Capítulo 2

Implementación del servicio de interfaz de usuario.

1 Introducción al servicio de interfaz de usuario

El servicio de interfaz de usuario nace de la necesidad de resolver el problema que plantea la representación de información en dispositivos tan heterogéneos como los dispositivos multimedia que se encuentran en un hogar.

El proyecto en que nos encontramos se engloba dentro de un proyecto de mayor entidad consistente en la generación del software necesario para el funcionamiento de una pasarela residencial.

Una parte importante de este proyecto consiste en la necesidad de representar la información en todos y cada uno de los dispositivos que encontramos en el entorno de la pasarela residencial. Son muchos y distintos los dispositivos que podemos encontrar en un hogar digital (televisión, ordenadores, pdas, teléfonos móviles…), cada uno de los cuales presentan características muy distintas a la hora de mostrar la información al usuario.

Surge por tanto un grave problema para el desarrollador de aplicaciones para la pasarela residencial. Es necesario que cada una de las aplicaciones que se inserten dentro de la pasarela residencial, sea capaz de representar la información en cada uno de los dispositivos que puedan encontrarse dentro del entorno de la pasarela.

Para ello el diseñador de aplicaciones debería desarrollar y compilar su aplicación para cada uno de los sistemas operativos que puedan encontrarse en los dispositivos del hogar digital.

Esto hace que el desarrollo de aplicaciones para la pasarela sea muy costoso en tiempo y en esfuerzo.

Nace por tanto la necesidad de encontrar un mecanismo que abstraiga al programador de aplicaciones del problema de la representación de la información, y que éste problema sea centralizado en una aplicación que lo resuelva para todas y cada una de las aplicaciones de la pasarela residencial.

El servicio de interfaz de usuario pretende resolver el problema mencionado anteriormente, haciendo más simple en tiempo y esfuerzo el diseño de interfaces de usuario para los servicios que van a residir en la pasarela residencial, y que serán desplegados en una gran diversidad de dispositivos distintos.

Es necesario pues realizar una definición exhaustiva de las capacidades del servicio, y definir perfectamente la interfaz que el servicio presenta tanto al programador de aplicaciones como al usuario final, así como la interacción con ambos.

2. Características del servicio de interfaz de usuario

El objetivo final del proyecto es desarrollar un bundle para el Framework de OSCAR, que sea capaz de resolver el problema de la representación de la interfaz de usuario en diferentes dispositivos.

El servicio de interfaz de usuario será un servicio que se estará ejecutando permanentemente en el framework de OSCAR como un "bundle". Éste "bundle" realizará el registro del servicio de interfaz, haciendo posible por parte de las aplicaciones usar el servicio de interfaz de usuario para mostrar la interfaz de usuario de las distintas aplicaciones.

Queda fuera del alcance del proyecto, la gestión de la visión compartida en una misma pantalla, de distintas interfaces de usuario. Tampoco forma parte del proyecto la detección automática del tipo de interfaz que debemos mostrar en cada dispositivo. Este tipo de lógica debe implementarse en bundles separados para que puedan ser usados en el futuro por el servicio de interfaz.

Nuestro proyecto inicial mostrará las distintas "vistas" de una misma interfaz bajo petición del programador.

2.1 La interfaz con el programador.

El servicio de interfaz de usuario debe proveer al programador de aplicaciones de un mecanismo simple para poder generar la "visión" de la interfaz gráfica que deber tener de la aplicación el usuario final.

Además el servicio de interfaz de usuario debe tener algún mecanismo para recoger los eventos que se produzcan en la interfaz de usuario y notificarlos para su posterior tratamiento al programador.

Por último el programador debe poder acceder a todos y cada uno de los elementos que se representen en la interfaz gráfica. Para ello el servicio de interfaz gráfica deber proveer algún mecanismo para poder recuperar las referencias a los objetos que se representen por pantalla, para que el programador pueda modificarlos a su antojo.

2.2 La interfaz con el usuario.

El servicio de interfaz de usuario deber ser completamente transparente al usuario final de las aplicaciones de la pasarela.

El usuario final solicitará la ejecución de un determinado servicio a la pasarela residencial, y es el framework de OSCAR es el que solicitará la instalación del bundle "servicio de interfaz de usuario" como requisito imprescindible para poder ejecutar dicho servicio.

Una vez resueltas todas las dependencias necesarias para ejecutar la aplicación, se producirá una comunicación entre el bundle del servicio y el bundle del servicio de interfaz para que se produzca la representación en pantalla de la interfaz de usuario del servicio en cuestión que se esté ejecutando.

3. El entorno de desarrollo

Este proyecto se engloba dentro de un proyecto superior que se encarga de generar todo el software que se ejecutará en una pasarela residencial. Todos y cada uno de los proyectos que atienden a este fin son proyectos de "Software Libre" con licencias GPL.

Pretendemos con este proyecto difundir el uso de herramientas de software libre, por lo que se optó escoger como sistema de desarrollo un pc en el que se instala el sistema operativo Linux, más concretamente su distribución Debian Woody.

Como entorno de desarrollo Java se escoge el IDE Eclipse 3.0, con el que podíamos contar con una gran cantidad de herramientas de software libre como plugins para UML como OMONDO.

Como ya hemos comentado anteriormente, como framework OSGi escogimos la implementación de OSCAR, la cual está disponible para que sea instalada tanto en el sistema operativo Window XX o Linux.

Como la solución final del sistema operativo debía funcionar en cualquier dispositivo independiente del sistema operativo, se optó por generar una partición con el sistema operativo Window XP, en el que instaló en entorno de desarrollo Java JBuilder de Borland, y se instaló el framework de OSCAR.

Así se podría comprobar que el funcionamiento del servicio de interfaz es independiente del sistema operativo donde se ejecute.

4. Esquema general del funcionamiento del servicio

La figura 6. representa un diagrama de bloques que describe el funcionamiento, a grandes rasgos, del servicio de interfaz de usuario.

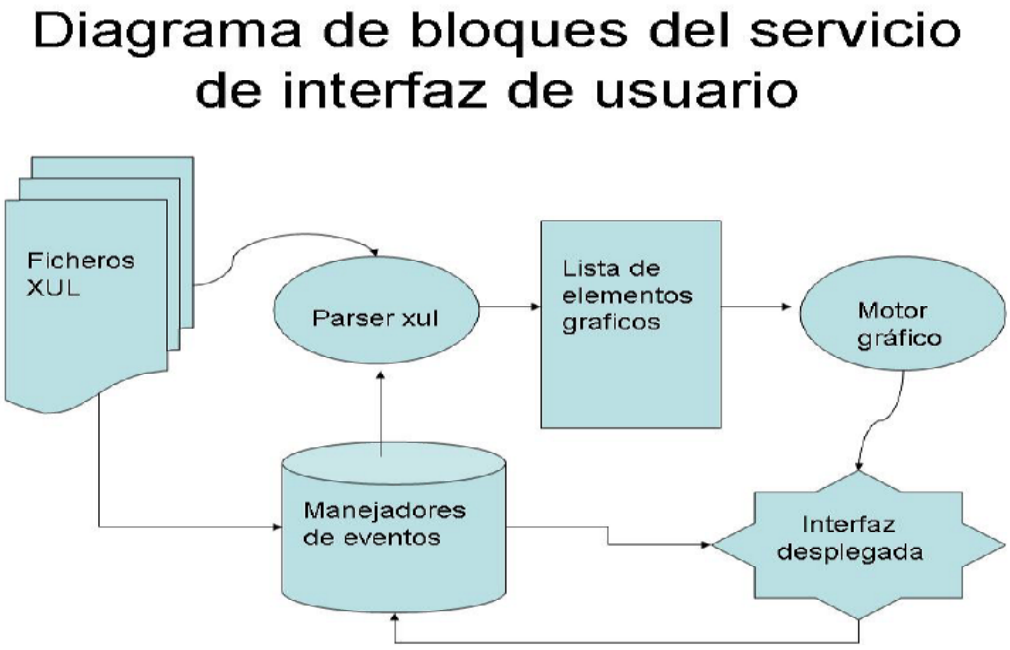


FIGURA 6. Diagrama de bloques del funcionamiento básico del servicio de interfaz de usuario.

Podemos realizar una descripción simplificada de la estructuración del servicio de interfaz de usuario:

- El programador del servicio describe su interfaz gráfica siguiendo un lenguaje "like xul" que describiremos en apartados posteriores.

- El servicio solicitará al servicio de interfaz de usuario que realice el parseado de los ficheros xul. Con esto conseguimos generar una lista de los "elementos gráficos", así como de sus atributos, que debemos representar en la pantalla.

- Una vez obtenida la lista de elementos que tenemos que dibujar, el servicio de interfaz de usuario, pasa esta lista al motor gráfico. Este motor gráfico es el que

se encarga de leer los atributos que debe tener cada uno de los elementos y representarlo convenientemente en la pantalla.

- Después de mostrar el contenido de la interfaz de usuario en la pantalla, comienza la gestión de la notificación de eventos por parte del servicio de interfaz de usuario al servicio en cuestión que la está usando.

4.1.El lenguaje de descripción de la interfaz de usuario

4.1.1. Introducción:

El lenguaje elegido para la descripción del servicio de interfaz de usuario ha sido el lenguaje XUL. Concretamente se ha generado un nuevo lenguaje basado en XUL, pero que usa sólo un pequeño conjunto de su potencialidad.

Llamaremos a nuestro lenguaje **LXUL** (Like XUL), para diferenciarlo del lenguaje del que procede (XUL) y del que sólo implementa un subconjunto de su potencialidad.

El lenguaje LXUL es un lenguaje basado en XML, con el que podemos realizar la descripción de una interfaz de usuario usando una serie de etiquetas definidas previamente.

Definimos como **"Widget"** a cada uno de los elementos independientes que pueden ser representados en la interfaz de usuario, tales como botones, etiquetas, contenedores verticales, etc.

Para este proyecto se han definido una serie de widgets elementales que permiten generar intefaces de usuario complejas.
Son muchos los "Widgets" que podrían complementar a los existentes, aunque la mayoría de ellos podría componerse usando la combinación adecuada de los ya existentes.

Son varias las simplificaciones que LXUL tiene con respecto a XUL:

- En XUL encontramos perfectamente separados la descripción estructural de la interfaz (elementos que la componen) y la descripción de la apariencia de esos elementos.

- En LXUL se usan atributos dentro de la descripción de la interfaz para generar la apariencia de la misma. Esto simplifica el proceso de representar los elementos por parte del motor gráfico. Esta solución ha sido adoptada por otros lenguajes basados en XUL como Thinlet o SwiXML.

- No se permite el uso de Scripts. Los manejadores de eventos en LXUL tienen que ser escritos en Java.

- No se permite el uso de etiquetas HTML. En el XUL de Mozilla el motor de renderizado Gecko, es capaz de aceptar etiquetas XUL y HTML indistintamente dentro de un mismo fichero de descripción interfaz de usuario.

4.1.2 Descripción de las etiquetas del lenguaje LXUL.

Para realizar la correcta definición de la interfaz de usuario de una determinada aplicación es necesario conocer en profundidad el lenguaje que vamos a emplear para describirla. En este apartado vamos a realizar la descripción de todas las etiquetas y los posibles valores de los atributos que hemos definido para formar nuestro lenguaje LXUL.

Veamos un ejemplo de la descripción de una interfaz de usuario con el lenguaje LXUL.

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

<?xml-stylesheet href="chrome://global/skin/" type="text/css"?>

<window id="example-window" title="Programador de riego V1.0"
      xmlns:html="http://www.w3.org/1999/xhtml"
      xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul">


        <menulist id="barra_menu_principal" >

                <menupopup id="file" text="Archivo" >

                        <menuitem id="nuevo" label="Nuevo..." />
                        <menuitem id="abrir" label="Abrir..." />
                        <menuitem id="guardar" label="Guardar"/>
                        <menuitem id="guardarcomo" label="Guardar Como" />
                        <menuitem id="salir" label="Salir" />
                </menupopup>

        </menulist>

</window>
```

Como podemos observar la estructura usada para definir la interfaz de usuario es exactamente igual a la usada en XUL. La única variación está en el tipo de etiquetas usasdas y en los atributos que tenemos en cada una de las etiquetas.

TIPO DE ETIQUETAS Y ATRIBUTOS

<table>
<tr><td rowspan="3">Nombre etiqueta:
<Window></td><td>Descripción: Etiqueta usada para describir ventanas en una interfaz de usuario.</td></tr>
<tr><td>Atributos:
• Id : Identificador de la ventana. Cada una de las ventanas que aparezcan en la interfaz debe tener un identificador diferente a los demás elementos que lo identifique unívocamente.
• Title: Título de la ventana.
• Sizex: Tamaño del eje x de la ventana.
• Sizey: Tamaño del eje y de la ventana.</td></tr>
<tr><td>Ejemplo:

<window id="example-window" title="Programador de riego V1.0"></td></tr>
<tr><td rowspan="3">Nombre etiqueta:
<hbox></td><td>Descripción: Etiqueta usada para describir paneles horizontales. En este tipo de panel podemos colocar cualquier tipo los demás elementos creados.</td></tr>
<tr><td>Atributos:
• Id : Identificador del panel horizontal. Cada uno de los paneles que aparezcan en la interfaz debe tener un identificador diferente a los demás elementos que lo identifique unívocamente.
• Hgap: Separación horizontal de los elementos.
• Sizex: Tamaño del eje x del panel.
• Sizey: Tamaño del eje y del panel.</td></tr>
<tr><td>Ejemplo:

<hbox id="contendorhorizont" hgap="2" sizex = "300" sizey ="200" >

<label id="etiqueta1" text="Widgets Contenedores" sizex = "250" sizey ="20" border="true" />

<button id="vbox" text="Vbox" color ="black" icon="D:/emilio jose/universidad/silla.gif" border="true" action="vbox" sizex="30" sizey="10" />

</hbox>

Panel horizontal que contiene una etiqueta y un botón.</td></tr>
<tr><td rowspan="2">Nombre etiqueta:
<vbox></td><td>Descripción: Etiqueta usada para describir paneles verticales. En este tipo de panel podemos colocar cualquier tipo los demás elementos creados.</td></tr>
<tr><td>Atributos:
• Id: Identificador del panel vertical. Cada uno de los paneles</td></tr>
</table>

	que aparezcan en la interfaz debe tener un identificador diferente a los demás elementos que lo identifique unívocamente. • **Hgap:** Separación vertical de los elementos. • **Sizex:** Tamaño del eje x del panel. • **Sizey:** Tamaño del eje y del panel.
	Ejemplo: *<vbox id="contendorhorizont" hgap="2" sizex = "300" sizey ="200" > <label id="etiqueta1" text="Widgets Contenedores" sizex = "250" sizey ="20" border="true" /> <button id="vbox" text="Vbox" color ="black" icon="D:/emilio jose/universidad/silla.gif" border="true" action="vbox" sizex="30" sizey="10" /> </vbox>* Panel horizontal que contiene una etiqueta y un botón.
Nombre etiqueta: <textbox>	**Descripción:** Etiqueta usada para describir campos de texto en una interfaz de usuario.
	Atributos: • **Id :** Identificador del panel horizontal. Cada uno de los paneles que aparezcan en la interfaz debe tener un identificador diferente a los demás elementos que lo identifique unívocamente.
	<textbox id="campotexto" /> Campo de texto con su correspondiente identificador.
Nombre etiqueta: <radiogroup>	**Descripción:** Etiqueta usada para describir paneles contenedores de radiobotones.
	Atributos: • **Id :** Identificador del panel contenedor de radiobotones. Cada uno de los paneles que aparezcan en la interfaz debe tener un identificador diferente a los demás elementos que lo identifique unívocamente. • **Hgap:** Separación vertical de los elementos. • **Sizex:** Tamaño del eje x del panel. • **Sizey:** Tamaño del eje y del panel.
	<radiogroup id="radiogroup1" hgap="20" sizex = "250" sizey ="30" > <radiobutton id="radiobutton1" text="Radio Button border="false" sizex="30" sizey="10" /> <radiobutton id="radiobutton2" text="Radio Button 2" border="true" sizex="30" sizey="10" /> </radiogroup> Panel contenedor de radiobotones con dos de ellos.
	Descripción: Etiqueta usada para describir radiobotones.
	Atributos:

<table>
<tr><td rowspan="2">Nombre etiqueta:
<radiobutton></td><td>

- **Id :** Identificador de radiobotones. Cada uno de los radiobotones que aparezcan en la interfaz debe tener un identificador diferente a los demás elementos que lo identifique unívocamente.
- **Text :** Etiqueta que se representa en el radiobotón.
- **Border:** Indica si el botón debe ser representado con un borde o sin el. Sus valores posibles son **true** o **false.**
- **Sizex:** Tamaño del eje x del radiobotón.
- **Sizey:** Tamaño del eje y del radiobotón.

</td></tr>
<tr><td>

<radiobutton id="radiobutton1" text="Radio Button 1" border="false" sizex="30" sizey="10" />

Ejemplo de radiobotón con todos sus atributos.

</td></tr>
<tr><td rowspan="3">Nombre etiqueta:
<menupopup></td><td>Descripción: Etiqueta usada para describir menús desplegables.</td></tr>
<tr><td>

Atributos:

- **Id :** Identificador del menú desplegable. Cada uno de los rmenús desplegables que aparezcan en la interfaz debe tener un identificador diferente a los demás elementos que lo identifique unívocamente.
- **Text :** Etiqueta que se representa en el menú desplegable.

</td></tr>
<tr><td>

<menupopup id="Ejemplo de Menu Desplegable" text="Ejemplo de Menu Desplegable" >

<menuitem id="label_e" label="menuitem1" />
<menuitem id="button_e" label="menuitem2" />

</menupopup>

Ejemplo de menú desplegable con dos ítems en su interior.

</td></tr>
<tr><td rowspan="3">Nombre etiqueta:
<menulist></td><td>Descripción: Etiqueta usada para describir barras de menús. En su interior se deben colocar menús desplegables con los campos que quieran ser representados en esta barra.</td></tr>
<tr><td>

Atributos:

- **Id :** Identificador de la barra de menús. Cada una de las barras de menús que aparezcan en la interfaz debe tener un identificador diferente a los demás elementos que la identifique unívocamente.

</td></tr>
<tr><td>

<menulist id="barra_menu_principal_ejemplo" >

<menupopup id="Widget Swing elementales" text="Widget Swing elementales" >

<menuitem id="label_ejemplo" label="Label..." />
<menuitem id="button_ejemplo" label="Button" />

</menupopup>

</td></tr>
</table>

	</menulist> Ejemplo de barra de menús con un menú desplegable en su interior.
Nombre etiqueta: <menuitem>	**Descripción:** Etiqueta usada para describir los ítems que se introducen en los menús desplegables.
	Atributos: • **Id :** Identificador del menuitem. Cada uno de los ítems que aparezcan en la interfaz debe tener un identificador diferente a los demás elementos que lo identifique unívocamente. • **label :** Etiqueta que se representa en el menuitem.
	<menuitem id="label_e" label="menuitem1" /> Ejemplo que representa un menuitem.
Nombre etiqueta: <listitem>	**Descripción:** Etiqueta usada para describir los ítems que se introducen en un listbox.
	Atributos: • **Id :** Identificador del listitem. Cada uno de los ítems que aparezcan en la interfaz debe tener un identificador diferente a los demás elementos que lo identifique unívocamente. • **label :** Etiqueta que se representa en el listitem.
	<listitem id="listitem1" label="listitem1" /> Ejemplo que representa un listitem.
Nombre etiqueta: <listbox>	**Descripción:** Etiqueta usada para describir menús de tipo lista.
	Atributos: • **Id :** Identificador del menú tipo lista. Cada uno de los rmenús tipo lista que aparezcan en la interfaz debe tener un identificador diferente a los demás elementos que lo identifique unívocamente. .
	<listbox id="listbox1" > <listitem id="listitem1" label="listitem1" /> <listitem id="listitem2" label="listitem2" /> </listbox> Ejemplo de menú desplegable con dos ítems en su interior.
Nombre etiqueta: <label>	**Descripción:** Etiqueta usada para describir etiquetas.
	Atributos: • **Id :** Identificador de etiquetas. Cada una de las etiquetas que aparezcan en la interfaz debe tener un identificador diferente a los demás elementos que la identifiquen unívocamente. • **Text :** Etiqueta que se representa en la etiqueta. • **Border:** Indica si la etiqueta debe ser representada con un

	borde o sin el. Sus valores posibles son **true** o **false.** • **Sizex:** Tamaño del eje x de la etiqueta • **Sizey:** Tamaño del eje y de la etiqueta.
	<label id="Etiqueta sin borde" text="Etiqueta sin borde" border="false" sizex="30" sizey="10" />
Nombre etiqueta: <image>	**Descripción:** Etiqueta usada para un panel conteniendo una imagen que se establece como fondo.
	Atributos: • **Id :** Identificador de la imagen. Cada una de las imágenes que aparezcan en la interfaz debe tener un identificador diferente a los demás elementos que la identifiquen unívocamente. • **Src :** Dirección de la imagen donde se alamacena la foto que queremos representar en el panel.
	<image id="imagen" src="file:///D:/ejemplos_xul/presentacion_swing/invierno.jpg" /> Ejemplo de una imagen que se encuentra en la dirección pasada como atributo.
Nombre etiqueta: <button>	**Descripción:** Etiqueta usada para describir botones.
	Atributos: • **Id :** Identificador de botones. Cada una de los botones que aparezcan en la interfaz debe tener un identificador diferente a los demás elementos que l identifiquen unívocamente. • **Text :** Etiqueta que se representa en el botón. • **Border:** Indica si el botón debe ser representada con un borde o sin el. Sus valores posibles son **true** o **false.** • **Sizex:** Tamaño del eje x del botón. • **Sizey:** Tamaño del eje y del botón. • **Color:** Color con el que se representa el texto del botón. • **Icon:** Icono que se representa en el botón. • **Action:** Nombre del método que va a manejar los eventos que se originan al pulsar el botón.
	<button id="Etiqueta sin borde" color ="red" icon="D:/emilio jose/universidad/stop.gif" action = "nulo" text="Boton sin borde" border="false" sizex="30" sizey="10" /> Ejemplo que muestra la descripción completa de un botón.
Nombre etiqueta: <filler>	**Descripción:** Etiqueta usada para describir rellenos entre elementos de la interfaz.
	Atributos: • **Id :** Identificador del relleno. Cada una de los rellenos que aparezcan en la interfaz debe tener un identificador diferente a los demás elementos que la identifiquen unívocamente. • **Sizex:** Tamaño del eje x del filler. • **Sizey:** Tamaño del eje y del filler.

	<filler id="relleno1" sizex="500" sizey = "50" /> Ejemplo de un elemento de relleno.

4.2. El concepto de Widget.

El objeto principal de la interfaz de usuario son los widgets. Definimos como widget a cada uno de los elementos fundamentales que componen una interfaz de usuario, como pueden ser etiquetas, botones, paneles, etc.

Como comentamos anteriormente, el objetivo primero de este proyecto, era generar la interfaz java del servicio de interfaz de usuario. Para ello uno de los primeros hitos que se marcaron en el proyecto fue definir la interfaz java que debía cumplir un widget que quisiera ser representado por parte de nuestro servicio interfaz de usuario.

Se realizó por tanto un estudio de las propiedades básicas que deben tener cada uno de los widgets, y se definió una interfaz java en la que definen los métodos que debe implementar cualquier widget que quisiera formar parte de la interfaz de usuario.

4.2.1 Interfaz Java de un Widget.

Para que un widget forme parte de nuestro servicio interfaz de usuario debe implementar la interfaz widget. Esta interfaz contiene los métodos básicos para que un widget pueda ser referenciado, añadido, modificado, y eliminado de la interfaz por parte del servicio de interfaz de usuario.

Para nuestro servicio de interfaz la interfaz java widget sirve como una "capa" sobre el elemento concreto que finalmente implementa el widget, y que nos permite actuar sobre el sin necesidad de conocer su implementación interna. Por ejemplo para implementar un botón en la "vista gráfica" se ha elegido la clase JButton del paquete swing. La definición de la interfaz de widget nos permite interactuar con este elemento sin necesidad de conocer cual es su implementación interna (JButton).

Esto nos da otra potencialidad importante. Si en cualquier momento necesitamos sustituir el elemento interno que implementa el widget(JButton), no tendríamos que modificar más que los métodos de la interfaz widget, ya que el resto se mantendría inalterado.

Vemos aquí un gráfico que muestra el concepto de widget y su interactuación con el servicio de interfaz de usuario.

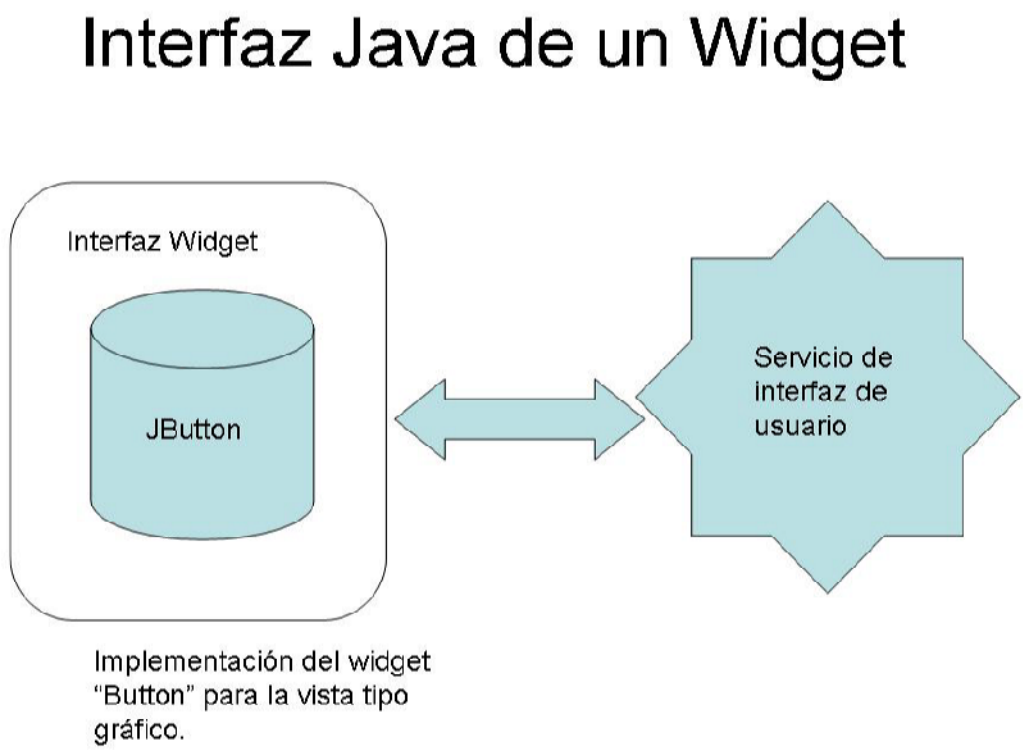


Figura 7. Representación del diálogo entre nuestro servicio de interfaz de usuario y un "widget" de la interfaz.

El diálogo entre el servicio de interfaz de usuario y los widgets que componen la interfaz gráfica, se realiza gracias a que todos los widgets implementan la interfaz widget, por lo que existen una serie de métodos básicos que conoce el servicio para poder dialogar con cada uno de los elementos que forman parte de la interfaz gráfica en cualquiera de sus "vistas" posibles.

Los métodos que debe implementar un widget para formar parte del servicio de interfaz de usuario son los siguientes:

- ***public Object returnSpecificObject() :*** Este método debe devolver el objeto que implementa internamente al widtget en cuestión. Por ejemplo si estamos usando un JButton del paquete swing para definir el widget Button de la "vista gráfica", el método devolvería un swing.JButton. El implementador del widget es el que debe proporcionar el tipo devuelto por parte de este método para poder realizar un "cast" a ese tipo en concreto.

Este método da una gran potencia al servicio de interfaz de usuario, ya que permite acudir directamente al núcleo del widget y poder usar todos sus métodos directamente. Si el tipo devuelto no fuese conocido, podemos interactuar igualmente con el widtet concreto, pero simplemente a través de los métodos públicos que de que nos provea la implementación del widget en cuestión.

- ***public void addComponent(widget element, Object position):*** Este método se utiliza para añadir al objeto sobre el que se ejecuta un widget en una posición determinada. El parámetro "widget element" es el elemento que queremos añadir, y el parámetro "Object position" es la posición determinada donde queremos que dicho elemento se añada.

  Éste método debe ser implementado por todos los elementos que puedan contener en su interior algún otro elemento, como por ejemplo el widget "window".

  Existen sin embargo otros elementos que no pueden contener a otros elementos como los widgets "label" y "button". En este caso el método debe dejarse vacío, a pesar de que sea obligatoria su definición.

- ***public void addMyChildren(Element element) throws MalformedURLException:***
  El método addMyChildren, se usa por parte del servicio de interfaz de usuario para añadir todos y cada uno de los elementos hijos que cuelguen del objeto sobre el que se está ejecutando. Este método se encuentra a un nivel superior que el método addComponent, al que llamará para añadir cada uno de los elementos hijos. Este método recibe como parámetro un objeto de tipo Element, definido en el paquete JDom para el tratamiento de ficheros XML en java. Este elemento contiene el elemento actual y todos lo elementos "hijos" que cuelgan de él. El método addMyChildren deber recorrer ese "árbol" que constituye el objeto Element, y encontrar cada uno de los widgets hijos del widget actual, generarlos y añadirlos dentro del widget actual a la interfaz de usuario.

  Este método puede lanzar la excepción MalformedURLException debido a que hay objetos que usan direcciones URL para identificar ciertos atributos que deben colocarse en su interior.

- ***public void addToMyParent(Element element):*** Este método se usa para añadir el elemento actual al elemento padre del que cuelga. Este método deber ser implementado por todos los elementos que forman parte de la interfaz, ya que todos excepto el elemento raíz de la interfaz tienen un elemento padre. Recibe como parámetros un objeto del tipo element, ya que necesita conocer quién es el elemento padre del elemento actual para poder añadirlo.

- ***public void repainMyParent(Element element):*** Este método es usado por parte del servicio de interfaz para refrescar el elemento padre el elemento sobre el que se ejecuta para refrescar la vista de la interfaz. Cuando añadimos un nuevo "widget" a la interfaz, es necesario refrescar la "vista" actual para que el nuevo elemento sea visible por parte del usuario. Recibe como atributo un objeto de tipo Element. Este elemento se usa para encontrar el elemento padre que debemos refrescar.

- ***public void repain():*** Este método se usa para realizar el refresco del elemento actual. Cuando cambiamos la propiedad de algunos de los parámetros de un widget es necesario llamar al método repain para que surta efecto en la vista actual que se está mostrando al usuario.

Mirando de forma genérica cualquier tipo de interfaz podemos hacer una clasificación muy genérica de los "widgets" que la componen.

En nuestro proyecto se han creado dos grandes grupos de elementos:

- **WidgetChild:** Son todos aquellos widgets que no pueden contener ningún otro widget en su interior, como label, button …

- **WidgetParent:** Son aquellos widgets que contienen otros widgets en su interior, como window, hbox, …

Para cada uno de estos grupos se ha creado una clase que implementa la interfaz widget y que agrupa todos los elementos genéricos de cada uno de los widgets que pertenecen a cada una de las clases. Así todos los widgets que generemos heredan de estas dos superclases, y heredan el interfaz widget a través de ellas.

Con esto conseguimos un código más eficiente y limpio.

4.3. El Parseador de XUL.

Otro de los grandes bloques que componen el servicio de interfaz de usuario es el parseador XUL.

El parseador XUL es la herramienta que nos permite interpretar el contenido de los ficheros XUL que almacenan la descripción de la interfaz de usuario, para generar una lista de los elementos que debemos presentarle al usuario.

Para cumplir el primer objetivo de este proyecto, (creación de una interfaz java del servicio de interfaz de usuario), se ha definido una interfaz java llamada GenericParser, que deben implementar todos los parseadores XUL que pretendan formar parte del servicio de interfaz de usuario.

En el siguiente esquema vemos el funcionamiento básico del parseador XUL.

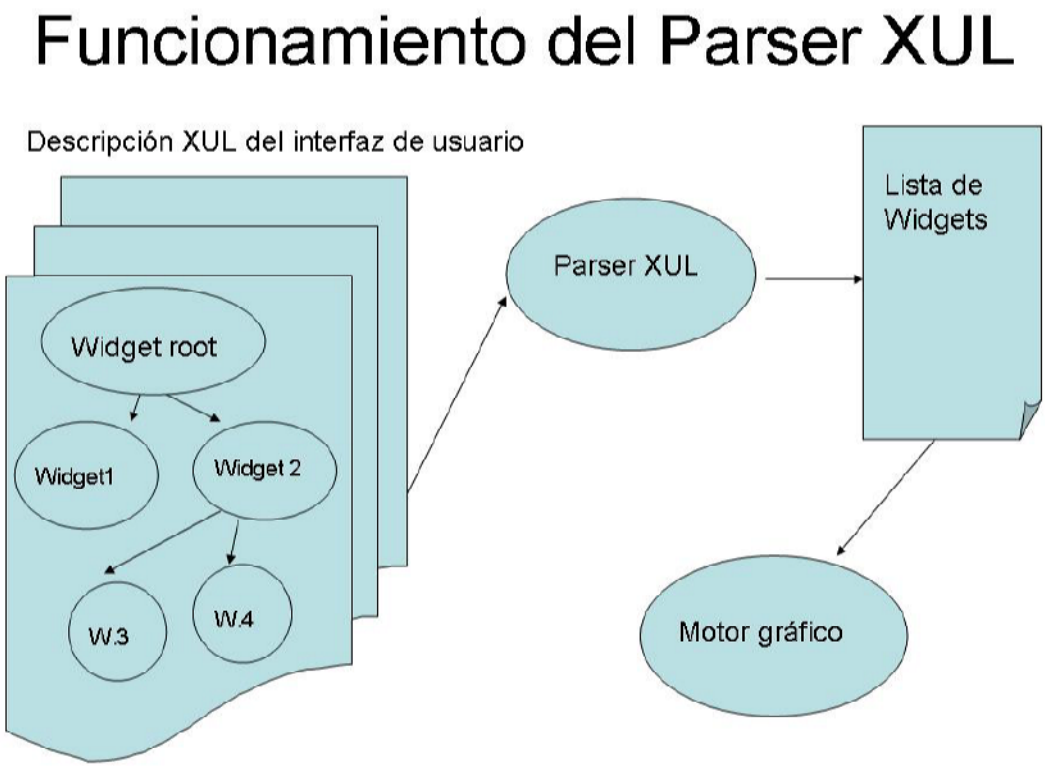


Figura 8. Representa el funcionamiento básico del parseador XUL.

La interfaz java del parseador XUL presenta los siguientes métodos :

- **public void createTree(String url):** Este método se usa para generar la interfaz gráfica contenida en el fichero XUL cuya dirección se pasa como parámetros. Para que este método funcione correctamente es necesario que el archivo contenga un elemento root de tipo <window> . Este método está orientado a generar el primer contenedor que va a contener a todos los widgets de la interfaz.

- **public void addTo(String url):** Éste método se usa para añadir el contenido del fichero XUL que se pasa como parámetros a una interfaz cuyo contendor principal ya ha sido mostrado. Este método necesita que el elemento root del fichero XUL que se le pase como parámetro se encuentre ya desplegado en la interfaz, ya que el elemento root no se añade a la interfaz, si no que se busca el objeto cuyo identificador coincide con el del elemento root del

fichero XUL y se le añaden los demás elementos que contiene el fichero XUL.

Este método da una gran potencialidad al servicio de interfaz de usuario. Nos permite describir la interfaz de usuario en tantos ficheros XUL como creamos oportunos.

Podríamos describir cada uno de los widgets de la interfaz en un fichero XUL aparte, con el consiguiente beneficio a la hora de describir y mantener la interfaz. De esta forma cada widget estaría localizado rápidamente, y la modificación de la interfaz de usuario sería tan sencillo como modificar el fichero que contiene el elemento que queremos modificar.

El comando addTo(), localiza el elemento root del fichero que se le pasa como parámetro. Posteriormente solicita al servicio interfaz de usuario que le pase la referencia del objeto cuyo identificador coincide con el del elemento root del fichero xul que queremos añadir.

Una vez localizado este elemento se añaden todos los widgets que se encuentran descritos en el fichero xul.

4.4. El Motor Gráfico.

Una vez que el parseador XUL genera la lista de elementos que hay que mostrar en la interfaz, necesitamos un elemento que identifique estos elementos, los instancia, los enlace adecuadamente y por fin que los muestre al usuario.

Para llevar a cabo esta tarea se ha creado otro elemento importante del servicio de interfaz de usuario llamado “motor gráfico”.

Otra tarea fundamental que realiza el motor gráfico es realizar el registro de los elementos a través de su identificador.

Cuando realizamos la instancia de un elemento, debemos tenerlo identificado para poder acceder posteriormente a él. Esta tarea es fundamental para el correcto funcionamiento de la interfaz, ya que se requerirá, a petición del usuario la modificación de ciertos elementos, y es necesario que estén unívocamente identificados cada uno de los widgets para poder acceder a ellos de forma correcta.

En esta figura podemos ver cómo es el funcionamiento del motor gráfico del servicio de interfaz de usuario:

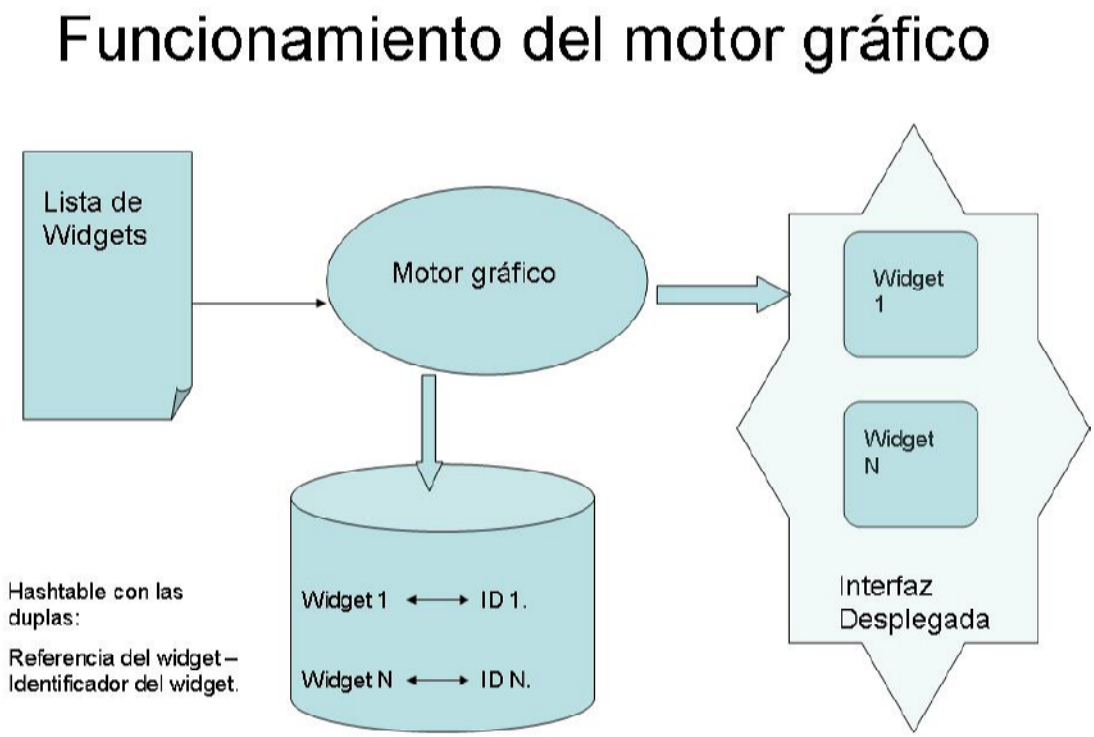


FIGURA 9. Funcionamiento genérico del motor gráfico.

Como para el caso del parseador xul y los elementos widgets, se ha generado una interfaz java que cualquier implementación del motor gráfico debe implementar.

La interfaz java del motor gráfico, GenericEngine, debe implementar los siguientes métodos:

- ***public void engine(Element element) throws MalformedURLException:*** El método engine es llamado por cada uno de los elementos que devuelve el parseador xul, y se encarga de generar los objetos widgets.

  Durante la creación de los objetos widgets, se encarga de guardar la referencia de los objetos junto a su identificador. Como el id de cada elemento es único, podemos acceder a cada uno de ellos sin problema de ambigüedad.

  También es el encargado de agregar los elementos a sus "padres" y a realizar el correcto refresco de los elementos existentes en la interfaz.

4.5 El Sistema de Registro de Widgets.

Como hemos comentando anteriormente, a la hora de generar los elementos, se introducen en una estructura junto con el identificador de cada objeto.
La estructura elegida para almacenar la referencia de los objetos es un objeto de tipo Hashtable, a la que se ha provisto de una clave para recuperar a los elementos. En esta tabla se almacenan todos los widgets como objetos widget, junto a su clave, que no es mas que el id de cada uno de los elementos.

De aquí la importancia de que el identificador que aparece como atributo de cada elemento aparezca en su definición, y que sea único.

Además del Hashtable de ha generado creado un método que sirve para dialogar con la tabla.

- ***public Object GiveMeObject(String object_name):*** Con este método podemos recuperar el elemento cuyo identificador se pasa como parámetro.

4.6 Los Manejadores de Eventos.

El servicio de interfaz de usuario, es capaz de detectar cualquier evento que se produzca a la interfaz y localizar el manejador de eventos asociado para notificarle el evento y que se trate de forma adecuada.

A diferencia que en XUL, en el que los manejadores de evento son scripts de Java, los manejadores de evento del lenguaje LXUL se escriben en Java, por lo que tenemos mayor capacidad de ejecutar acciones.

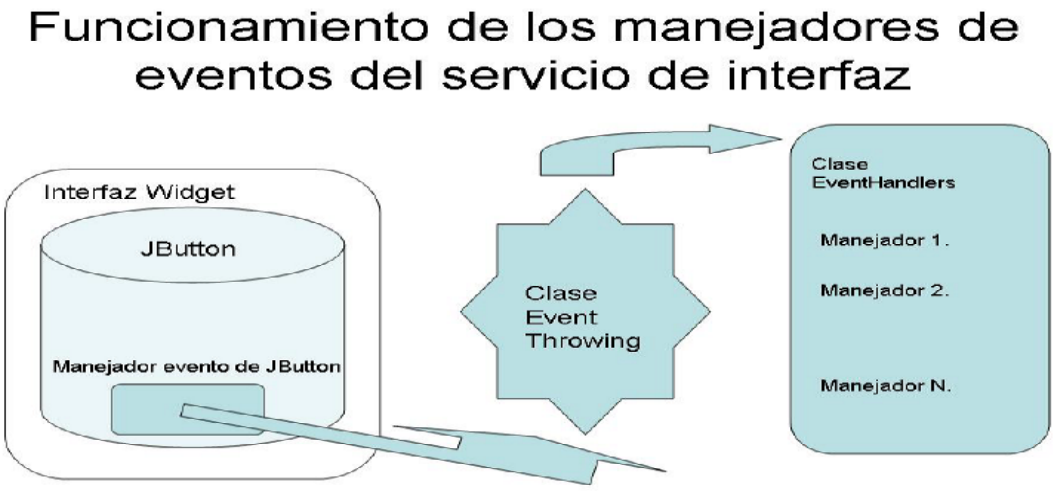


Figura 10: Funcionamiento de los manejadores de eventos.

Para llevar a cabo la gestión de los eventos se ha acudido a usar el API reflect de Java. Se han generado dos clases fundamentales para llevar a cabo tal fin que pasamos a comentar seguidamente.

- ***Clase EventThrowing:*** Esta clase resuelve el lanzamiento externo de los manejadores en tiempo de ejecución. El manejador de eventos es resuelto en tiempo de ejecución y es lanzado para manejar el evento en cuestión.

  Cuando un el "núcleo" de uno de los widgets detecta un evento acude a su manejador de eventos. Es aquí donde debemos realizar la lectura del atributo xul que contiene el nombre del método que queremos lanzar. Una vez conocido, usando el método throwing() de esta clase lanzamos el método que manejará el evento.

- ***public void setEvent(String event_name):*** Este método establece el nombre del método que debe ser lanzado por el método throwing en su lanzamiento.

- ***public void setEventContainer(String event_class):*** Este método establece el nombre de la clase que contiene a los manejadores de eventos.

- ***public void throwing():*** Este método lanza el método contenido en la clase establecida por setEventContainer(), y cuyo nombre fue establecido por setEvent.

5. El concepto de vista

El servicio de interfaz de usuario tiene como objetivo dar al desarrollador de aplicaciones de la pasarela residencial, la posibilidad de describir la interfaz de usuario una sola vez y que ésta descripción sea válida para cualquier tipo de dispositivo donde pueda mostrarse.

Definimos como "vista" de la interfaz de usuario, a cada una de las representaciones posibles que el servicio interfaz de usuario puede mostrar una misma descripción de interfaz de usuario.

Por ejemplo, podemos tener una vista tipo "gráfico", o tener una interfaz tipo "texto", o una vista interfaz tipo "html".

6. Implementación de la interfaz Java del servicio

Llegados a este punto en el que tenemos completamente definida la interfaz java del servicio de interfaz de usuario, vamos a describir el segundo objetivo marcado en nuestro proyecto: la implementación de la interfaz java del servicio de interfaz de usuario.

Para comprobar la viabilidad de nuestro servicio de interfaz de usuario hemos realizado la implementación de dos vistas distintas de nuestra interfaz java del servicio de interfaz de usuario.

6.1. Vista gráfica tipo Swing

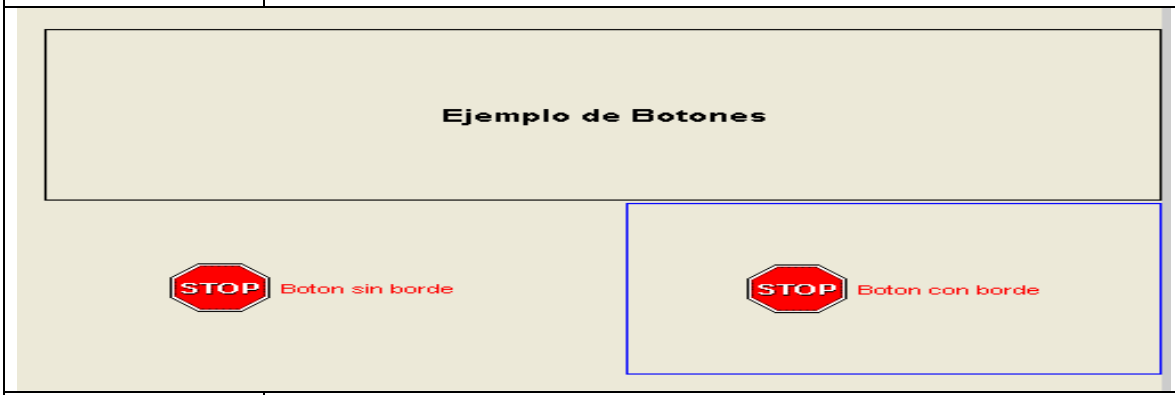
La vista tipo swing ha sido desarrollada para terminales capaces de mostrar widgets del tipo swing de java. Se han implementado todos los widgets descritos en la descripción del lenguaje LXUL.

Se implementa la interfaz java GenerirParser y se crea el ParserSwing. Este parseador tiene en cuenta los tipos de widgets implementados para la interfaz swing.

Se implementa también la interfaz java GenericEngine y se crea el SwingEngine, que usa los widgets swing implementados para componer la interfaz de usuario.

Presentaremos aquí una breve descripción de los manejadores que se han implementado para cada uno de los widgets swing y una imagen de cada uno de ellos.

Estos manejadores pueden usarse directamente por parte de los desarrolladores de la interfaz para modificar la apariencia de la misma.

NOMBRE WIDGET	MÉTODOS MODIFICADORES
Window	• SetWindowLookAndFeel() : Establece la apariencia de la aplicación. Se elige la apariencia por defecto del entorno en el que se esté ejecutando la aplicación. • SetWindowSize(int sizex, int sizey) : Establecemos el tamaño de la ventana al tamaño representado por los parámetro del método. • SetWindowVisible(boolean state): Visibilidad de la ventana.
	
Button	• SetBorder(boolean state) : Permite establecer si el botón debe representarse con borde o no. • SetColor(Color color): Establece el color de la fuente de la etiqueta que aparece en el botón. • SetFont(Font font) : Establece la fuente de la etiqueta del botón. • SetIcon(String iconpath): Establece la dirección del icono que debe representarse junto a la etiqueta del botón. • SetLabel(String Label): Establece el valor de la etiqueta que se representa en el botón.
	
Label	• SetText(String Text) : Establece el texto de la etiqueta. • SetColor(Color color): Establece el color de la fuente de la etiqueta que aparece en el label. • SetFont(Font font) : Establece la fuente de la etiqueta del label.

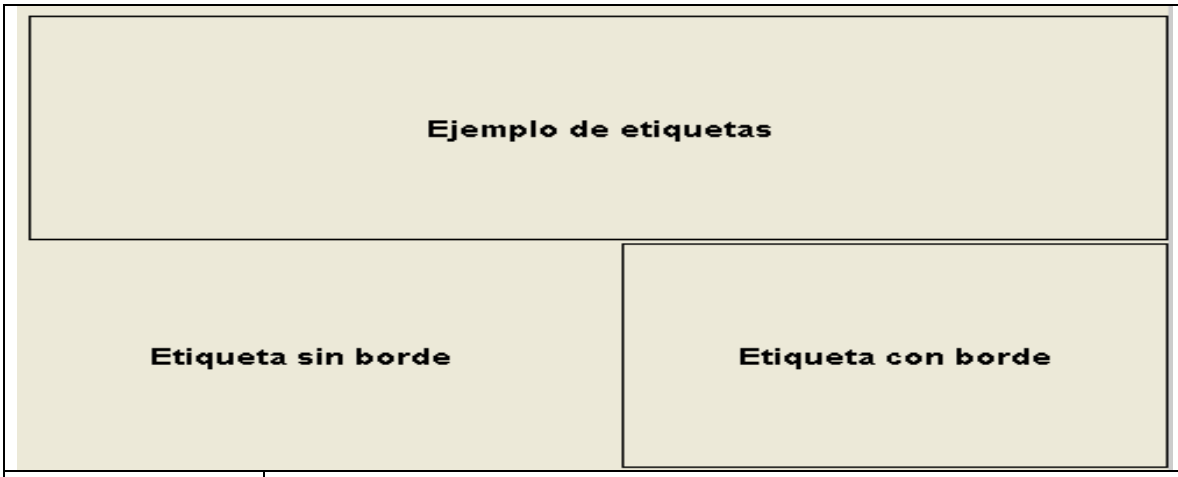
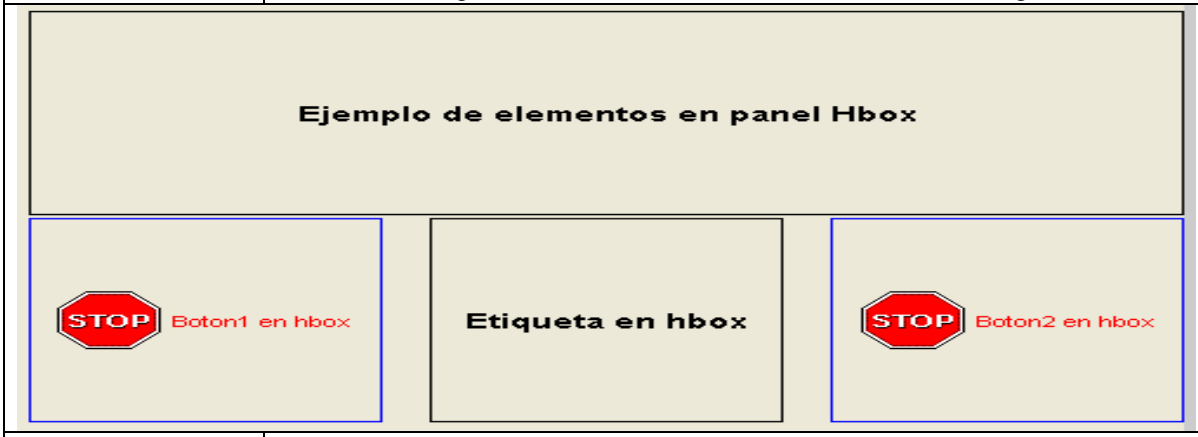


Image	• Sólo se permiten modificaciones a través del núcleo del widget

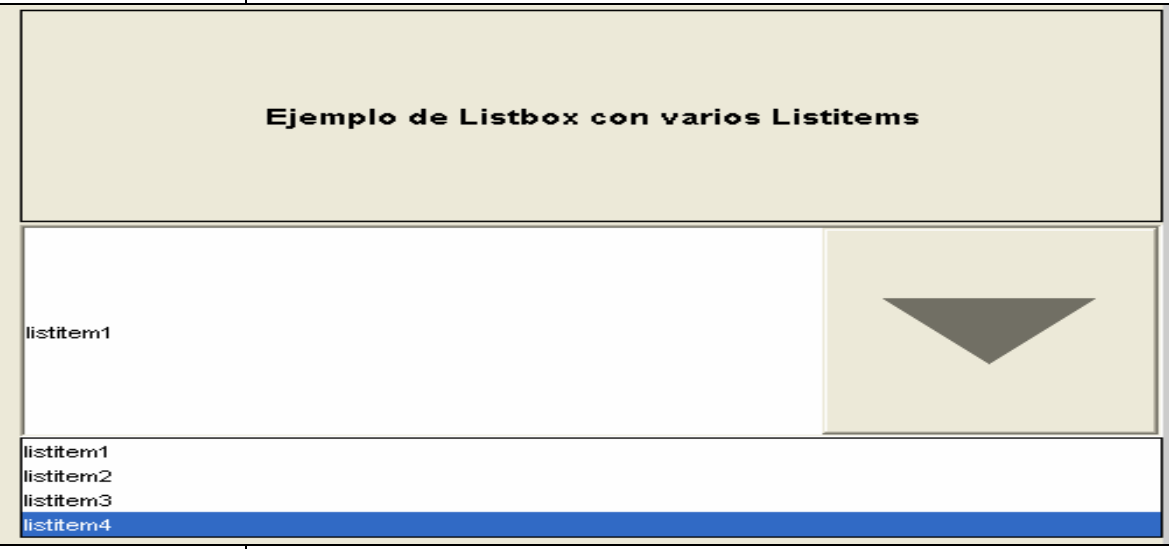
Hbox	• Sólo se permiten modificaciones a través del núcleo del widget



Vbox	• Sólo se permiten modificaciones a través del núcleo del widget

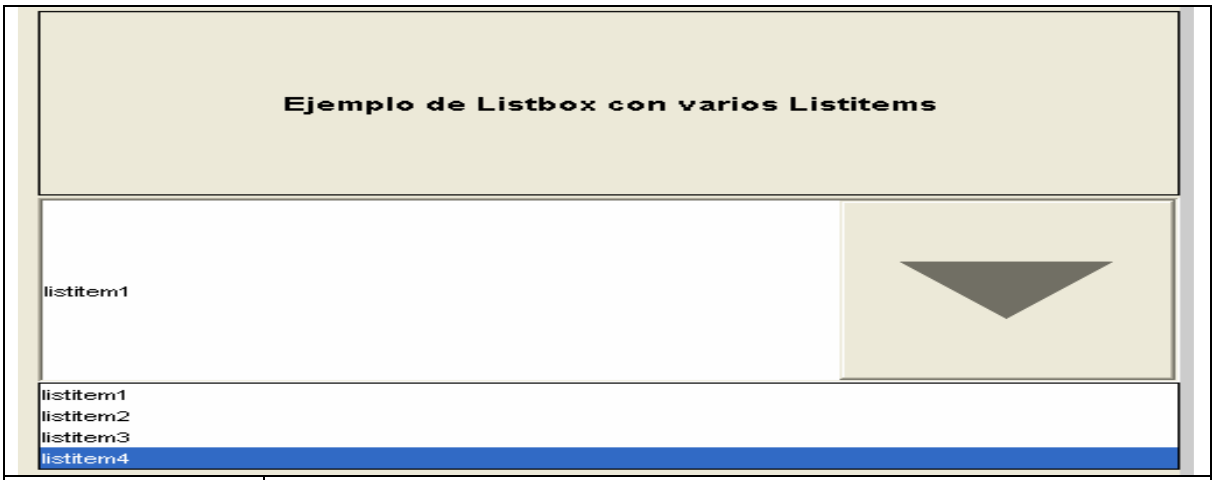
BoxFiller	• SetDimension(Sizex, Sizey) : Establece el tamaño del relleno.

ListBox	• Sólo se permiten modificaciones a través del núcleo del widget



ListItem	• Sólo se permiten modificaciones a través del núcleo del widget
• Puede verse un ejemplo de listitem en la imagen anterior.	
MenuItem	• Sólo se permiten modificaciones a través del núcleo del widget

• Puede verse un ejemplo de listitem en la imagen siguiente.	
MenuPopup	• Sólo se permiten modificaciones a través del núcleo del widget
	
RadioButton	• Sólo se permiten modificaciones a través del núcleo del widget
	
RadioGroup	• Sólo se permiten modificaciones a través del núcleo del widget
	
TextBox	• GetText() : Obtiene el texto introducido en el campo de texto.



MenuList	• Sólo se permiten modificaciones a través del núcleo del widget

6.2 Vista tipo Texto

La interfaz tipo texto ha sido implementada para representarse en terminales que no permiten interfaces de tipo swing de java. Tiene un estilo similar a las interfaces NCurses de Linux.

Para su implementación se ha usado un paquete java denominado charva. Este paquete permite desarrollar una interfaz de tipo texto usando los mismos elementos de una interfaz swing de java.

Se implementa también la interfaz java GenericEngine y se crea el TextEngine, que usa los widgets texto implementados para componer la interfaz de usuario.

Presentaremos aquí una breve descripción de los manejadores que se han implementado para cada uno de los widgets texto y una imagen de cada uno de ellos.

Estos manejadores pueden usarse directamente por parte de los desarrolladores de la interfaz para modificar la apariencia de la misma.

NOMBRE WIDGET	MÉTODOS MODIFICADORES
Window	• SetWindowSize(int sizex, int sizey) : Establecemos el tamaño de la ventana al tamaño representado por los parámetro del método.
Button	• Sólo se permiten modificaciones a través del núcleo del widget
	
Label	• SetSize(int x,inty): Establece el tamaño de la imagen.

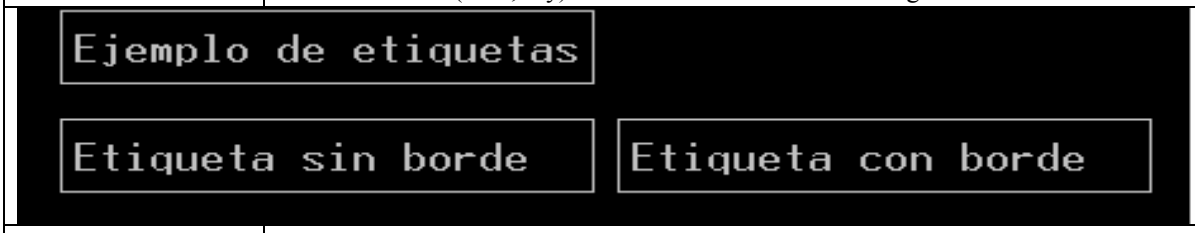
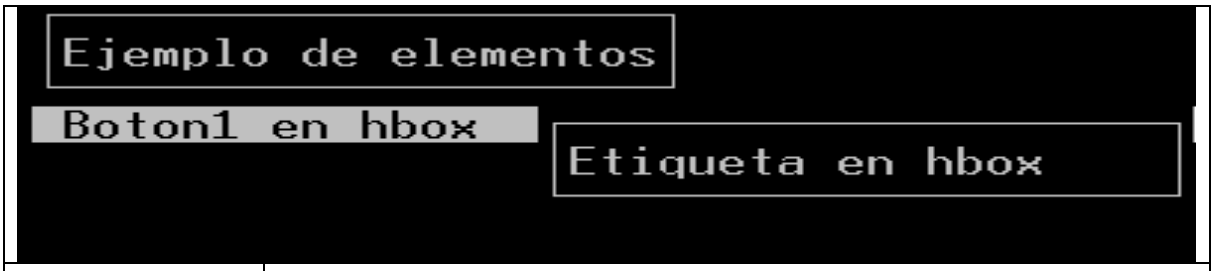
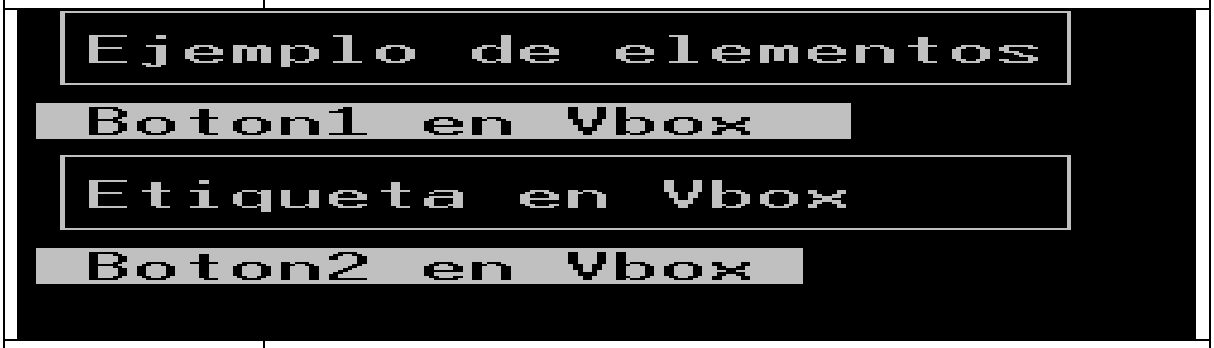



Image	• No implementado en la interfaz texto.
Hbox	• Sólo se permiten modificaciones a través del núcleo del widget



Vbox	• Sólo se permiten modificaciones a través del núcleo del widget

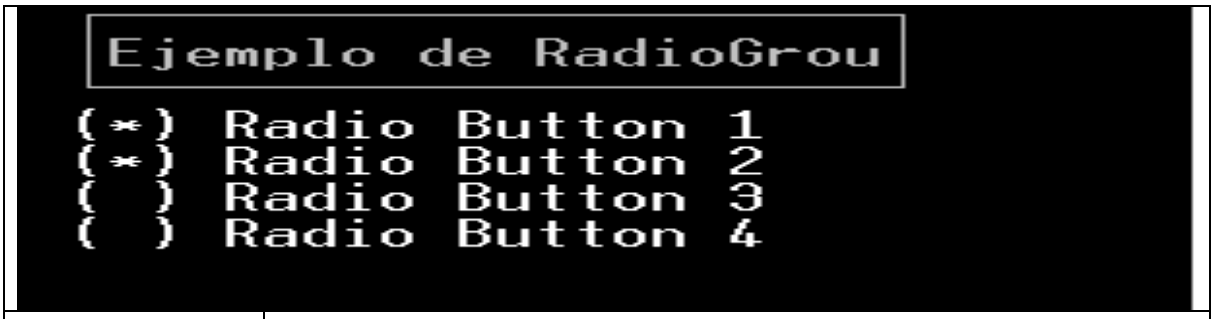


BoxFiller	• No implementado en la vista texto.
ListBox	• Sólo se permiten modificaciones a través del núcleo del widget

ListItem	• Sólo se permiten modificaciones a través del núcleo del widget
• Puede verse un ejemplo de listitem en la imagen anterior.	
MenuItem	• Sólo se permiten modificaciones a través del núcleo del widget
• Puede verse un ejemplo de listitem en la imagen siguiente.	
MenuPopup	• Sólo se permiten modificaciones a través del núcleo del widget

RadioButton	• Sólo se permiten modificaciones a través del núcleo del widget

RadioGroup	• Sólo se permiten modificaciones a través del núcleo del widget



TextBox	• GetText() : Obtiene el texto introducido en el campo de texto.

MenuList	• Sólo se permiten modificaciones a través del núcleo del widget

7. Generación del bundle "Servicio de Interfaz de Usuario"

El objetivo final que nos marcamos al inicio del proyecto, fue la creación del bundle "servicio de interfaz de usuario".

La creación del bundle "servicio interfaz de usuario" permite ofrecer el servicio para todas las aplicaciones que se ejecutan en la pasarela residencial.

Crearemos un servicio al que llamaremos OSGi_UIService. Para llevar a cabo la implementación del bundle, necesitamos crear un archivo llamado "manifiesto", en el que se guarda cierta información referente al bundle.

Podemos ver aquí el contenido del archivo manifiest.mf :

Bundle-Name: SwingTextTutorial
Bundle-Description: Bundle que contiene una aplicación de ejemplo del framework de desarrollo de interfaces de usuario con XUL.
Bundle-ClassPath: .
Bundle-Activator: org.ait.osgi.SwingTextTutorial.Activator
Bundle-Vendor: Emilio Jose Maldonado García
Bundle-Version: 1.0

La etiqueta Bundle-Activator indica la clase equivalente al "main" de un proyecto java convencional.
Lo último que necesitamos es definir el interfaz del servicio. Llamaremos a esta clase OSGi_UIInterface.

Ya que nuestra aplicación hace uso de paquetes como jdom (tratamiento xml en java) o charva (librería gráfica para generar elementos texto), hemos optado por incluir estos paquetes dentro de nuestro bundle.

No es la solución más elegante, ya que lo ideal sería generar un nuevo bundle cuyo objetivo sea exportar estos paquetes, pero simplifica nuestro proyecto y nos permite probar nuestro bundle de una forma simple.

Una vez obtenido el empaquetado final del bundle pasaremos a ver la manera de instalar dicho bundle en el framework de oscar para su correcto funcionamiento.

Si tenemos instalado el framework de oscar en nuestro equipo, debemos buscar la carpeta "oscar" donde se encuentre el ejecutable del servicio de registro de bundles.

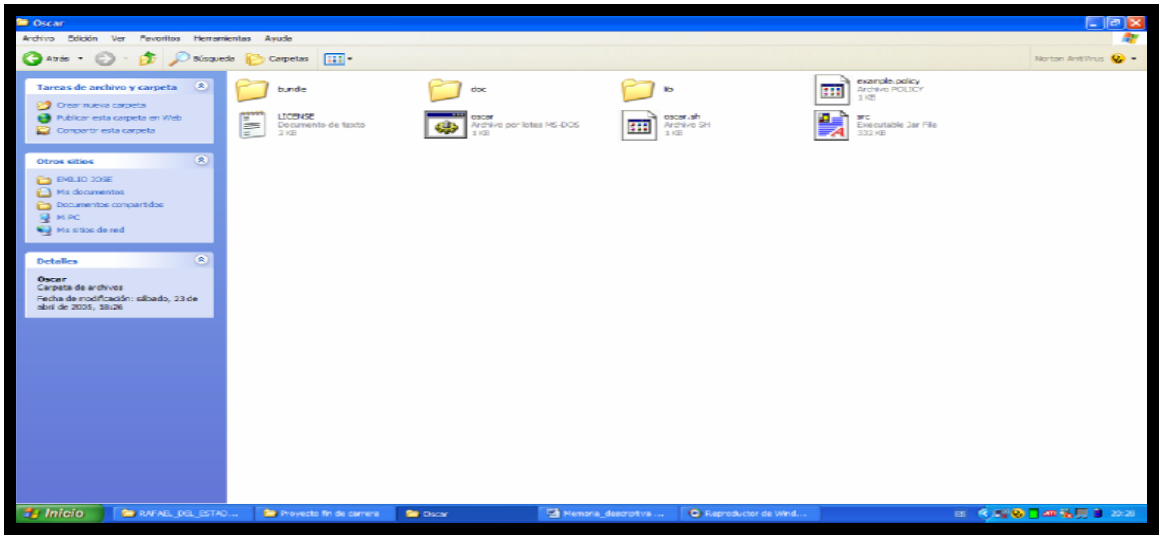


Figura 11: Carpeta donde podemos encontrar el script de Linux para lanzar el servicio de registro de bundle, o el archivo de ejecución por lotes de window para el mismo fin.

Una vez localizada la carpeta oscar debemos introducir en ella el bundle de "servicio_interfaz" que hemos creado y que queremos registrar en nuestro framework de OSCAR.

Debemos introducir el bundle en la carpeta "bundle" de la carpeta "oscar".

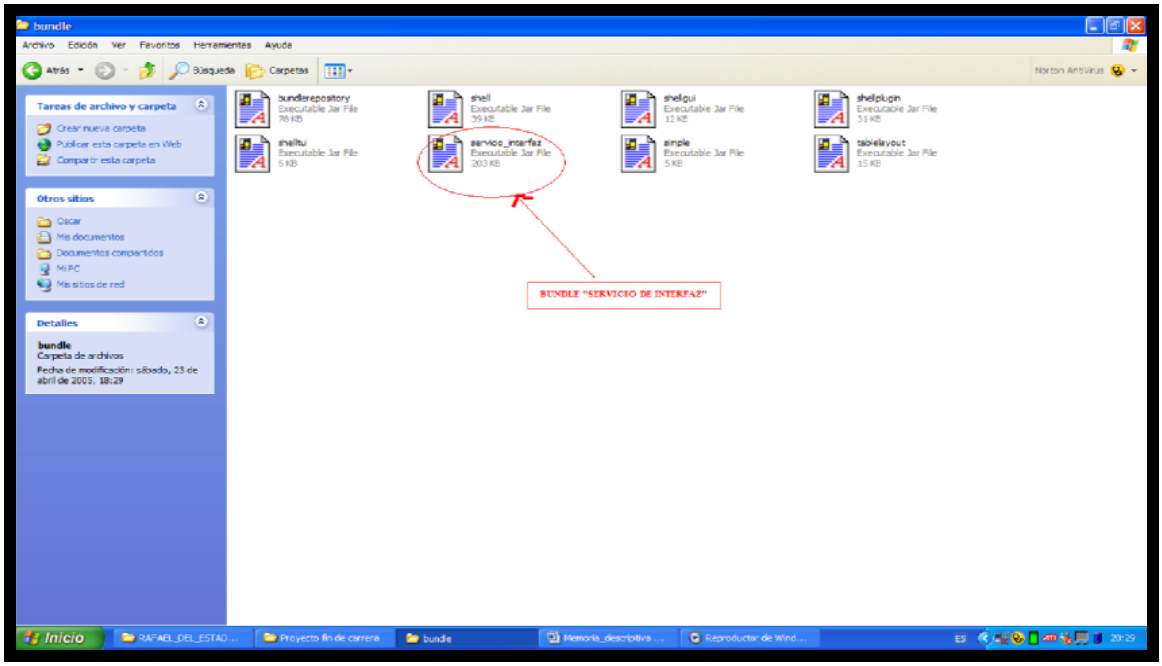


Figura 12: Carpeta Bundle donde están contenidos los bundles registrados en el framework de OSCAR.

Debemos lanzar el servicio de registro de bundles, para indicarle al framework que queremos instalar nuestro servicio.

Para ello hacemos doble clic en el archivo “oscar.bat”.
Nos aparece el interfaz gráfico del servicio de registro de bundles del framework de OSCAR.

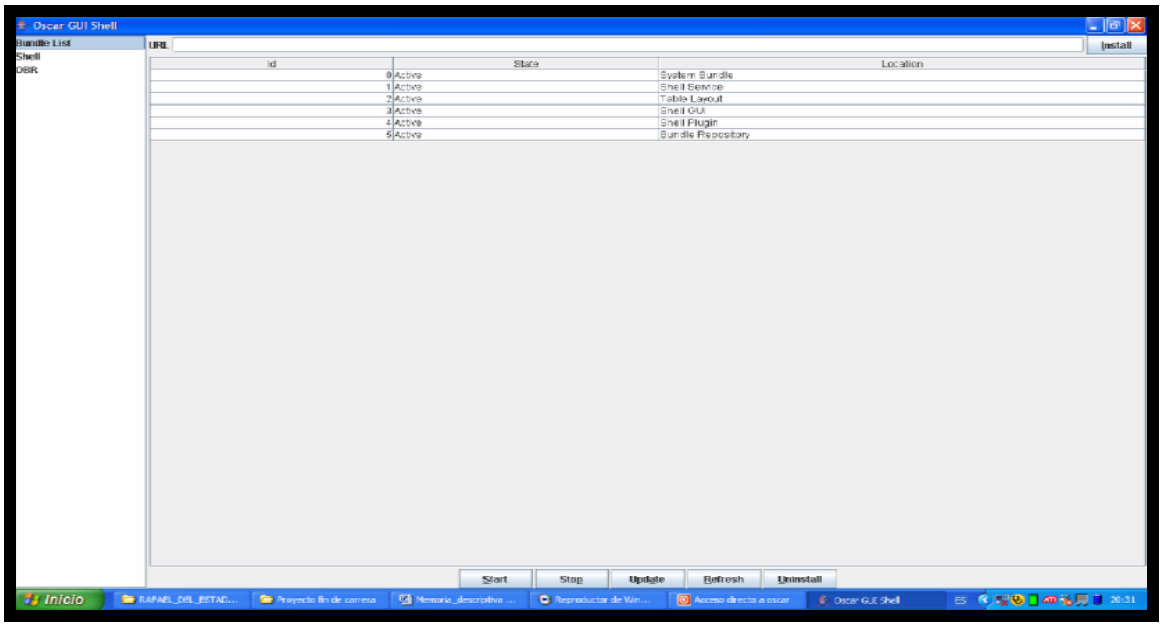


Figura 13: Interfaz grafico el Shell de OSCAR.

Para instalar nuestro bundle debemos especificar la ruta donde se encuentra en el campo de texto “URL” y pinchar en el botón “Install”.

Si la ruta es correcta aparecerá una nueva entrada con el nombre de nuestro burdle.

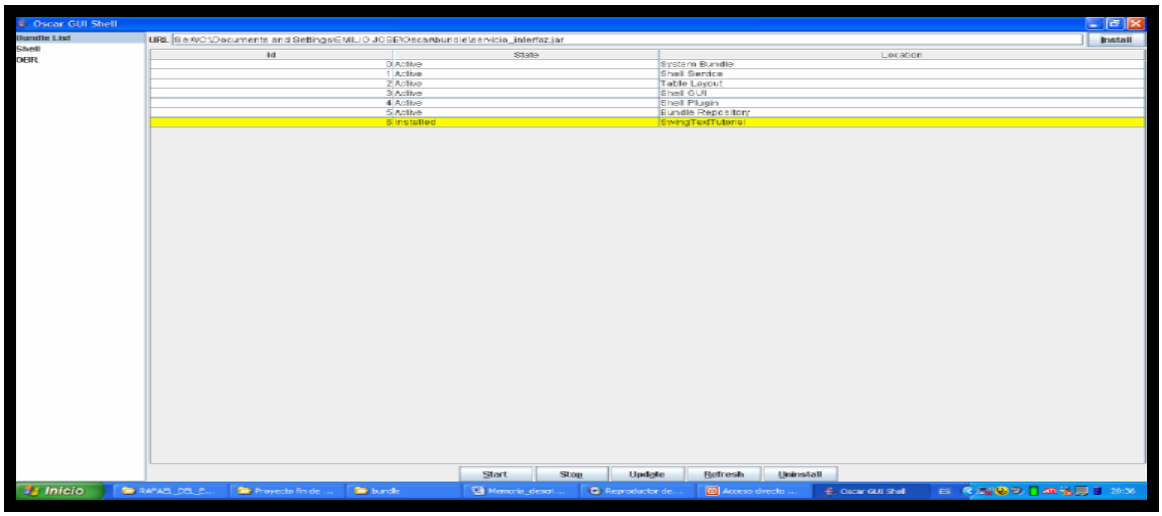


Figura 14: Bundle SwingTextTutorial instalado en el Framework.

Por último debemos activarlo para ver su funcionamiento. En este paso el framework verifica que se satisfacen todas las dependencias necesarias para el correcto funcionamiento del bundle. Si fallase alguna se lanzaría un mensaje de error y sería necesario resolver el problema antes de continuar.

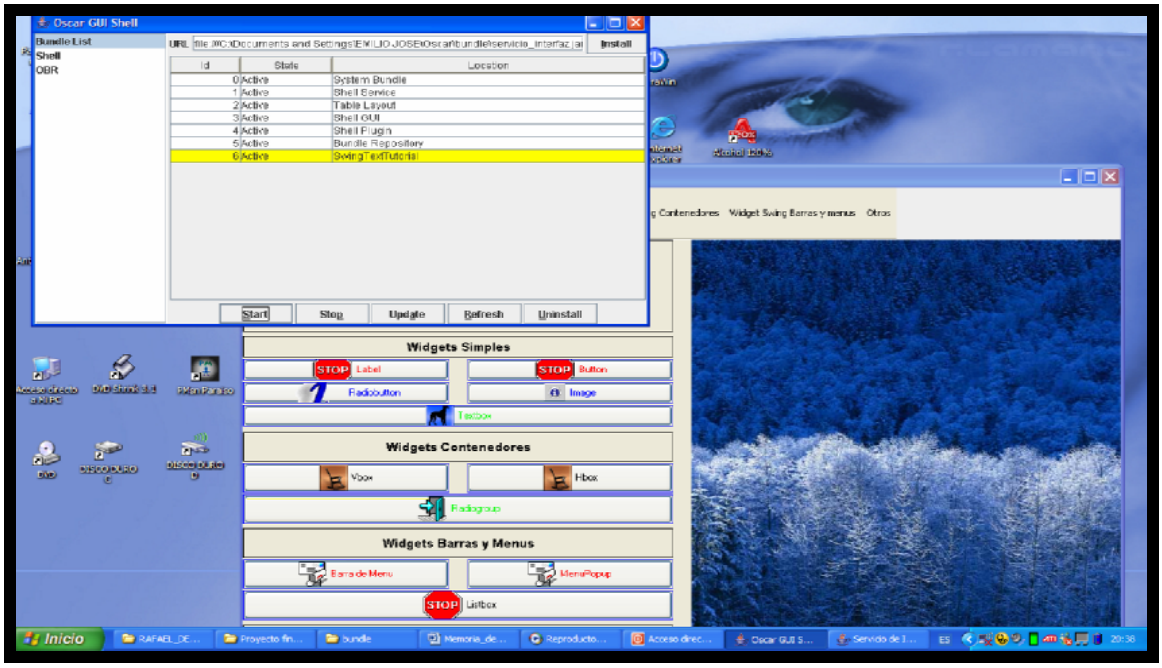


Figura 15: Bundle SwingTextTutorial ejecutándose. Podemos ver la aplicación de ejemplo desplegada en su vista "swing".