear nuestra aplicación para el dispositivo móvil.

- **Configuración de dispositivos con conexión, CDC** (Connected Device Configuration), como se ha mencionado anteriormente, esta configuración está orientada a dispositivos con más capacidades que en

[4] Aplication Programming Interface (Interfaz de Programación de Interfaces)

la anterior configuración. Un buen ejemplo de estos dispositivos podría ser televisores con Internet o **GPS**[5]. Esta configuración añade más librerías que la anterior, que no vamos a numerar ya que no nos van a concernir en la realización del proyecto.

2.1.2.3. Perfiles

Los perfiles definen las características más detalladas de un dispositivo, como podría ser la interfaz de usuario o las conexiones de red. Identifican a los diferentes grupos de dispositivos por las funciones específicas que desempeñan y por el tipo de aplicación que se ejecutará en ellos. Los perfiles se construyen sobre una configuración determinada y permiten la portabilidad de aplicaciones J2ME entre diferentes dispositivos.

Como ocurría en el apartado de configuraciones, aquí también hay diferentes perfiles según el tipo de configuración sobre la que queramos construir la aplicación. Existen perfiles específicos para la configuración CDC y otros para la configuración CLDC.

Para la configuración CDC tenemos los siguientes perfiles:

- Foundation Profile
- Personal Profile
- **RMI** [6]Profile

Para la configuración CLDC tenemos los siguientes perfiles:

- PDA Profile
- **Mobile Information Device Profile (MIDP)**

[5] Global Positioning System (Sistema de Posicionamiento Global)
[6] Remote Method Invocation (Invocación de Método Remoto)

Como ya se ha marcado, nos centraremos en el perfil MIDP, que dentro de la configuración CLDC (la que nos concierne) es el perfil que más se adecúa a las características de nuestro dispositivo móvil.

Mobile Information Device Profile (MIDP): Como ya se ha visto en la clasificación anterior, este perfil está construido sobre la configuración CLDC. Decimos que es el que mejor se adapta a las características de nuestro dispositivo, ya que es un perfil orientado a teléfonos móviles, como lo es nuestro dispositivo.

Este perfil incluye algunas de las librerías básicas para la creación de nuestra aplicación en el dispositivo móvil, más concretamente la interfaz de éste, como `javax.microedition.lcdui` o `javax.microedition.midlet`.

Las aplicaciones que realizamos utilizando el perfil MIDP reciben el nombre de MIDlets (por simpatía con **Applets**[7]). Entonces podemos afirmar que un MIDlet es una aplicación Java realizada con el perfil MIDP sobre la configuración CLDC.

[7] Componente de una aplicación que se ejecuta en el contexto de otro programa.

2.2. XUL

XUL es el acrónimo de XML-based User-interface Language, o mejor dicho, Lenguaje Basado en XML para la Interfaz de Usuario. Es el lenguaje que utiliza el software de Mozilla, en nuestro caso el navegador Firefox, para definir su interfaz de usuario. Con él, podremos modificar dicha interfaz a nuestro antojo, no sin antes tener claros algunos conceptos que pasaremos a explicar a lo largo de este apartado.

Al estar basado en XML hace que los datos estén almacenados en simples archivos de texto, facilitando así la portabilidad de dichos datos.

Como anteriormente se ha comentado, tiene la gran desventaja de ser excluyente, razón que hace que el lenguaje XUL no se haya convertido en un lenguaje estándar.

Pero también se puede ver esa desventaja como una ventaja. Los desarrolladores que quieran hacer una interfaz de usuario o una extensión para el navegador, lo tendrán mucho más fácil, ya que XUL fue concebido con esa idea, la de aligerar el trabajo de los desarrolladores.

Este proyecto se basa en uno de los productos de Mozilla, el navegador Firefox, por lo que para nosotros esta ventaja supone una gran ayuda.

En la Figura 6 se puede ver un ejemplo de una aplicación creada en XUL, se trata de un simple **TPV**[8] para la administración de una tienda.

2.2.1. Registro chrome

El procesamiento de XUL va muy ligado al procesamiento **HTML**[9], ya que este último accede a un sitio web deseado por el usuario y descarga su

[8] Terminal Punto de Venta

contenido. Este contenido recién descargado, es transformado por el motor de Mozilla en un árbol. Los nodos del árbol, a su vez, se convierten en un conjunto de objetos, que representan las partes las partes del documento que serán mostradas en pantalla.

El procesamiento es muy parecido a HTML, pero XUL tiene algunas características propias, como puede ser los **Overlays**[10], que se explicarán en el siguiente apartado o el registro **chrome**, que ahora explicaremos.

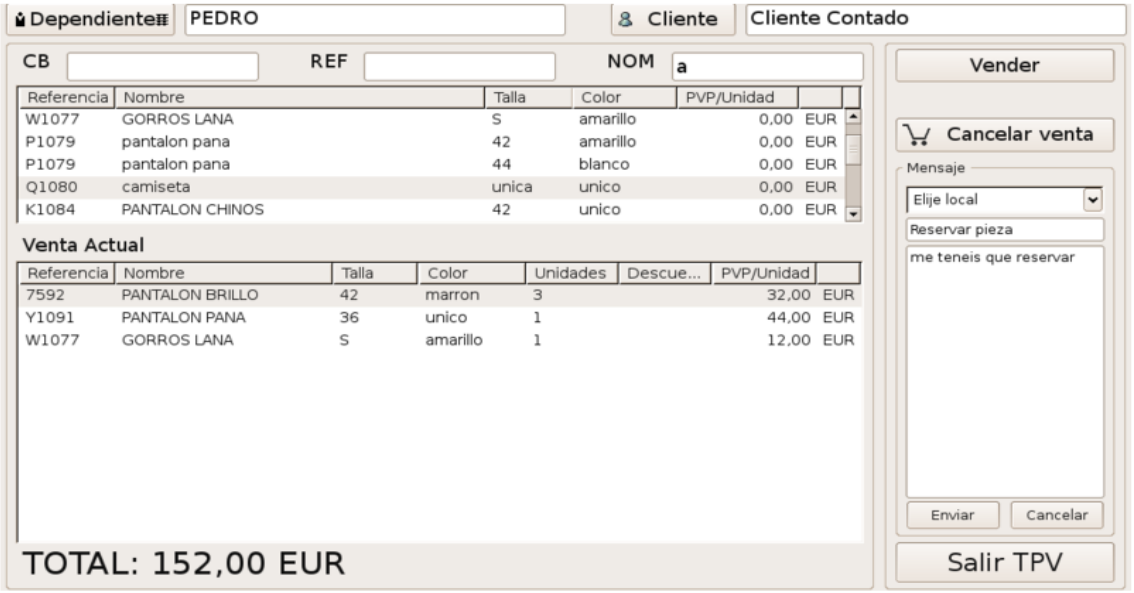


Figura 6. Aplicación creada en XUL

Como es obvio, el contenido de una fuente remota como podría ser una página web, no puede acceder, por ejemplo, a los archivos de directorios locales del usuario o ejecutar operaciones privilegiadas, por razones de seguridad.

Por esta razón, Mozilla creó un registro especial, llamado el registro chrome, para poder instalar el contenido de las aplicaciones en directorios locales y quedar de esta manera registrados en el sistema, para que dichas aplicaciones tengan permisos ampliados.

[9] HyperText Markup Language (Lenguaje de Marcas de Hipertexto)
[10] Revestimientos

Para acceder a estos archivos se usa una **URL**[11] especial, la URL chrome. Al acceder a un archivo usando esta URL especial, éste gana privilegios especiales, como podría ser el acceso a archivos locales o ejecutar operaciones privilegiadas.

La URL chrome se refiere al directorio `chrome` que hay dentro de la instalación de Mozilla Firefox, pero al crear una extensión, no vamos a crear los archivos dentro de ese directorio, sino que crearemos una jerarquía de directorios tal que en el directorio principal, (cuyo nombre puede ser el nombre de la extensión) tendremos un directorio llamado `chrome` el cual contendrá los archivos necesarios para crear la extensión y es el directorio que Firefox añadirá al registro chrome para ganar los privilegios mencionados anteriormente.

Dentro del directorio principal, junto al directorio `chrome`, crearemos un archivo llamado `install.rdf` para que Firefox "vea" las características principales de nuestra extensión. También en ese mismo directorio se creará un archivo llamado `chrome.manifest` para que Firefox sepa dónde buscar los archivos principales que conforman nuestra extensión y para añadir los Overlays, que serán explicados en el siguiente apartado. Como se acaba de decir, los archivos principales de nuestra extensión estarán dentro del directorio `chrome` y los dividiremos en varios directorios. En un directorio llamado `skin` crearemos los archivos de los estilos o **CSS**[12] y en otro llamado `content` crearemos los archivos XUL de la interfaz y los archivos Javascript para la funcionalidad.

Finalmente todo esto se empaquetará con un compresor de archivos, como podría ser winrar y se cambiará el nombre de la extensión, en vez de acabar en `.rar` tiene que acabar en `.xpi` para que Firefox lo reconozca como una extensión y proceda a su instalación, ya sea en nuestro equipo o en cualquier otro. De ahí la importancia de la jerarquía de directorios, ya que una mala

[11] Uniform Resource Locator (localizador uniforme de recurso)
[12] Cascading Style Sheets (Hoja de estilo en cascada)

jerarquía hará que Firefox no “entienda” lo que contiene la extensión y haga que no funcione correctamente.

Como acabamos de ver, las extensiones no son solo archivos XUL, sino que también están formadas por hojas de estilo y archivos Javascript entre otros que explicaremos más adelante.

Como se ha visto en este apartado, el registro chrome juega un papel muy importante a la hora de la creación de extensiones, ya que necesitaremos tener los privilegios otorgados por tal registro para poder llevar a cabo la elaboración de una extensión.

También se ha comentado el concepto de los Overlays, que también juegan un papel importante y que pasaremos a explicar en el siguiente apartado.

2.2.2. Overlays

Los Overlays sirven para definir contenido extra en la interfaz gráfica de usuario.

Podemos añadir componentes adicionales a la interfaz de usuario , como podría ser por ejemplo, una nueva entrada en el menú “`Herramientas`” de nuestro navegador Firefox.

Gracias a los archivos Overlay podemos conseguir esto sin tener que reescribir toda la interfaz de usuario del navegador.

Si desde nuestra extensión queremos añadir algún elemento adicional a la interfaz de usuario, esto se tendrá que hacer mediante el uso de los Overlays. Como se ha comentado en el apartado anterior, en el archivo `chrome.manifest` se tiene que especificar que se quiere hacer un Overlay sobre una parte de la interfaz de usuario.

Pongamos un ejemplo, supongamos que queremos añadir como acabamos de comentar, una nueva entrada en el menú "`Herramientas`" del navegador. Para hacer posible esto se tendría que crear una entrada parecida a la siguiente en el archivo `chrome.manifest`:

```
Overlay                chrome://browser/content/browser.xul
chrome://nombre_extension/content/nombre_archivo.xul
```

El parámetro `Overlay` nos está indicando que se trata de un Overlay, seguidamente separado por un espacio hay que poner el archivo base que se quiere "revestir" y finalmente separado con otro espacio, el archivo creado por nosotros que añadirá el elemento al menú "`Herramientas`" del navegador. Como se puede suponer, el primero de los archivos se trata de toda la interfaz del navegador Firefox al cual le vamos a añadir el nuevo elemento.

2.2.3. DOM

Como ya se ha comentado anteriormente, una extensión no solo se compone de archivos XUL, sino no tendríamos nada de funcionalidad en nuestra aplicación. Hace falta añadir archivos Javascript a nuestra extensión para añadir dicha funcionalidad. Esta funcionalidad se basa en interactuar con las partes del código de una página web y realizar acciones. ¿Pero cómo accedemos al código de una página web?

Anteriormente, en el apartado de procesamiento de XUL, se ha descrito el procesamiento de éste, diciendo que el lenguaje HTML accede a la web deseada por el usuario y descarga el contenido, pasándolo al motor de Mozilla que transforma este contenido en un árbol y los nodos del árbol, a su vez, se convierten en un conjunto de objetos. Pues bien, a este conjunto de objetos, Javascript puede acceder a ellos gracias al **DOM.**

DOM es el acrónimo de Document Object Model o mejor dicho, Modelo de Objetos para la representación de Documentos y es una API para documentos HTML y XML.

El DOM permite el acceso a los elementos HTML de una página web. Estos elementos se convierten en nodos y cada trozo de texto en un nodo de texto.

Es importante entender la diferencia entre elementos y nodos de textos. Los elementos normalmente están asociados a las etiquetas. En HTML todas las etiquetas son elementos, tales como `<p>`, `<img>` y `<div>` por lo que tienen atributos y contienen nodos hijo. Sin embargo, los nodos de textos no poseen atributos e hijos.

En el siguiente ejemplo se puede ver la diferencia entre nodos de texto y elementos:

```
<body>
<p>Esto es un párrafo que contiene <a href="#">un enlace</a> en el medio. </p>
<ul>
<li>Primer punto en la lista</li>
<li>Otro punto en la lista</li>
</ul>
</body>
```

Como podemos observar el elemento `<a>` está dentro del elemento `<p>` convirtiéndose en su hijo. También podemos observar diferentes nodos de texto entre los elementos.

Javascript permite acceder a cada uno de los elementos de una página utilizando tan sólo algunos métodos y propiedades. Dos de los más importantes y que más vamos a usar en nuestra extensión son los métodos `getElementById` y `getElementsByTagName`. El primero nos servirá para

encontrar un elemento de la página web solo con saber su atributo `id`. Véase el siguiente ejemplo:

```
<p>
<a id="contacto" href="contactos.html">Contáctenos</a>
</p>
```

Puede usarse el atributo `id` del elemento `a` para poder acceder a él, como se muestra a continuación:

```
var elementoContacto = document.getElementById("contacto");
```

El segundo de los métodos sirve para trabajar sobre un grupo de elementos. Si llamásemos al método con el parámetro `input`, nos devolvería todos los elementos del documento que se denominen input.

Con el DOM se puede encontrar, cambiar, adicionar y eliminar elementos de una página web. Es una técnica poderosa para poder ser usada en el desarrollo de nuestra extensión.

2.2.4. XPCOM

Ya hemos visto como añadir funcionalidad a nuestra extensión mediante Javascript y el DOM. Pero imaginemos que necesitamos hacer alguna cosa bastante compleja, como podría ser el acceso mediante Bluetooth a un dispositivo móvil, sí, como en la extensión que vamos a crear.

Con Javascript no podemos hacer cosas complejas como la que acabamos de describir, ya que fue diseñado para ejecutar aplicaciones muy limitadas. Un programa escrito es Javascript no puede, por ejemplo, acceder a los archivos de nuestro ordenador.

Por ese motivo necesitamos algo que nos deje ejecutar aplicaciones más complejas en nuestra extensión, porque solo con Javascript se limita mucho la funcionalidad de la extensión.

Mozilla pensó en esto y por eso nos proporcionó una herramienta potentísima, los componentes **XPCOM** (Cross-platform Component Object Model o Modelo de Objeto de Componentes Multiplataforma).

Los componentes XPCOM están escritos en código nativo, por lo que pueden hacer cosas que Javascript por sí solo no podría. Pero seguimos teniendo el mismo problema, ya que en principio no podemos usar mas que Javascript para crear nuestra extensión, y los componentes están escritos en otro código. Por esa razón Mozilla nos proporciona una capa que permite que los objetos Javascript accedan y manipulen los objetos XPCOM, para poder ser usados desde Javascript. Esta capa se llama **XPConnect.**

Por lo tanto tenemos que los componentes XPCOM implementan la funcionalidad y las **interfaces** describen cómo se debe implementar esa funcionalidad mediante un conjunto de definiciones.

Mozilla tiene implementadas muchas interfaces, para poder ser usadas a nuestro antojo. Estas interfaces se las reconoce porque el nombre suele empezar con el prefijo `nsI`. Así, existen interfaces para el manejo de archivos, `nsILocalFile` o para la reproducción de sonidos, `nsISound`.

Si no existiera una interfaz determinada para la necesidad que queremos cubrir, en nuestro caso la comunicación Bluetooth, tendríamos que crearla nosotros mismos y registrarla en el navegador para poder usarla desde Javascript. En el capítulo de la implementación se explica cómo crear un componente XPCOM desde cero.

2.3. Bluetooth

Bluetooth es una tecnología de radio de corto alcance, que permite la conexión entre dispositivos remotos sin la necesidad de cables. Fue diseñado pensando básicamente en tres objetivos: pequeño tamaño, mínimo consumo y bajo precio.

Todas las compañías telefónicas están incorporando esta tecnología en todos los dispositivos móviles que lanzan al mercado. Ya hace tiempo que casi todos los móviles que salen nuevos al mercado incorporan la tecnología Bluetooth.

Es obvio entonces, que utilicemos esta tecnología para la comunicación entre PC y dispositivo móvil para la transferencia de datos.

En el lado del dispositivo móvil, utilizaremos la tecnología Bluetooth para el desarrollo de una aplicación en J2ME y en el lado del PC utilizaremos esta misma tecnología pero para el desarrollo de una aplicación escrita en **C++.**[13] Es por esta razón que deberemos usar APIs diferentes a la hora de programar las aplicaciones del PC y del dispositivo móvil.

Ahora pasaremos a ver las principales características de las diferentes APIs que vamos a usar en el desarrollo de la extensión.

2.3.1. APIs Java para Bluetooth

Hasta que no apareció la especificación **JSR**[14] 82 no se podía crear aplicaciones Java Bluetooth. Esta especificación fue la que estandarizó la manera de crear aplicaciones Bluetooth usando la tecnología Java.

[13] Lenguaje de Programación Orientado a Objeto.
[14] Java Specification Request (Petición de Especificación de Java)

Aquí no vamos a describir toda la API JSR 82 pero sí algunas características que nos serán útiles a la hora de desarrollar la comunicación en el lado del dispositivo móvil.

Primeramente vamos a ver la parte de la API que se encarga de descubrir dispositivos remotos disponibles dentro del radio de cobertura.

Para empezar tenemos la clase `javax.bluetooth.LocalDevice` que es la encargada de acceder y controlar el dispositivo local, el nuestro. Dentro de esta clase tenemos el método `getLocalDevice` que nos proporcionará el acceso a nuestro dispositivo remoto. El acceso a nuestro dispositivo con el método anterior, se usará para hacer que nuestro móvil sea visible para otros dispositivos remotos. Esto se conseguirá con el método `setDiscoverable`.

Para descubrir dispositivos remotos tendremos que recurrir a la clase `javax.bluetooth.DiscoveryAgent`. Dentro de esta clase tenemos el método `getDiscoveryAgent` que nos retornará los dispositivos encontrados por nuestro dispositivo local. Con estos dispositivos encontrados, tendremos que seleccionar al que queramos acceder para establecer la comunicación. Para hacer esta selección primero tendremos que utilizar el método `selectService` de la clase anterior, que como parámetro habrá que pasarle el identificador único del dispositivo al que queramos acceder. Este identificador lo tienen todos los dispositivos para ser diferenciados entre sí, ya que el identificador es único (**UUID**[15]). Este identificador lo proporciona la clase `javax.bluetooth.UUID` y sirve tanto para crear el identificador en el lado del servidor, como para descubrir dicho identificador si se le pasa como parámetro al método `selectService`, comentado anteriormente, en el lado del cliente.

Con todo lo acabado de comentar, ya seremos capaces tanto de descubrir dispositivos remotos como de ser descubiertos por otros.

[15] Universally Unique Identifier (Identificador Universal Único)

Ahora solo nos hará falta llegar a realizar la conexión con el dispositivo que hayamos elegido para el intercambio de datos. Esto es precisamente lo que vamos a ver, la parte de conexión de la API.

Para crear una conexión, si estamos en el lado del servidor, lo primero que tenemos que crear es una URL con una serie de parámetros específicos. En el siguiente ejemplo podemos ver una URL válida:

```
btspp:// localhost:UUID
```

El primer parámetro determina que estamos creando una URL para dispositivos Bluetooth con conexión al puerto serie. `Localhost` es para saber que estamos en el lado del servidor y el UUID, como se ha dicho anteriormente, es el identificador único del dispositivo. Ahora tendremos que hacer uso de los métodos de la clase `javax.microedition.io.Connector` para poder establecer la conexión. Concretamente usaremos el método `open`, al que le pasaremos cómo parámetro la URL creada. Este método será el encargado de crear y abrir la conexión. Ahora solo hace falta esperar a que alguien solicite conectarse con nosotros, entonces nosotros estableceremos dicha conexión con el método `acceptAndOpen` y pasaremos a procesar la petición.

Si estamos del lado del cliente la cosa es más sencilla. Ya sabemos cómo seleccionar el dispositivo mediante el UUID, ya que se explicado anteriormente. Entonces solo nos hace falta llamar al método `open` y esperar que el servidor acepte nuestra conexión.

Con todo lo descrito, aparte de descubrir dispositivos, ya seremos capaces de realizar una conexión a éstos o esperar una petición de conexión para poder establecerla.

2.3.2. APIs C++ para Bluetooth

La API que usaremos para desarrollar la aplicación en el lado del PC, será la API C++ para Bluetooth de Microsoft.

Vamos a empezar definiendo todo lo necesario para poder crear una conexión en el lado del cliente, usando **sockets**[16] y la tecnología Bluetooth.

Para empezar, tenemos que llamar a la función `WSAStartup` que nos proporcionará información necesaria sobre **Winsock**[17], como podría ser la versión y los detalles de la implementación. Una vez ya hemos recogido la información necesaria de Winsock, ya podemos pasar a crear el socket que vincularemos con el del servidor para establecer la conexión. La creación de un socket se efectúa mediante la función `socket` con unos parámetros específicos, que variarán según el tipo de conexión que hagamos. En nuestro caso, queremos realizar una conexión Bluetooth, por lo que los parámetros serán, `AF_BT`, `SOCK_STREAM` y `BTHPROTO_RFCOMM`. Siempre usaremos estos parámetros cuando queramos realizar una conexión mediante Bluetooth.

Una vez creado el socket, tendremos que guardar información sobre el dispositivo Bluetooth al que nos queremos conectar. Para realizar esto se usará la estructura `SOCKADDR_BTH` que nos proporciona la API. Rellenaremos los campos de dicha estructura, con la información necesaria proveniente del dispositivo servidor.

Ahora ya podemos crear la conexión. Ésta se hará mediante la función `connect` que recibirá como parámetros, el socket local y la estructura con la información necesaria del dispositivo remoto para iniciar la vinculación.

Una vez producida la conexión, ya podremos enviar y recibir datos mediante las funciones `send` y `recv`, respectivamente.

[16] Punto de acceso, por el cual otro dispositivo remoto puede conectarse a nosotros.
[17] Sockets de Windows

Ésta ha sido la descripción de cómo crear una conexión en el lado del cliente, ahora vamos a ver cómo hacerlo en el lado del servidor.

La conexión en el lado del servidor empieza de la misma manera que en la del cliente, con la llamada a la función `WSAStartup` y con la creación del socket.

Ahora hay una pequeña diferencia con respecto a la parte cliente, en vez de rellenar la estructura `SOCKADDR_BTH` con la información del dispositivo al que nos queremos conectar, tendremos que rellenarla con la información de nuestro dispositivo, ya que ahora nos encontramos en el lado del servidor.

Una vez tenemos la estructura creada, tenemos que asociar dicha estructura a nuestro socket. Esto se consigue con la función `bind`, haciendo que el socket creado al inicio tenga la información de la estructura.

Ahora solo tenemos que esperar a conexiones de entrada por el lado del cliente. Esto se consigue utilizando la función `listen`. Ahora solo nos queda aceptar dichas conexiones mediante la función `accept`.

Por último y no menos importante, no olvidarnos de cerrar el socket al acabar la conexión mediante la función `closesocket`.

Con esta introducción a la API Bluetooth de Microsoft, ya seremos capaces de realizar una aplicación que detecte a un dispositivo remoto y se conecte a él para realizar algún tipo de intercambio de datos.

3. Diseño y Arquitectura

3.1. Introducción

El diseño determina la arquitectura general del sistema que mejor satisface los requisitos: componentes del sistema, el software utilizado y la interacción con el usuario. Define como debe realizar su función el sistema.

Los actores son personas o entidades externas que intercambian información con el sistema. En éste sistema tenemos 1 actor, el usuario que utiliza la extensión y activa la aplicación de su dispositivo móvil.

Un caso de uso representa una interacción típica entre el usuario y el sistema. Los casos de uso se utilizan para capturar los requisitos funcionales del sistema. La descripción de éstos se centra en el comportamiento y no en la implementación de las partes que define. En la figura 7 podemos ver los casos de uso de nuestro sistema.

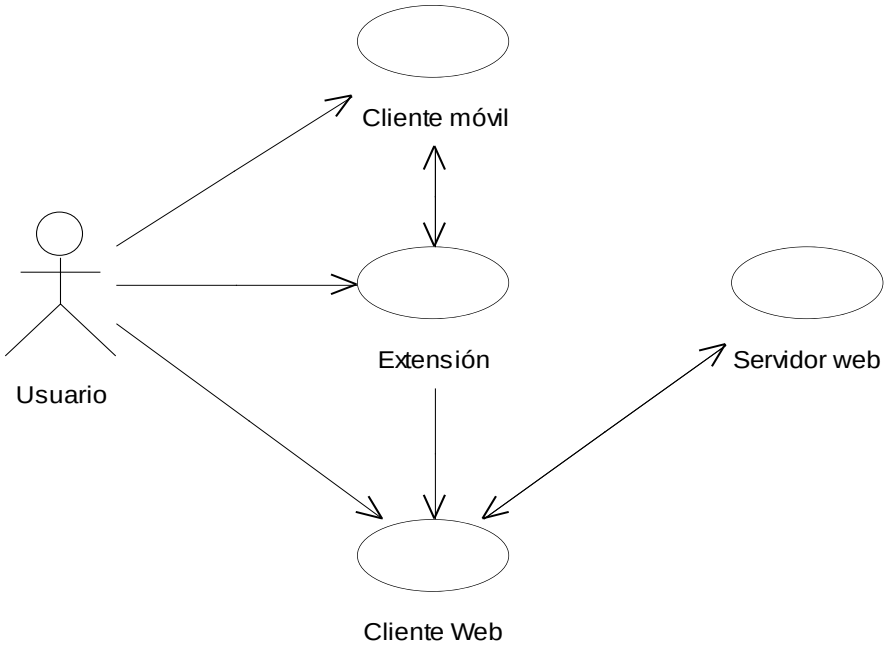


Figura 7. Diagrama de casos de Uso

3.2. Servidor web

El servidor Web es el encargado de recoger los datos introducidos en el navegador y dar acceso a la página web deseada. También es el responsable de recoger los datos introducidos en el formulario de la página web con acceso restringido a la que deseamos acceder, validar estos datos introducidos en la base de datos y si todo es correcto, dar acceso al usuario a la página web solicitada.

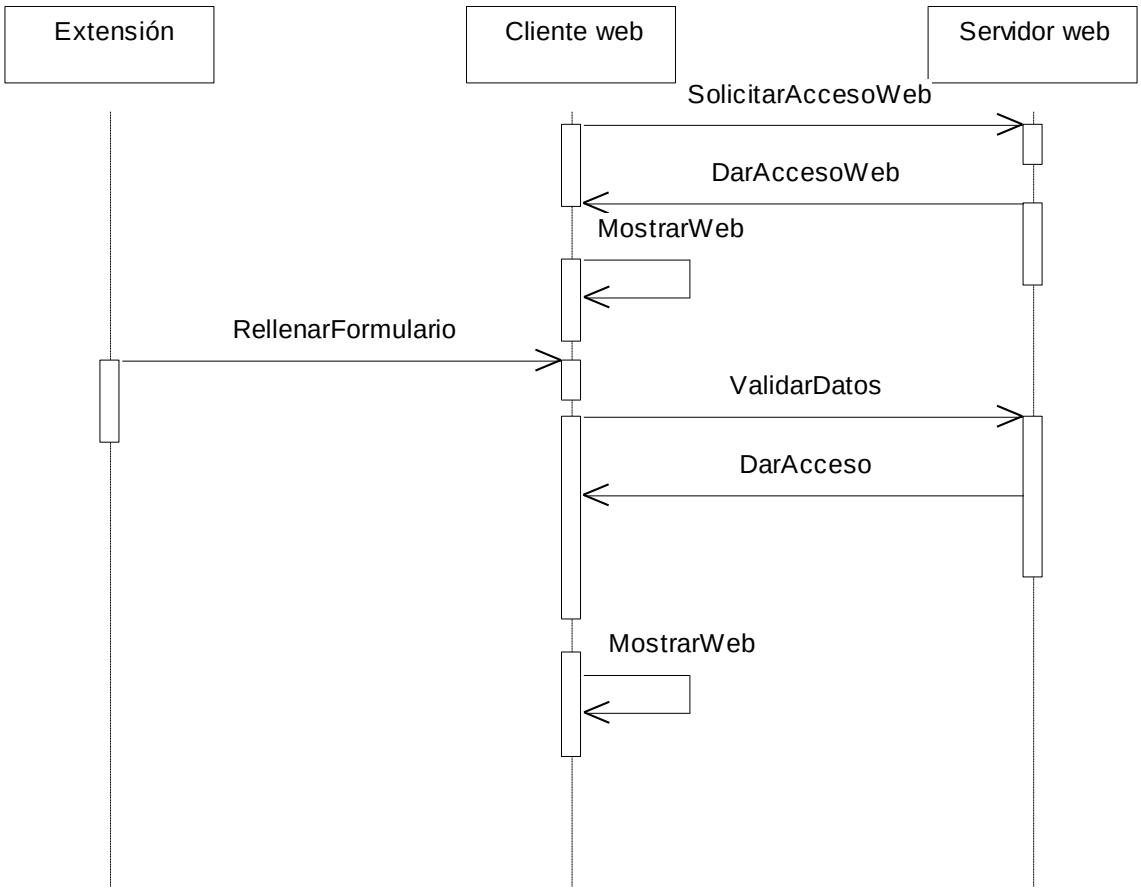


Figura 8. Diagrama del caso de uso Servidor Web

3.3. Cliente Web

Como cliente Web entendemos al navegador web, que en nuestro caso será el Firefox de Mozilla y a la extensión en sí, que es la encargada primero de detectar si la página web en la cual estamos en el momento de activar la extensión, contiene un formulario de login/password, con lo que iniciará el proceso de comunicación con el móvil. Si no existiera el formulario de login/password en la página web en que nos encontramos, saldría una alerta por pantalla mostrando un mensaje de error.

Se barajaron varias opciones a la hora de iniciar la aplicación, una es la que acabamos de comentar, tener que ir al menú "`Herramientas`" del navegador Firefox y activar manualmente la extensión.

La otra opción que se barajó, fue la de que cada vez que entremos en una página web nueva, la aplicación se inicie automáticamente sin tener que ir al menú "`Herramientas`" del navegador e inicie el todo el proceso necesario para rellenar el formulario automáticamente.

Aunque la segunda opción es más cómoda en principio, ya que se inicia automáticamente cada vez que entramos en una página web, al final se hace más engorrosa ya que cada vez que una página no contenga un formulario de login/password saldrá el mensaje de error por pantalla. Si por casualidad entramos en una página que si contiene ese tipo de formularios pero para el cual no tenemos la contraseña guardada en el móvil también nos saldrá un mensaje de error por pantalla. Por estas dos incomodidades, se ha preferido usar la primera opción, aunque se tenga que activar manualmente.

Para incorporar en el menú "`Herramientas`" de nuestro navegador la opción de iniciar la aplicación, se ha creado un pequeño programa en XUL y se ha colocado en el directorio de extensiones de Firefox. Se ha elegido incorporar el elemento de inicio de la aplicación en dicho menú, ya que como hemos podido contemplar con otras extensiones, éstas siempre incorporan elementos

adicionales al menú "`Herramientas`" una vez están instaladas en el navegador.

Mediante el registro de la extensión en la URL chrome del navegador y gracias a los Overlays, podremos añadir la nueva entrada deseada, en el menú "`Herramientas`" en nuestro caso.

El cliente Web se comunica con el dispositivo móvil mediante la creación de sockets y gracias a la API Bluetooth de Microsoft. Se creará un componente XPCOM para tal efecto que también será el encargado de recibir los datos necesarios para rellenar los campos del formulario. Una vez nuestra extensión tiene los datos, ésta se encarga de rellenar automáticamente los campos del formulario mediante una función escrita en Javascript.

En la Figura 9 podemos ver el diagrama del caso de los casos de uso Extensión y Cliente Web. Se ha unido el diagrama de los dos ya que la Extensión es un complemento del Cliente Web, por tanto forma parte de él.

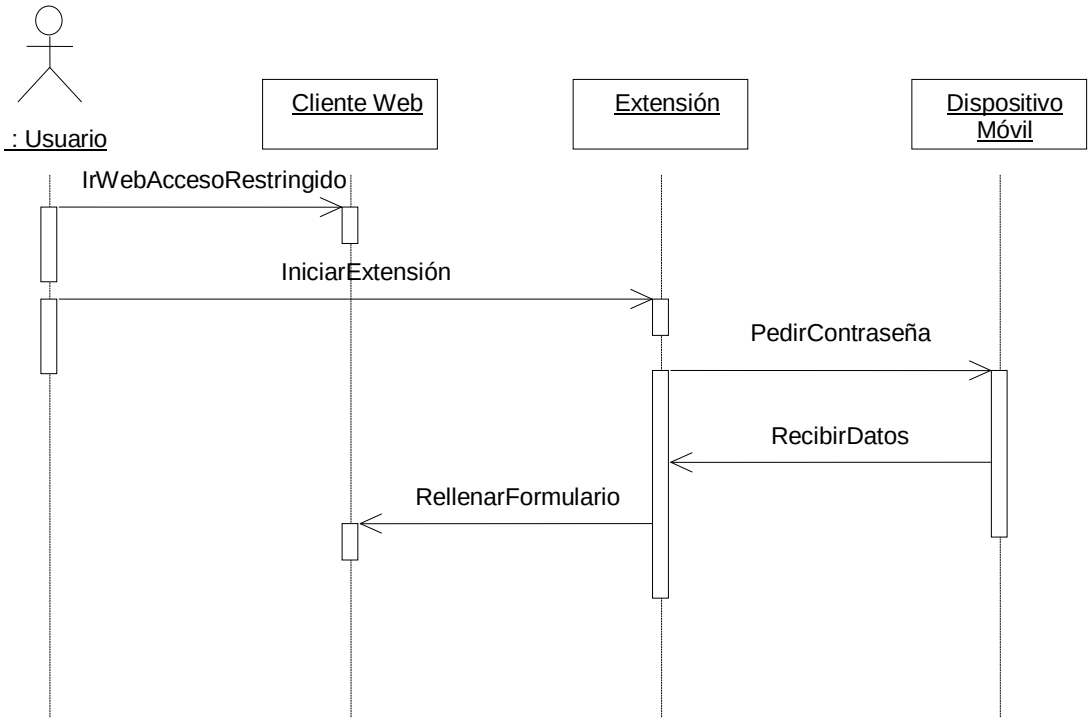


Figura 9. Diagrama del caso de uso Extensión

3.4. Cliente Móvil

El cliente móvil, en nuestro caso un teléfono móvil, es el encargado de esperar peticiones procedentes del navegador y satisfacer dichas peticiones. Además, una vez llegue una petición procedente del "exterior" se hará un proceso de emparejamiento de dispositivos, para asegurar que nadie, excepto nosotros, tenemos acceso a la contraseña que nos brindará el dispositivo móvil.

Este emparejamiento se realiza una vez que llega una petición. El dispositivo móvil nos pide que introduzcamos una contraseña, que una vez introducida, esperará a que el otro lado de la conexión introduzca la misma contraseña para poder hacer la vinculación. El otro lado de la conexión, en nuestro caso el PC donde estemos ejecutando la extensión, nos pedirá que entremos una contraseña, que introduciendo la misma que acabamos de poner en el dispositivo móvil, se producirá la vinculación y se podrá empezar la transferencia de datos. Al producirse la primera vinculación, tendremos una opción en el dispositivo móvil de guardar dicha vinculación, para que las próximas veces que se intente acceder al dispositivo móvil desde el mismo PC que la primera vez, no haga falta la introducción de ninguna contraseña, evitando así que la vinculación se convierta en una tarea muy engorrosa.

La aplicación del dispositivo móvil está escrita en el lenguaje de programación Java para móviles, J2ME y se comunica con la extensión mediante sockets y mediante la tecnología inalámbrica de redes, Bluetooth, gracias a la API Java para dicha tecnología.

Esta aplicación consiste en un **bucle**[18] infinito que se dedica a esperar peticiones y cuando una de éstas se produce, se crea un canal de salida por donde irán los datos hacia la extensión.

La aplicación ha sido escrita con el entorno de programación NetBeans más el paquete Mobility, que facilita mucho las cosas ya que dentro de éste hay una

[18] repetición

opción para crear Midlets, que nos crea los archivos necesarios, una vez ya tenemos la aplicación programada, para instalar nuestras aplicaciones en nuestro dispositivo móvil. Estos archivos consisten en dos archivos llamados `nombre_de_extensión.jar` y `nombre_de_extensión.jad` que simplemente los tendremos que transferir a nuestro móvil, ya sea por cable serie, infrarrojos o bluetooth, para poder usar la aplicación en nuestro dispositivo móvil.

Una vez que iniciemos la aplicación en nuestro móvil, ya estará lista para esperar peticiones procedentes del navegador.

En la Figura 10 podemos ver el diagrama del caso de uso del cliente móvil.

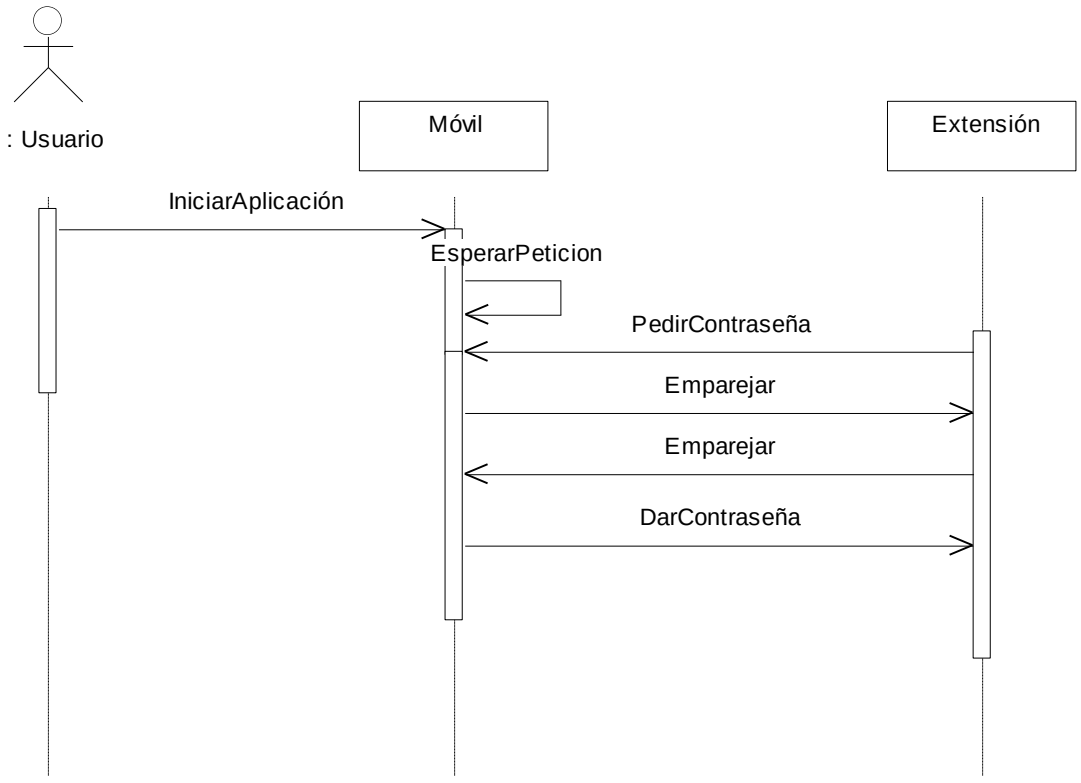


Figura 10. Diagrama del caso de uso Móvil

4. Implementación

La fase de implementación consiste en la traducción del diseño realizado anteriormente a un lenguaje de programación.

Los lenguajes de programación propuestos han sido java y su variante para dispositivos móviles J2ME, C++, XUL y Javascript. Como veremos en el punto 4.4. ha habido diferentes variantes en la implementación, a causa de los problemas encontrados.

En la Figura 11podemos ver el diagrama de clases de nuestro sistema, que describe las clases, atributos y las relaciones entre ellos.

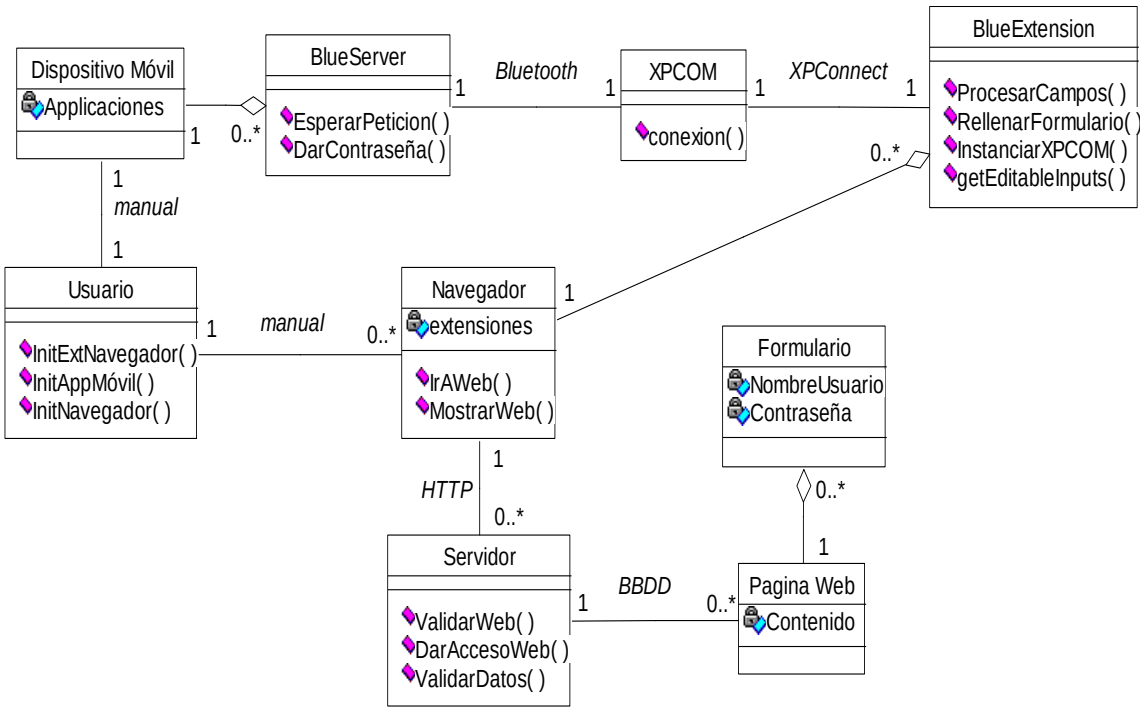


Figura 11. Diagrama de clases

Para empezar, tenemos la clase `Usuario`, que es la clase que nos representa, y los métodos son iniciar el navegador, iniciar la aplicación móvil e iniciar la

extensión del navegador. La clase `Usuario` se comunicará con el navegador y con el dispositivo móvil de forma manual.

La clase `Dispositivo Móvil` tiene agregada la clase `BlueServer` que representa la aplicación del dispositivo móvil. Esta última clase tiene los métodos necesarios para esperar la petición procedente de la extensión y dar la contraseña.

La clase `Navegador` tiene agregada la clase `BlueExtension` que representa la extensión del navegador Firefox y que tiene los métodos necesarios para la comunicación con el dispositivo móvil y el relleno automático de los campos del formulario. Para iniciar la conexión, la clase `BlueExtension` tendrá que hacer uso de la clase `XPCOM` y la conexión entre estas dos clases se realiza mediante XPConnect. La clase `XPCOM` se comunicará con la clase `Dispositivo Móvil` mediante Bluetooth.

La clase `Navegador`, por su parte, se comunicará con la clase `Servidor` mediante HTTP. La clase Servidor será la encargada de validar las páginas web introducidas por la clase `Navegador` y de dar acceso a ellas. También será la encargada de validar los datos del formulario. Para validar las páginas web, la clase `Navegador` tendrá que “mirar” en su Base de Datos (BBDD) y si existe dicha web, dar acceso a ella.

La clase `Pagina Web` tendrá como agregación la clase `Formulario`, que es la que contendrá los atributos nombre de usuario y contraseña.

4.1. Software usado

El software usado para la implementación de la aplicación es el siguiente:

- Microsoft Windows XP service Pack 3
- Microsoft Office 2007
- Microsoft Visual Studio 6.0
- Microsoft Platform SDK
- Notepad ++
- Netbeans mobility pack
- Nokia Wireless Toolkit
- Firefox
- XulRunner SDK o Gecko SDK
- XulExplorer
- DOM Inspector
- Extension Developer

4.2. Hardware empleado

- Sony Vaio FS-515b
- Procesador Intel Pentium M a 1,73 GHz
- 1GB RAM
- HDD de 80 GB
- WidComm Bluetooth USB Adapter
- Nokia 6300
- Impresora Epson Stylus DX9400F

4.3. Servidor

En el servidor nosotros no tenemos que programar nada, simplemente esperar primero a que nos dé acceso a la página web que deseemos entrar y luego a que cuando hayamos introducido nuestros datos en el formulario deseado, los valide y si los datos son correctos, nos de acceso al área restringida a la que queremos acceder. Todos estos datos serán introducidos por el usuario mediante el Cliente Web.

Nuestra aplicación solo servirá para entrar en una sola página web, ya que en nuestro dispositivo móvil solo tendremos guardada la contraseña para un solo formulario.

La página que se ha escogido para nuestra aplicación es una página Web muy conocida por nosotros, (éste ha sido el motivo de la elección de la página en concreto) la Web del mail de los alumnos de la UAB (ver Figura 12).

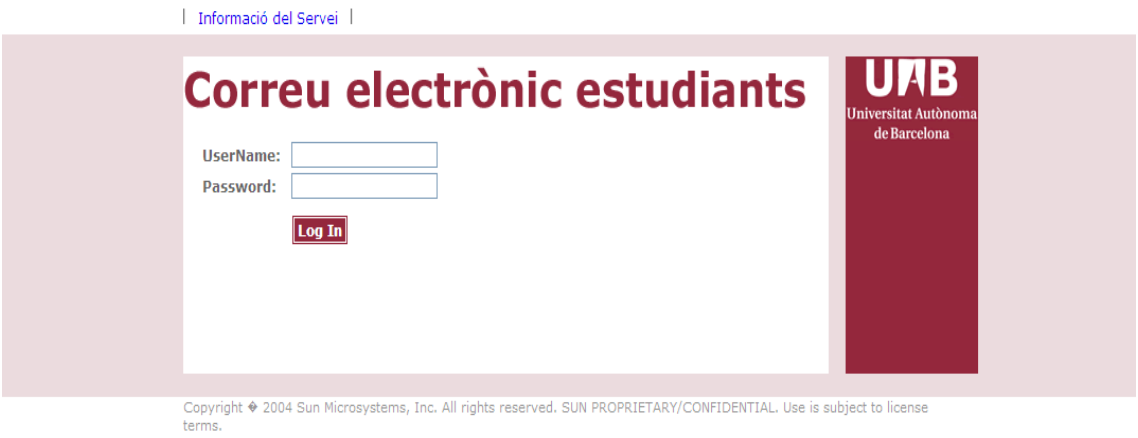


Figura 12. Webmail alumnos UAB

4.4. Cliente Web

La programación del cliente web se podría dividir en 3 partes:

- La interfaz gráfica, creada en XUL.
- El script que da funcionalidad a la aplicación, en JavaScript.
- El componente XPCOM, hecho en C++.

La primera parte, no es solo la interfaz gráfica, es crear todos los archivos y jerarquía de directorios para que nuestro cliente Web, Firefox, entienda que hemos creado una extensión y la instale.

Para empezar con esta tarea, crearemos un archivo, allí donde está ubicado el directorio perteneciente a las extensiones instaladas en nuestro cliente Web, que suele ser un directorio parecido a éste: "`C:\Documents and Settings\usuario\ Datos de programa\ Mozilla\ Firefox\ Profiles\ 83x0s50h.default\extensions`", ese archivo lo hemos llamado `BluetoothExtension@foo.net` y contiene la dirección de nuestro PC donde vamos a crear la extensión. Es una forma de ahorrar tiempo a la hora de navegar por los directorios y a la vez es necesario crear un archivo como el nuestro, ya que si no, Firefox no detectaría que hay una nueva extensión instalada.

En el directorio donde vamos a crear la extensión, tendremos una jerarquía tal que en el directorio principal vamos a tener dos archivos, el `install.rdf` y el `chrome.manifest` y 3 directorios, `content`, `skin` y `components`.

En el directorio `skin`, van los archivos CSS que son los encargados del estilo de la aplicación. Con ellos podemos cambiar colores y alineaciones a nuestro gusto, tal y como se explicó en el apartado de tecnologías. En nuestra extensión no se ha utilizado ningún estilo, se ha preferido dejar todo por defecto.

En el directorio `components`, van los archivos correspondientes a la creación de componentes XPCOM, ya sea en JavaScript o en C++. En el punto 4.4.1 se explicará la creación de componentes más detalladamente.

Antes de explicar el contenido del directorio `content`, se hablará de los dos archivos que tenemos en el directorio principal, el `install.rdf` y el `chrome.manifest`.

El archivo `install.rdf` es el archivo que una aplicación XUL usa para determinar información sobre cómo tiene que ser instalada esa aplicación. Contiene **metadatos**[19] identificando la extensión, quien la creó, donde se puede encontrar la información referente a la extensión, que versiones de aplicaciones con las cuales es compatible, etc. En la Figura 13 podemos ver el archivo `install.rdf` para nuestra aplicación, cabe destacar los atributos `id`, para identificar donde están los datos de la extensión, el `type`, que se ha puesto un 2 ya que es el número que identifica a las extensiones. El atributo `name` y `description` serán los que nos aparezcan una vez tengamos instalada la extensión en el menú "`Complementos`" dentro del menú "`Herramientas`" de nuestro navegador, ver Figura 14.

El otro archivo que nos concierne, el `chrome.manifest`, es el encargado de decirle al registro chrome, que hay un nuevo chrome disponible. Nosotros necesitamos registrar nuestro archivo XUL que está en el directorio `content` y lo haremos de la siguiente manera:

```
content     bluextension     chrome/content/
```

Donde el primer parámetro es el tipo de directorio, el segundo es el nombre de nuestro paquete y el tercero es la **uri** [20] hacia nuestros archivos.

[19] Datos que describen otros datos.
[20] Uniform Resource Identifier (Identificador uniforme de recurso).

Con esto le decimos al navegador donde están los archivos para el registro chrome, pero lo que nosotros queremos es superponer nuestro archivo XUL a nuestro navegador, esto se consigue de la siguiente manera en el archivo `chrome.manifest`:

```
overlay chrome://browser/content/browser.xul
chrome://bluextension/content/bluextension.xul
```

```
<?xml version="1.0"?>

<RDF xmlns="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:em="http://www.mozilla.org/2004/em-rdf#">

  <Description about="urn:mozilla:install-manifest">
    <em:id>BluetoothExtension@foo.net</em:id>
    <em:version>1.0</em:version>
    <em:type>2</em:type>

    <!-- Target Application this extension can install into,
         with minimum and maximum supported versions. -->
    <em:targetApplication>
      <Description>
        <em:id>{ec8030f7-c20a-464f-9b0e-13a3a9e97384}</em:id>
        <em:minVersion>1.0+</em:minVersion>
        <em:maxVersion>2.0.0.*</em:maxVersion>
      </Description>
    </em:targetApplication>

    <!-- Front End MetaData -->
    <em:name>Bluetooth Extension!</em:name>
    <em:description>Gestor de contraseñas en dispositivo móvil accesible por bluetooth</em:description>
    <em:creator>Sergio Laguna Garcia</em:creator>
    <em:homepageURL>http://www.foo.com/</em:homepageURL>
  </Description>
</RDF>
```

Figura 13. Archivo install.rdf

Lo que estamos diciendo con la instrucción overlay es que superponga a la interfaz del navegador (`browser.xul`) nuestro archivo, `Bluextension.xul`, podemos ver el resultado en la Figura 15.

En el directorio, `content`, se hallarán los archivos de la interfaz gráfica (XUL) y los de funcionalidad (**JS**[21]). El archivo escrito en XUL se encarga de crear la interfaz gráfica de nuestro navegador, que junto al archivo `chrome.manifest` da el resultado que vemos en la Figura 15. Al clicar en la nueva entrada en el menú "`Herramientas`" se da inició a la funcionalidad de la extensión.

[21] Javascript

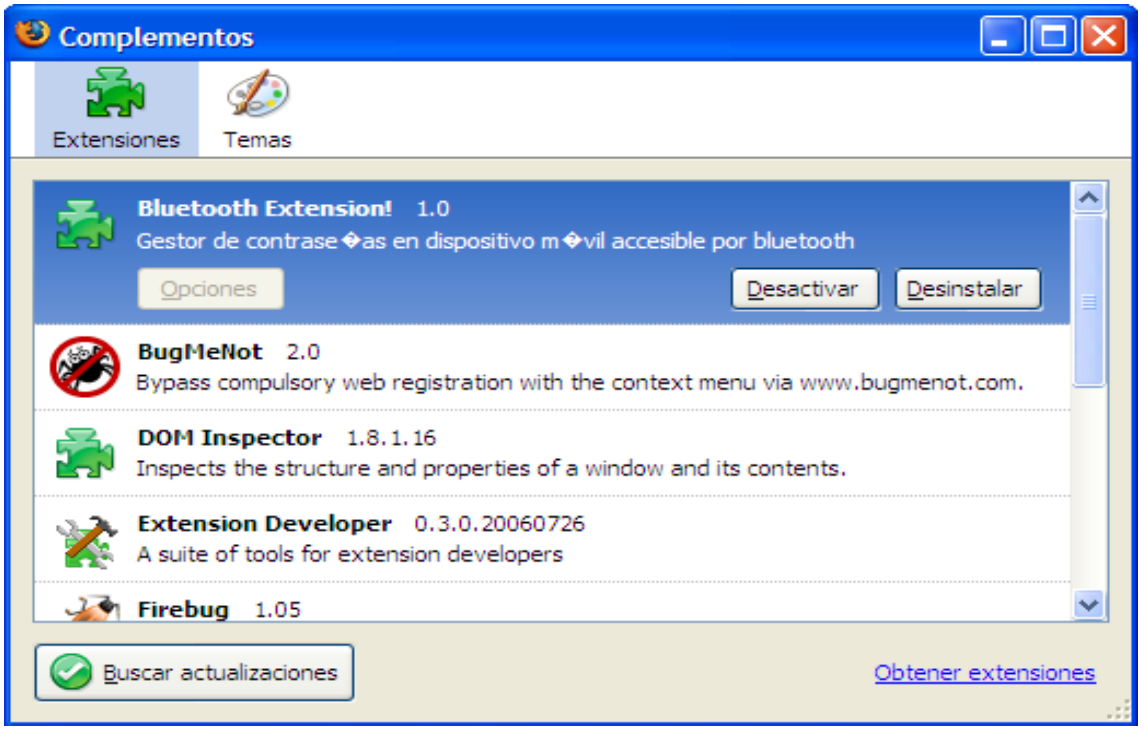


Figura 14. Extensión instalada

El archivo JavaScript contiene básicamente una función llamada `ProcesarCampos`, que es la encargada de realizar todo el proceso de verificación. Éste proceso consiste en ver si la página en la cual estamos en ese mismo instante contiene un formulario de login/password válido, es decir, que si la web no contiene un formulario o contiene uno que no es del tipo http/https, nos "saltará" una advertencia por pantalla diciendo que la web no contiene un formulario de login/password válido. Hemos hecho una prueba para ver lo sucedido en una página cualquiera y hemos tomado como ejemplo una web de Mozilla que nos habla de XPCOM, ver Figura 16. Como se puede observar, al no tener un formulario válido, nos aparece la advertencia por pantalla.

Si por el contrario, nos encontramos ante un formulario válido, la función `ProcesarCampos` llamará a otra función llamada `getEditableInputs`, que se encarga de almacenar las entradas editables de un formulario, que nos servirá posteriormente para rellenar el campo de la contraseña del mismo. Si

nuestra variable donde guardamos las entradas editables del formulario se llama formulario, para entrar un valor se debe llamar al atributo "`value`", por ejemplo, con `formulario.value[0]` rellenaríamos el campo de la primera entrada editable del formulario, que usualmente corresponde con la del nombre de usuario y con `formulario.value[1]` rellenaríamos el campo de la segunda entrada editable, correspondiéndose normalmente con el de la contraseña.

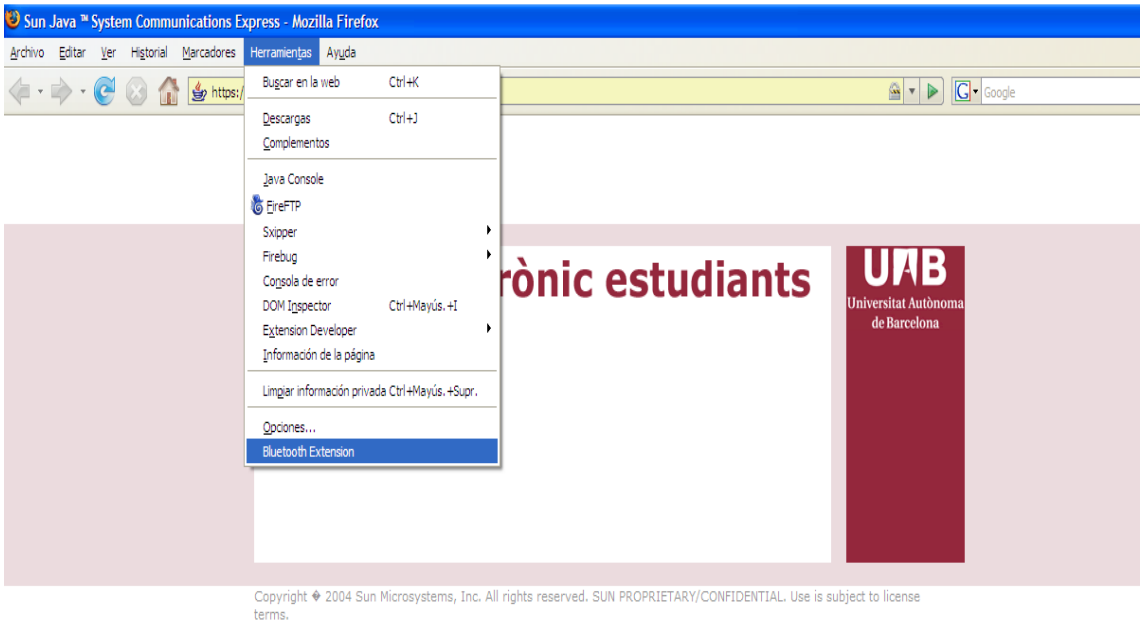


Figura 15. Nueva entrada en el menú Herramientas

Después de llamar a la función `getEditableInputs`, lo que hay que iniciar es la comunicación vía Bluetooth con el dispositivo móvil. Desde JavaScript no podemos hacer esto, por las limitaciones comentadas en el apartado de tecnologías, entonces necesitamos crear un componente XPCOM en C++ que sea capaz de comunicarse con el dispositivo móvil y recibir los datos. Una vez creado el componente, lo podremos instanciar desde JavaScript utilizando XPConnect y utilizar su servicio para poder comunicarnos mediante Bluetooth con el móvil.

Figura 16. Alerta de formulario no válido

4.4.1. Creación del componente XPCOM en C++

Para empezar a crear un componente XPCOM en C++ hay que tener instalado el Gecko SDK. El Gecko SDK es un conjunto de archivos **XPIDL**[22], cabeceras y herramientas para desarrollar componentes XPCOM para poder acceder a ellos desde Javascript.

Lo primero que tenemos que crear es un **GUID** [23] para nuestra interfaz con la herramienta **guidgen**[24] de Windows. En la Figura 17 se puede ver la plantilla para crear la interfaz que rellenaremos con la GUID creada y los métodos y atributos que queramos. En nuestro caso solo habrá un método que se llamará conexión que nos devolverá el string correspondiente a la contraseña que estará almacenada en el dispositivo móvil.

[22] Lenguaje de descripción de interfaces multi plataforma.
[23] Globally Unique Identifier (Identificador Global Único).
[24] Herramienta incluida en Windows para generar GUIDs.

```
#include "nsISupports.idl"

[scriptable, uuid(5F05DFE8-0A5E-4e44-B27F-8EA59A3BEDB2)]
interface nsIBlueComp : nsISupports
{
  string connexion();
};
```

Figura 17. Plantilla para crear la interfaz

Con la herramienta xpidl que viene con el Gecko crearemos el archivo de cabecera (`.h`) y el archivo `typelib` (`.xpt`) de la interfaz creada. El primero nos servirá como plantilla de cabecera e implementación para crear los archivos **CPP**[25] donde irán la implementación de la conexión y la creación del módulo. El segundo, junto con el archivo **DLL**[26] que crearemos al finalizar el proceso de creación del componente, serán los archivos necesarios para registrar el componente en el navegador y poder instanciar la interfaz recién creada a nuestro antojo.

Con el archivo de cabecera (`.h`) creado, usaremos la parte referente a la cabecera, como plantilla para crear nuestra cabecera real. A esto le tendremos que añadir unas líneas de código para identificar a nuestro componente (ver Figura 18).

```
#define BlueComp_CONTRACTID "@laguna.com/BlueComp;1"
#define BlueComp_CLASSNAME "XPCom comunicación Bluetooth"
//{42949D8D-1543-4ca9-A3FE-33E768F65C31}
#define BlueComp_CID  { 0x42949d8d, 0x1543, 0x4ca9, { 0xa3, 0xfe, 0x33, 0xe7, 0x68, 0xf6, 0x5c, 0x31 } }
```

Figura 18. Archivo de cabecera

[25] Archivos de implementación del lenguaje C++.
[26] Dinamic Link Library (Librería de conexión dinámica)

El primer `define`, define como instanciar a nuestro componente XPCOM desde Javascript. El segundo es una pequeña definición y el tercero es otro identificador único.

Antes de empezar con la implementación de la conexión, tendremos que crear otro archivo `CPP` que será el encargado de generar el componente cuando lo instanciemos. En la Figura 19 se puede ver cómo quedó el archivo en cuestión.

```
#include "nsIGenericFactory.h"
#include "BlueComp.h"

NS_GENERIC_FACTORY_CONSTRUCTOR(BlueComp)

static nsModuleComponentInfo components[] =
{
    {
       BlueComp_CLASSNAME,
       BlueComp_CID,
       BlueComp_CONTRACTID,
       BlueCompConstructor,
    }
};

NS_IMPL_NSGETMODULE("MiModuloComponente", components)
```

Figura 19. Archivo del Módulo.

Ahora toca crear el otro archivo `CPP` donde irá la implementación de la conexión. Como se ha hecho antes, cogemos la plantilla de cabecera creada y con la parte de la implementación crearemos el archivo `CPP`, al cual, solo tendremos que añadir la implementación del método `connexion` creado en la interfaz.

Para empezar la implementación hemos tenido que instalar el Platform **SDK** [27] de Microsoft, ya que contiene las librerías necesarias para realizar la comunicación por Bluetooth, tal y como se ha explicado en el apartado de API C++ para Bluetooth del capítulo de las tecnologías.

La implementación de la conexión se empieza llamando a la función `WSAStartup`, que es la encargada de iniciar la `DLL` WinSock por un proceso. Se crea el socket, con los parámetros necesarios para realizar la comunicación

[27] Software Development Kit

Bluetooth y ahora tenemos que pasar la dirección física de nuestro dispositivo móvil, que está en el formato "xx:xx:xx:xx:xx:xx" como string, a una dirección Bluetooth. Una vez acabo esto, hay que crear la estructura SOCKADDR_BTH (sab en nuestro caso) para contactar con nuestro dispositivo remoto. En la Figura 20 se puede ver como ha quedado esta estructura.

```
sab.addressFamily = AF_BTH;
sab.btAddr =  ululRemoteBthAddr;    //dirección Bluetooth del dispositivo remoto
sab.serviceClassId = SerialPortServiceClass_UUID;
sab.port = 25;
```

Figura 20. Estructura SOCKADDR_BTH

Acto seguido lo que tenemos que hacer es conectar con el dispositivo remoto, esto se consigue con la función connect, pasándole como parámetro la estructura que recién se ha creado. Una vez hecho esto, la conexión se habrá establecido y solo nos quedará recibir la contraseña procedente del dispositivo móvil. Para conseguirlo es necesario llamar a la función recv, añadiendo a uno de los parámetros un buffer donde almacenar los datos. Después de la llamada a dicha función, ya tendremos los datos almacenados en una variable listos para ser devueltos como valor de retorno de la función. Solo hace falta una cosa más, cerrar el socket creado con la función closesocket y ya estará el asunto resuelto.

Ahora que ya tenemos todos los archivos en condiciones solo nos hace falta crear la DLL. Para ello tendremos que abrir un proyecto en el Visual Studio con la opción de crear una Win32 Dinamic-Link Library. Se añaden todos los archivos al proyecto y se construye la DLL.

Una vez tenemos la DLL construida solo tenemos que añadirla, junto con el archivo XPT creado al principio de todo el proceso de creación del componente, al directorio components de nuestro navegador, que suele estar en la ruta "C:\Archivos de programa\Mozilla Firefox\components".

Con el componente ya creado, se puede proseguir en la creación de la aplicación.

Si recordamos, se había llamado a la función `getEditableInputs` para poder rellenar el campo de la contraseña, pues bien ahora hay que instanciar al componente XPCOM. En la Figura 21 se puede ver un ejemplo de cómo se ha logrado esto. Primero se activa el privilegio de poder usar XPConnect, luego se crea una instancia del componente y finalmente se llama al método deseado, que en este caso es el método `connexion`, para que nos devuelva un string con la contraseña deseada.

```
netscape.security.PrivilegeManager.enablePrivilege("UniversalXPConnect");
var oMyComponent =Components.classes['@laguna.com/BlueComp;1'].createInstance(Components.interfaces.nsIBlueComp);
if (oMyComponent == null) {
    alert("oMyComponent is null");
}
var pass = oMyComponent.connexion();
```

Figura 21. Instanciar Componente.

Una vez acabado este proceso, solo hay que rellenar el campo de la contraseña devuelto por la función `getEditableInputs` con el string devuelto por el método `connexion` del componente XPCOM. En las Figuras 22 y 23 se puede ver el resultado de llamar a la extensión desde la página del Webmail de los alumnos de la UAB y como nos da permiso el servidor Web para entrar al área restringida.

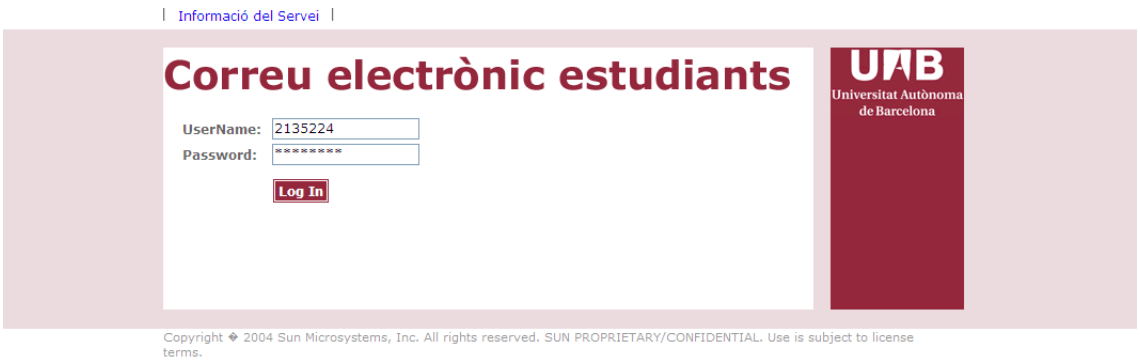


Figura 22. Formulario después de iniciar la extensión

Figura 23. Acceso permitido

4.5. Cliente Móvil

Para la parte del cliente móvil se ha utilizado el **IDE**[28] de desarrollo NetBeans, con un paquete adicional de "movilidad", necesario para trabajar sobre dispositivos móviles, crear Midlets y muy útil ya que nos provee de emuladores para probar nuestros programas sin tener que instalarlos en ningún dispositivo.

El entorno de desarrollo no contiene librerías para trabajar sobre la tecnología Bluetooth, así que lo primero que hay que hacer es bajarse una librería para poder empezar a crear el Midlet.

Se decide usar `Bluecove`, que provee una implementación del protocolo JSR-82, que es el que necesitamos. Solamente tenemos que incluir al archivo `JAR` que acabamos de descargar en el **path**[29] de nuestro IDE y ya estaremos listos para usar la librería.

Ahora tenemos que crear un nuevo proyecto, abrimos dentro de la categoría "`mobility`", la parte de creación de una aplicación MIDP. Ahora salen unas opciones que dependiendo del dispositivo móvil se escogen unas u otras. En el caso de este proyecto se han escogido `CLDC 1.1` y `MIDP 2.0`. Ahora si ya se está listo para empezar a crear el Midlet.

Dentro del paquete se ha de crear un "`Visual Midlet`" para ir creando los elementos visualmente, sin tener que picar código, aunque no todo se podrá hacer "visualmente".

Creamos un nuevo formulario que será la pantalla de bienvenida cuando ejecutemos la aplicación. Como se puede ver en la Figura 24, se ha puesto como titulo del formulario "Servidor esperando petición", ya que cuando se inicie la aplicación, el dispositivo ya estará listo para dar la contraseña. También se ha añadido la opción de salir de la aplicación mediante un

[28] Integrated Development Environment (Entorno de Desarrollo Integrado)
[29] Ruta o forma de referenciar un archivo.

comando de “Exit” que al apretar la tecla correspondiente del dispositivo móvil, saldremos de la aplicación.

Figura 24. Formulario inicial de la aplicación

Antes de crear el formulario visual, hay que hacer una serie de cosas. Primero tenemos que conseguir el objeto `Dispositivo local` para nuestro dispositivo Bluetooth local, eso se consigue con la función `getLocalDevice`. Segundo tenemos que hacer que nuestra aplicación sea visible para otras aplicaciones, para eso hace falta llamar al método `setDiscoverable`, que nos dice si se ha llevado a cabo bien la petición. Ahora hay que crear una URL, con su identificador único y otros parámetros como “`localhost`” diciendo que estamos en la parte servidora, para que otros dispositivos se puedan conectar a nosotros a través de ella. Finalmente, se llamará al método

`Connector.open` para crear un **stream**[30] de entrada con la URL creada anteriormente.

Después de hacer un **`display`**[31] del formulario de inicio, el programa se quedará esperando a que le llegue una petición, esto se consigue con un bucle infinito. Dentro de este bucle, estará el método `acceptAndOpen` que devolverá un objeto `streamConnexion` que representará la conexión por socket del lado del servidor.

Acto seguido, se lanzará un hilo que será el encargado de realizar el proceso de enviar la contraseña al cliente.

Al lanzarse el hilo, significará que ya se ha recibido una petición, y se mostrará por la pantalla del dispositivo un mensaje confirmándolo, ver Figura 25.

Ahora solo tenemos que crear un stream de salida llamando al método `openDataOutputStream` y con el método `write` enviar la contraseña por ese stream de salida.

Finalmente, solo nos quedará por hacer, cerrar la conexión creada y el stream de salida.

[30] flujo
[31] Método para mostrar un formulario por la pantalla del dispositivo móvil.

Figura 25. Formulario de petición recibida

4.6. Hilo de la implementación

Lo primero que se empezó a implementar fue todo lo relacionado con la interfaz gráfica, se creó la jerarquía de directorios y los archivos necesarios para que el navegador entendiera que había una nueva extensión instalada, luego se pasó a crear el archivo XUL en sí, para añadir el nuevo elemento en menú herramientas.

Lo segundo, fue crear el archivo Javascript, que detectará si en la web en la que nos encontramos hay un formulario válido y si es así recoger las entradas editables de dicho formulario para ser rellenadas posteriormente.

Después de lo realizado anteriormente, se pasó a crear el programa del dispositivo móvil, pero sin llegarlo a instalar, solamente se hacía funcionar en el emulador del NetBeans.

Entonces, se intentó crear el programa que conectara con el dispositivo en C++, pero como veremos en el apartado de problemas encontrados, no se pudo inicialmente y se pasó a probar otra opción. Esta opción fue crear el mismo programa pero en Java. Con la librería usada para crear el programa del dispositivo móvil fue bastante fácil crear el programa.

Para probar el programa con el servidor que ya teníamos creado, teníamos que crear el programa de conexión en forma de Midlet para poder simularlo y realizar la conexión entre los dos simuladores, el servidor y el cliente, para ahorra tiempo de instalación.

En cuanto ya se producía la conexión y se recibían los datos correctamente, se pasó a crear el componente XPCOM mediante la tecnología JavaXPCOM, pero como también se podrá ver en el apartado de problemas encontrados, esto era una opción inviable y tuvimos que pensar otras opciones.

Se volvió a pensar en crear el programa en C++ y tras muchos problemas se acabó consiguiendo. Ahora si podíamos crear un componente XPCOM en C++, ya que esto sí que era una opción viable.

Después de crear el componente XPCOM en C++ solo había que volver al archivo Javascript y desde ahí instanciar el componente para que realice la conexión con el dispositivo móvil y con el resultado de ésta conexión rellenar los campos del formulario automáticamente.

5. Conclusiones

El lenguaje XUL ha demostrado ser muy fácil de utilizar y a la vez muy potente, capaz de crear interfaces gráficas con unas pocas líneas de código y cualquiera podría adentrarse en él, ya que en unos pocos días seremos capaces de trabajar con él habiendo empezado desde cero.

Pero para crear una extensión no solo nos va hacer falta saber XUL, a menos que hagamos una extensión muy, muy simple. Si queremos añadir funcionalidad tendremos que saber algo de Javascript, lenguaje que a lo mejor no es tan fácil de aprender como XUL pero sí que es bastante fácil hacerse con él. Con XUL y Javascript ya seremos capaces de crear extensiones de mayor complejidad, que nos sean de utilidad y sin mucho esfuerzo. Esto, hace que existan miles de extensiones creadas por usuarios que son enormemente útiles. Esto es una de las cosas por las que el navegar Firefox es tan popular entre la comunidad programadora.

Si queremos hacer cosas todavía más complejas, tendremos que recurrir a código nativo, ya que en Javascript no podemos hacer por ejemplo, una comunicación vía Bluetooth con un dispositivo móvil. Para realizar esto, tendremos que crear componentes XPCOM en C++ para añadir más funcionalidad a nuestras extensiones. Crear un componente no es tan sencillo como podría serlo XUL o Javascript, pero existen tutoriales muy buenos que nos guían paso a paso en la creación del mismo. Una vez ya hemos creado uno, crear otro ya no tendrá más misterio y se podrán crear extensiones enormemente más potentes, gracias a la tecnología XPCOM.

Gracias a todo esto, es posible crear extensiones muy potentes para nuestro navegador, que con un poco de imaginación, podemos llegar a crear cosas verdaderamente provechosas para el usuario, como puede ser la realizada en éste proyecto.

En cuanto a la programación de Midlets, también es una cosa muy provechosa ya que se pueden hacer virguerías con el dispositivo móvil. Si se tienen conocimientos previos de Java, nos será muy fácil coger la dinámica de J2ME y si no la tenemos puede ser que nos cueste algo más, pero tampoco mucho ya que Java es un lenguaje muy intuitivo y que se le coge el hilo bastante rápido. En este proyecto se ha creado un Midlet que simplemente habría un canal de comunicación por el cual enviaba una contraseña, pero como se ha podido ver en diferentes foros, se le han visto múltiples utilidades y de mucha potencia para nuestro dispositivo móvil.

La parte del Midlet ha sido una parte muy provechosa, como todo el proyecto, ya que se ha aprendido a crear un Midlet desde cero y ya se está pensando en crear otras aplicaciones para el dispositivo móvil.

5.1. Problemas encontrados

Han sido bastantes los problemas encontrados a lo largo de toda la implementación y por consiguiente vamos a dividir los problemas encontrados en tres partes que serán por un lado la parte de creación de todo lo relacionado con Javascript, la creación del componente XPCOM y finalmente la creación del Midlet en el dispositivo móvil.

En lo referente a la primera parte, la creación de la validación del formulario de login/password, instanciar el componente XPCOM y el relleno automático del formulario, decir que la instanciación del componente XPCOM no supuso ningún problema, en cambio las otras dos si dieron problemas.

Para validar un formulario en una web primero teníamos que encontrar dicho formulario, para hacerlo, tenemos que recorrer los **tags**[32] html hasta encontrar un tag `input` que nos lleve hasta un tag `form`, entonces sabremos que esa web contiene un formulario. Pero no se era capaz de "navegar" por esos tags, sino que se navegaba por los tags de la interfaz del navegador. Entonces aquí surgió el primer problema, cómo recorríamos los tags de la web en la que nos encontrábamos y no la interfaz del navegador?

La respuesta fue llamando a un objeto del DOM que nos adentraba en el contenido de la web en la que nos encontrábamos, el "`window._content`". Así ya éramos capaces de recorrer los tags de la web para poder así validar el formulario de login/password.

El otro problema fue el saber acceder a los campos del formulario una vez ya lo teníamos validado, pero gracias a una extensión descargada, se supo cómo hacer esto y se creó una función adaptada a la de la extensión descargada. No fue un problema tan grande como el anterior, pero se tardaron algunos días en poder solucionarlo.

[32] etiquetas

Ahora pasamos a la parte de creación del Midlet, para seguir el orden en que se implementaron las cosas.

Aquí el primer problema fue encontrar una librería que nos permitiera trabajar con la tecnología Bluetooth desde Java o J2ME. Buscando por diversos foros se encontró `Bluecove`, que funciona a las mil maravillas. Después de resolver el tema de la librería, se empezó a crear el Midlet, cosa que no se había hecho nunca por mí y por lo tanto no se sabía por dónde empezar. La creación de los formularios no supuso muchos problemas pero si el crear el resto del programa. Primero fue la creación de una URL para que otros dispositivos nos pudieran detectar, había que poner varios parámetros específicos por tratarse de una conexión por Bluetooth y por estar del lado del servidor. Estos parámetros supusieron problemas que se solucionaron como no, en los diferentes foros relacionados con J2ME y la comunicación Bluetooth. Todo lo demás fue rodado excepto un pequeño detalle a la hora de enviar la contraseña por el stream de salida, al final hay que cerrar la conexión creada y el stream de salida creado.

Ahora viene la parte de la creación del componente XPCOM, que con diferencia es la parte que más problemas dio de toda la implementación.

Para empezar, antes de crear el componente se pensó en crear un programa que creará una conexión Bluetooth y recibiera datos, así sería más fácil de comprobar los fallos. Se utilizó un IDE como es Visual Studio para tal efecto y se empezó a crear el programa.

El programa escrito en C++, no se era capaz de hacerlo funcionar, aunque todo parecía indicar que el programa era correcto, era un misterio que no se sabía cómo resolverlo.

Entonces se pensó en otra solución y se vio que en la web de desarrollo de Mozilla había una solución para crear componentes en Java, la solución se llamaba JAVAXPCOM y prometía mucho, ya que se podría crear el programa en Java que sería mucho más fácil dado que ya teníamos creado el MIDlet en

J2ME y en teoría no debería ser muy diferente. Y así fue, tal y como se había pensado no fue muy difícil crear el programa en el IDE NetBeans y comunicarlo con el MIDlet, el problema vino a la hora de crear el componente en XPCOM en Java.

Leyendo la información que había sobre el tema parecía que si incrustabas el Mozilla en la aplicación Java si podías crear el componente para que se comunicaran entre ellos, pero no se quería hacer eso, ya que lo que se quería es lanzar la aplicación desde el navegador, no desde una aplicación Java. Y eso no era posible hacer, no es posible lanzar código escrito en Java desde una extensión, como bien se ha comentado en este foro (`osdir.com/ml/mozilla.devel.netlib`). Sin embargo, sí que existe una manera de lograrlo, dándonos unos permisos que en principio no tenemos para así poder llamar al código Java tal y como se muestra en la extensión "Java_Firefox_Extension", pero se descartó la opción por la complejidad y por no parecer muy lícito.

En la parte de intentar la creación del componente en Java se perdió mucho tiempo y se pensó entonces en volver al inicio e intentar resolver los problemas que tenía el programa escrito en C++.

Los problemas se resolvieron, no sin varias consultas en foros, resultando ser que la versión del SDK que se tenía instalada no era la correcta, se tenía que tener instalada una versión anterior.

Pero sólo con esto no se solucionó el problema, había otro problema que solucionar. El dispositivo Bluetooth que se ha usado para este proyecto, viene con unos drivers que se instalaron correctamente. Pues éste era el problema, había que desinstalar los drivers y dejar que Windows instala los suyos propios automáticamente. Y así fue, en cuanto Windows instaló los drivers, el programa en C++ pasó a funcionar perfectamente.

Ahora tocaba la creación del componente XPCOM en C++, que siguiente este tutorial, muy recomendado, no se tuvo mayores problemas excepto en el valor

de retorno del componente, que tenía que ser la contraseña en cuestión. En la web de desarrollo de Mozilla y más concretamente en el apartado de creación de componentes XPCOM en C++ hay una guía de cómo usar los strings, que se siguió y se consiguió solucionar el último problema que se tuvo.

5.2. Trabajo futuro

Se pueden hacer muchas mejoras al programa, con un poco más de tiempo se puede llegar a hacer de esta extensión una extensión completísima lista para que otros usuarios puedan descargársela y usarla.

Para empezar solo se puede tener acceso a un formulario de login/password, que como se ha dicho anteriormente es el Webmail de los alumnos de la UAB. Una posible ampliación sería que el dispositivo móvil gestionara todas las webs que nosotros queramos y darnos la contraseña apropiada para una web dada. Esto se podría conseguir añadiendo una pequeña base de datos, que tendría como campos la página web y la contraseña, se presentaría como un formulario donde el usuario rellenaría con los campos mencionados. Entonces al activar la extensión el dispositivo móvil buscaría en su base de datos la página web en la que nos encontramos y si la encuentra dar la contraseña específica para esa web.

Otra posible mejora también relacionada con el dispositivo móvil, es que al iniciar la aplicación, ésta se queda esperando sin que nosotros podamos realizar otro tipo de operación con nuestro dispositivo. La mejora propuesta sería esa, hacer que al iniciar la aplicación se quede funcionando pero en la pantalla de inicio del dispositivo, haciendo que podamos hacer otras cosas con nuestro móvil.

6. Bibliografía

No existe Bibliografía en sí, toda la información ha sido extraída de fuentes Web.

- `es.wikipedia.org`: es una enciclopedia libre.
- `java.sun.com`: página oficial de Sun MicroSystems.
- `ganttproject.biz/webstart.php`: página para crear diagramas de Gantt.
- `developer.mozilla.org/En`: web de desarrollo de Mozilla.
- `mozilla.org`: Web principal de Mozilla.
- `iosart.com/firefox/xpcom`: excelente guía sobre cómo crear componentes XPCOM.
- `xulplanet.com`: excelente web para todo lo relacionado con XUL.
- `lcc.uma.es/~galvez/J2ME.html`: página web con información sobre dispositivos móviles.
- `bluetooth.org/apps/content`: página principal de la tecnología Bluetooth.
- `forum.nokia.com`: excelente web con herramientas e información para la creación de aplicaciones en dispositivos Nokia.
- `msdn.microsoft.com`: página web de desarrollo de Microsoft.
- `wainu.ii.uned.es:8081/WAINU/canal-programación/tutoriales/java/tutorial-j2me.pdf`: buen tutorial sobre cómo crear aplicaciones J2ME en un dispositivo móvil con Netbeans y el paquete Mobility.
- `rational.com`: página oficial de la herramienta UML Rational Rose.

Sergio Laguna García

Bellaterra, 18 de Setiembre de 2008

RESUMEN

Este proyecto nace de la necesidad de dar más seguridad a nuestros datos cuando navegamos por Internet.
Se ha implementado una plug-in para el navegador Firefox de Mozilla, que detecta un formulario de login/password conocido y rellena el campo de la contraseña automáticamente.
La contraseña estará en nuestro dispositivo móvil y la comunicación entre el navegador y el dispositivo se hará mediante la tecnología Bluetooth.

RESUM

Aquest projecte neix de la necessitat de donar un grau més de seguretat a les nostres dades quan naveguem per Internet.
S'ha implementat un plug-in per el navegador Firefox de Mozilla, que detecta un formulari de login/password conegut y emplena el camp corresponent a la contrasenya automàticament.
La contrasenya estarà en el nostre dispositiu mòbil i la comunicació entre el navegador i el dispositiu es farà mitjançant la tecnologia Bluetooth.

ABSTRACT

This Project stems from the need to give more security to our data when we surf the Internet.
It has implemented a plug-in for the Mozilla Firefox browser, which detects a known login/password form and fills the password field automatically.
The password will be in our mobile device and the communication between the browser and the device will be using Bluetooth technology.