Programación de TDT

Manuel J. Prieto[1]

[1] Banco Popular, Spain
mprieto@popular.es

Resumen. La aparición de la televisión digital supone una revolución. En ella se aumenta notablemente la calidad de la imagen y se optimiza el ancho de banda permitiendo un mayor número de canales. Con el objetivo de crear un mercado horizontal para la distribución de la televisión digital, aparece DVB Project, que es el consorcio industrial encargado de la definición de todos los aspectos relacionados con su transmisión. Al orientar DVB Project la distribución en un mercado horizontal, basado en estándares abiertos los dispositivos son interoperables. Se consigue que el receptor funcione con cualquier difusor (broadcaster) y, por tanto, el proveedor de contenidos tan solo tiene que desarrollar su producto para una plataforma, ya que funcionará en cualquier equipo. En este capitulo se hace una profunda revisión de la programación TDT y de sus estandares. Ademas, se analizarán los sistemas de difusión que convierte la imagen analogía y digital con el fin de optimizar el ancho de banda.

Palabras clave: Optimización del ancho de banda; DVB Project; TDT

Abstract. The emergence of digital television is a revolution. It significantly increases image quality and optimizes bandwidth allowing a greater number of channels. With the aim of creating a horizontal market for the distribution of digital television, DVB Project appears, which is the industrial consortium in charge of defining all aspects related to its transmission. By orienting DVB Project distribution in a horizontal market, based on open standards the devices are interoperable. The receiver is made to work with any broadcas-ter and, therefore, the content provider only has to develop its product for a silver-form, as it will work on any equipment. In this chapter there is a deep revision of the TDT programming and its standards. In addition, the broadcasting systems that convert analog and digital images will be analyzed in order to optimize the bandwidth.

Keywords: Bandwidth optimization; DVB Project; TDT

ISBN: 978-84-9012-858-9
Colección Aquilafuente: AQ-238
Ediciones Universidad de Salamanca - cc by nc dc

1 Evolución de la Televisión

La palabra Televisión viene de la composición de la palabra griega “Tele” (distancia) y la latina “Visio” (Visión), el cual hace referencia a todos los aspectos sobre su transmisión y recepción.
La televisión analógica ha sobrevivido durante más de 50 años, evolucionando.
En 1940, con objetivo de unificar los sistemas de EEUU, nace el National Television System Committe (NTSC), el cual es un sistema de codificación y transmisión analógica a color. Se emplea en la mayor parte de Estados Unidos y Japón entre otros países y se caracteriza por la transmisión de 30 imágenes por segundo (60 campos entrelazados) formadas por 525 líneas horizontales (visibles tan solo 486) y 640 verticales con una relación de aspecto-imagen de 4/3. Como hemos dicho, y con el fin de obtener mayor partido del ancho de banda, se utiliza un modo entrelazado por el cual se presentan primero las líneas pares y luego las impares, de modo que nunca coinciden en el mismo tiempo [1-5].
Una década más tarde, aparece PAL (Phase Alternating Line) para adaptar NTSC a Europa, mejorando algunos problemas de NTSC con la fase del color. Transmite a 25 imágenes por segundo (50 campos entrelazados) formadas por 625 líneas horizontales (576 visibles) y 720 verticales con una relación de imagen de 4/3, al igual que NTSC.
La diferencia de frames por segundo (fps) entre ambos sistemas viene determinado por la frecuencia de la corriente alterna ya que en Estados Unidos es de 60Hz y en Europa de 50Hz.
Por último, en 1967 aparece SECAM para adaptar NTSC a Francia más por orgullo que por razones tecnológicas, ya que PAL era alemán. También fue implantado en la URSS con el fin de evitar la propaganda capitalista.
En la imagen siguiente podemos ver la distribución de los distintos formatos por el mundo.

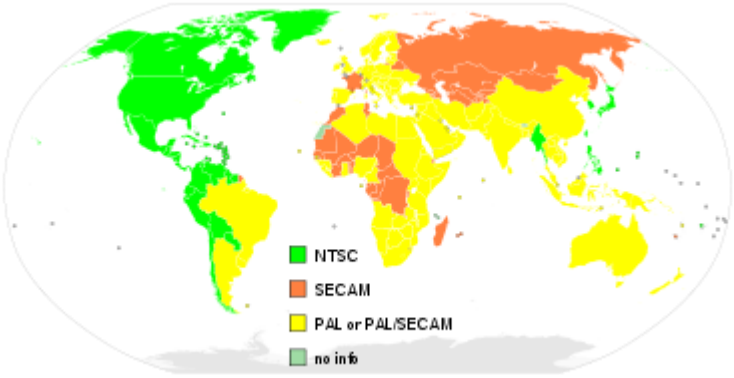


Figura 80 Imagen obtenida de Wikipedia

La aparición de la televisión digital supone una revolución. En ella se aumenta notablemente la calidad de la imagen y se optimiza el ancho de banda permitiendo un mayor número de canales. Permite el HTDV (televisión de alta definición) y servicios adicionales como la interactividad y EPG (guía electrónica de programas).
Inicialmente, las primeras emisiones digitales no estaban sujetas a ningún estándar, siendo el operador el encargado de proporcionar el decodificar específico al usuario (Canal Satélite Digital, Vía Digital o las operadoras de cable). En estos mercados verticales, los productos de distintas marcas no son compatibles, de modo que el proveedor de contenidos tiene que adaptar estos a los diferentes sistemas.
Con el objetivo de crear un mercado horizontal para la distribución de la televisión digital, aparece DVB Project, que es el consorcio industrial encargado de la definición de todos los aspectos relacionados con su transmisión.

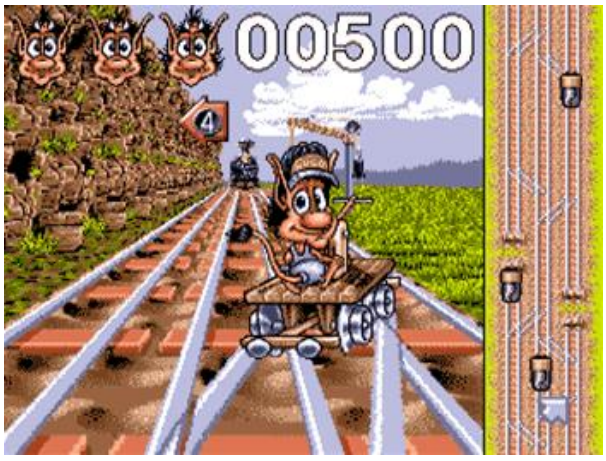
ISBN: 978-84-9012-858-9

Colección Aquilafuente: AQ-238
Ediciones Universidad de Salamanca - cc by nc dc

Al orientar DVB Project la distribución en un mercado horizontal, basado en estándares abiertos los dispositivos son interoperables. Se consigue que el receptor funcione con cualquier difusor (broadcaster), y por tanto, el proveedor de contenidos tan solo tiene que desarrollar su producto para una plataforma, ya que funcionará en cualquier equipo. Esto también posibilita que los fabricantes desarrollen dispositivos con valor añadido con el fin de fomentar la competitividad.
El estándar DVB para televisión reutiliza estándares ya existentes (MPEG-2), definiendo distintos medios de transmisión (DVB-T, DVB-S y DVB-C) y una plataforma de interactividad (DVB-MHP).
Analizando brevemente los fundamentos del sistema de difusión, nos encontramos que inicialmente un Compresor se encarga de convertir la señal analógica en digital, aplicando un formato de compresión con el fin de optimizar el ancho de banda. DVB utiliza como ya hemos citado anteriormente MPEG-2, que diferencia entre las tramas de video, audio y datos. A partir de estas tramas de entrada se crea un único flujo de salida mediante un multiplexor, ordenando los paquetes de las tramas originales, de forma que sigan un orden lógico.
Equivalente a un canal de TV Analógica, disponiendo del mismo ancho de banda (8MHz, 19.9Mb/s) y sabiendo que un canal requiere aproximadamente 4-5Mb/s, esto supone que caben 4 o 5 programas de TV en un multiplexor. Por lo que si se incluyen sólo 4 tenemos que queda un espacio residual que es el utilizado para las aplicaciones interactivas.
Por tanto, en lo que en la televisión analógica se enviaba un canal, ahora se tiene capacidad para enviar varios y además sobra espacio para enviar aplicaciones interactivas, sacando mas partido así a la frecuencia de transmisión que es un recurso limitado [6-10].

2 ¿Qué es la Interactividad?

Consiste en la participación activa del espectador en la programación de TV. Esta ha existido desde siempre, ya fuese usando el correo, el teléfono o más recientemente internet o SMS.
Como claro ejemplo disponemos del teletexto, como primer servicio interactivo bajo demanda con un canal unidireccional, en el cual se transmite la información en las líneas de sincronización vertical, viajando la información mediante un carrusel que se repite periódicamente.
O como olvidar el famoso juego de Hugo a principios de los 90, un claro ejemplo de interactividad telefónica, el cual utilizaba los números del teléfono para indicar al personaje cual era la vía que tenía que tomar.



ISBN: 978-84-9012-858-9
Colección Aquilafuente: AQ-238
Ediciones Universidad de Salamanca - cc by nc dc

2.1 Interactividad en la Televisión Digital

En la televisión digital podremos distinguir entre dos tipos (requiriendo siempre de un canal bi-direccional):

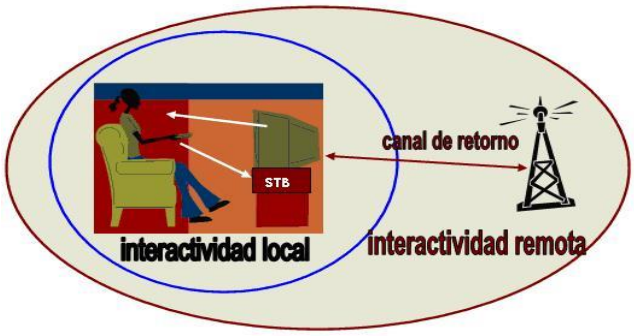


Figura 81 Imagen tomada de televisiondigitalterrestretdt.com

2.1.1 Interactividad Local

Que abarca los servicios básicos de la TV Digital, en el que los datos se encuentran exclusivamente en el canal de difusión. Podemos encontrar aplicaciones simples como teletextos avanzados hasta juegos o complementos a la programación, en las que el usuario puede acceder a la información, pero no puede enviar datos de vuelta.

2.1.2 Interactividad con canal de retorno

Al disponer de canal de retorno (ya sea Modem o DSL), además de poder disponer de los contenidos, el usuario podrá enviar respuestas. Ejemplos claros son chats, encuentras o votaciones.

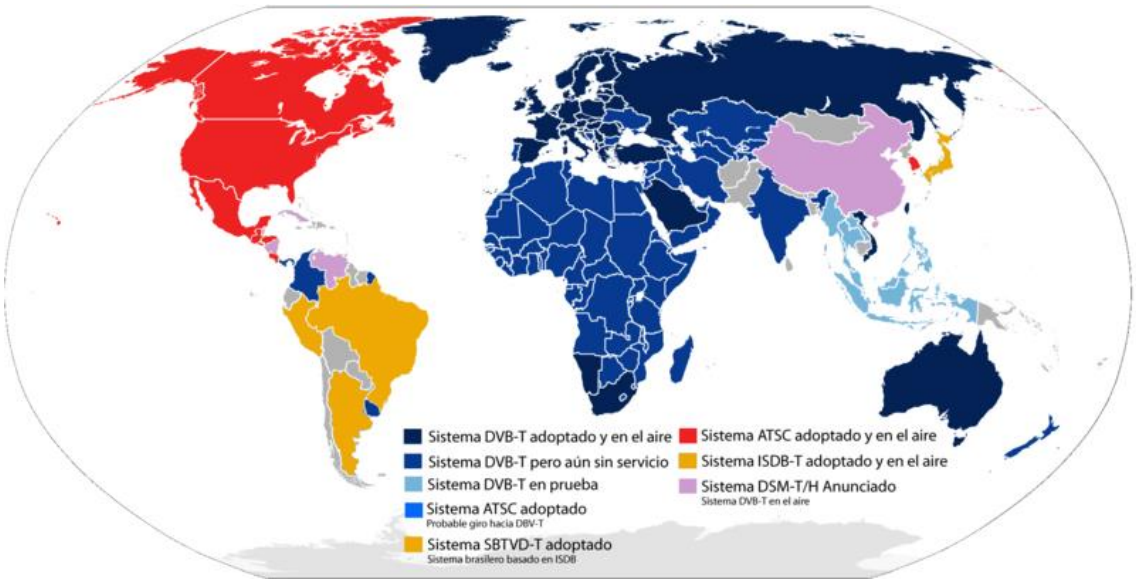
ISBN: 978-84-9012-858-9
Colección Aquilafuente: AQ-238
Ediciones Universidad de Salamanca - cc by nc dc

La TV Interactiva nos aporta grandes ventajas con respecto al PC.
De salida, la actitud del usuario por norma general es más favorable al uso de una televisión que al de un ordenador, ya que lo ve como un elemento de ocio y no de trabajo.
La distancia optima de visión del la TV es de mas 2 metros, mientras que la del PC es de menos de 50 cm, con lo que el usuario no necesita estar justo al lado de la pantalla.
Como es lógico, la TV también se encuentra sometida a ciertas carencias. Por ejemplo, al no disponer de teclado ni ratón, el medio de interacción es el mando a distancia, con lo que aparece una seria dificultad a la hora de escribir textos [11-15].
Podríamos resumir el marco de uso diciendo que la TV es muy útil para consultas esporádicas y rápidas, pero para la realización de trabajos más intensivos se hace necesario el uso de un PC.

3 Multimedia Home Platform – MHP

Plataforma de interactividad definida por el grupo DVB, que aprovecha estándares existentes (MPEG-2, DAVIC…) y que persigue conseguir un mercado horizontal. Es el sistema adoptado como estándar en casi toda Europa, Korea y Australia.

ISBN: 978-84-9012-858-9
Colección Aquilafuente: AQ-238
Ediciones Universidad de Salamanca - cc by nc dc



Según las capacidades del receptor podremos distinguir distintos perfiles:

- **Enhanced Broadcast Profile (MHP 1.0)**

Es el perfil más básico y carece de conectividad IP (no incluye canal de retorno), por lo que está pensado para la descarga a través del canal de difusión (broadcast). Orientado a interactividad local.

- **Interactive Broadcast Profile (MHP 1.0)**

Se incluye el canal de retorno, por lo que ya podemos establecer la interactividad remota.

- **Internet Acces Profile (MHP 1.1)**

Aparte de las capacidades de los otros perfiles, permite compatibilidad con servicios Web, e-mail… ya que permite acceder a internet a través del modem. (En la versión posterior MHP 1.2 ya se da soporte para redes de banda ancha)
La arquitectura de MHP se encuentra definida en tres capas; los recursos (MPEG, CPU,…), los sistemas de software (siendo una capa intermedia utilizada para acceder a los recursos) y las aplicaciones interactivas (Xlets) recibidas a través del canal de difusión junto al audio y video.
Para la construcción de las aplicaciones DVB-MHP utiliza Java como lenguaje de programación, definiendo la plataforma DVB-J basada en la maquina virtual de java (JVM).

ISBN: 978-84-9012-858-9

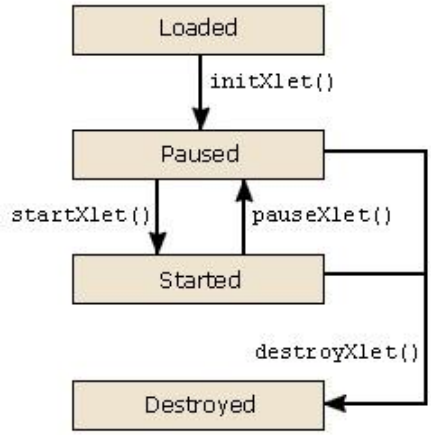
Colección Aquilafuente: AQ-238
Ediciones Universidad de Salamanca - cc by nc dc

3.1 Ciclo de vida de un Xlet

Las aplicaciones interactivas o Xlet's, son una clase java con constructor público que implementa la interfaz javax.tx.xlet.Xlet.

```
public interface javax.tv.xlet.Xlet {
        public void initXlet(XletContext ctx) throws XletStateChangeException;
        public void startXlet() throws XletStateChangeException;
        public void pauseXlet();
        public void destroyXlet(boolean unconditional) throws XletStateChangeException;
}
```

El Xlet puede ser residente o bien ser descargado por un gestor de aplicaciones (Application Manager). Este a su vez será el encargado de gestionar el ciclo de vida del Xlet supervisando y notificando cambios de estado, o destruyéndolo en cualquier momento.
A la derecha podemos ver una imagen que resume los estados del ciclo de vida de un Xlet.



Inicialmente, el gestor de aplicaciones creará una instancia de la clase Xlet pasando al estado Loaded.
Una vez que ya se encuentra cargado, el propio gestor de aplicaciones llamará al método initXlet() haciendo que se quede en estado de pausa. Esta invocación al método de inicio tan solo se realizará una vez en todo el ciclo de vida del Xlet.
En el estado de pausa el Xlet permanece en estado de inactividad a la espera de que se invoque el método startXlet(), momento en el cual comenzará su ejecución.
Y en cualquiera de los estado anteriores, siempre podrá llamarse al método destroyXlet(), finalizando así la ejecución del xlet. En este estado se deberán liberar los recursos utilizados, listeners, etc…

3.2. Modelo grafico de MHP

Por último comentar una de las partes posiblemente más complejas del estándar, el modelo gráfico. En ella debemos afrontar una serie de problemas:

- Los pixeles de la TV no son cuadrados (en el PC sí)
- La relación de aspecto puede cambiar: 4/3, 16/9, 14/9
- Los gráficos han de poder ser transparentes
- No existe un gestor de ventanas

ISBN: 978-84-9012-858-9
Colección Aquilafuente: AQ-238
Ediciones Universidad de Salamanca - cc by nc dc

- Y la dificultad de entrada de datos, ya que disponemos de un mando a distancia como periférico de control.

El modelo grafico se encuentra dividido en tres capas:
El **Background Layer**, que por lo general muestra un fondo solido (aunque algunos dispositivos permiten mostrar una imagen estática).
El **Video Layer**, que muestra el contenido emitido. Por lo general ocupa toda la pantalla por lo que oculta a la capa de fondo. Permite redimensionar su tamaño y situarlo en el lugar deseado ya que puede resultar frustrante para el usuario si se oculta completamente.
Y el **Grafics Layer**, siendo esta la capa donde se renderizan los componentes de Java (botones, etiquetas, etc...)

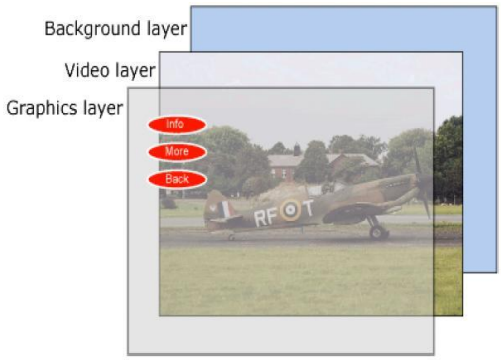


Figura 82. Modelo Gráfico

MHP agrupa las tres capas en el HScreen (HAVI).
MHP establece un mínimo de resolución de 720x576 pixeles (para las tres capas) y 256 colores para el Grafics Layer, pero puede ser mayor dependiendo del receptor [16-20].
La Grafics Layer no está obligada a soportar pixeles cuadrados, aunque si se daría el caso en resoluciones de 768x576 en 4/3 Dysplays y en resoluciones de 1024x576 en 16/9 Dysplays.

3.1.1 Sistemas de Coordenadas

Dado que existen tres capas que se superponen unas a otras, veamos los sistemas de coordenadas:
Mediante el esquema de **coordenadas normalizadas** se ofrece la posibilidad de establecer posiciones y tamaños independientes de la resolución y aspecto de la imagen. En ella, la esquina superior izquierda corresponde con los valores (0,0) y la esquina inferior derecha con (1,1).
El esquema de **coordenadas de pantalla** establece como origen (0,0) la esquina superior izquierda, al igual que en el esquema normalizado. En cambio, la esquina inferior derecha (X,Y)(720?, 576?) no se encuentran definidos, dependiendo de la propia pantalla en la que queramos visionarlo.
Y por último las **coordenadas en AWT**, basada en pixeles en el que su dimensión final viene dada por el área que ocupa el contenedor principal.
Por tanto, ¿con qué sistemas trabajaremos?
Una solución es pintar un lienzo principal (HGraficsDevice) en coordenadas normalizadas, el cual a no ser que lo modifiquemos, ocupara toda la pantalla y que a su vez contendrá las HScenes (ya que no disponemos del gestor de ventanas). Estos org.havi.ui.HScenes son paneles que se colocan sobre el "lienzo" principal (HGrafiscDevice), pudiendo definir su posición y tamaño mediante coordenadas normalizadas.
Una vez situado el contenedor HScene en la capa gráfica, los componentes que le añadamos usaran las coordenadas en pixeles.

ISBN: 978-84-9012-858-9

Colección Aquilafuente: AQ-238
Ediciones Universidad de Salamanca - cc by nc dc

3.2 Hola Mundo

Una vez que disponemos de unos conocimientos básicos vamos a ver cómo debemos crear nuestro primer Xlet que escribirá el texto "Hola Mundo" por pantalla.
Como ya dijimos anteriormente, necesitaremos crear una clase llamada MainXlet que implemente la interfaz "javax.tv.xlet.Xlet".

```
public class MainXlet implements Xlet {
public void destroyXlet(boolean param) throws XletStateChangeException {
// TODO Auto-generated method stub
}
public void initXlet(XletContext xletContext) throws XletStateChangeException {
// TODO Auto-generated method stub
}
public void pauseXlet() {
// TODO Auto-generated method stub
}
public void startXlet() throws XletStateChangeException {
// TODO Auto-generated method stub
}
}
```

Como se ve en el código, es necesario implementar 4 métodos. Explicaremos brevemente cada uno de ellos.

- **public void initXlet(XletContext xletContext) throws XletStateChangeException**, este método es siempre llamado cuando una instancia de este Xlet es creada.
- **public void startXlet() throws XletStateChangeException**, este método es invocado cuando se arranca este Xlet.
- **public void destroyXlet(boolean param) throws XletStateChangeException**, cuando el Xlet es destruido se ejecutara antes lo incluido en este método.
- **public void pauseXlet()**, cuando el Xlet se encuentre pausado, se ejecutara lo incluido en este método.

Se recomienda que cuando lancemos la aplicación desde el método startXlet se realice a través de una hebra nueva para que el método startXlet termine lo más rápido posible.
Lo primero que tenemos que hacer es definir nuestra escena, que es un componente que es creado por la implementación del **MHP**. La escena es el componente a partir del cual podemos agregar otros componentes para que sean mostrados en la pantalla. El componente es ***HScene***, y es creado de la forma siguiente:

```
HSceneFactory factory = HSceneFactory.getInstance();
HScene scene = factory.getDefaultHScene();
scene.setBounds(0,0,720,576);
scene.setVisible(true);
scene.requestFocus();
```

De la escena podemos definir su tamaño, si es visible o no. En resumen como cualquier otro **Container.**
A la escena le podemos añadir cualquier tipo de componente perteneciente al paquete *org.havi.ui.**, como cualquier componente creado por nosotros a partir de la clase *HContainer* o *HComponent*. Más adelante se mostrará cómo crear estos componentes.
Ejemplo:

```
package es.gpm.mhp;
import java.awt.Color;
import javax.tv.xlet.Xlet;
import javax.tv.xlet.XletContext;
```


ISBN: 978-84-9012-858-9
Colección Aquilafuente: AQ-238
Ediciones Universidad de Salamanca - cc by nc dc

```
import javax.tv.xlet.XletStateChangeException;
import org.havi.ui.HScene;
import org.havi.ui.HSceneFactory;
import org.havi.ui.HText;
public class MainXlet implements Xlet,Runnable {
public void destroyXlet(boolean arg0) throws XletStateChangeException {
// TODO Auto-generated method stub
}
public void initXlet(XletContext arg0) throws XletStateChangeException {
// TODO Auto-generated method stub
}
public void pauseXlet() {
// TODO Auto-generated method stub
}
public void startXlet() throws XletStateChangeException {
Thread thread = new Thread(this);
thread.start();
}
public void run() {
HSceneFactory factory = HSceneFactory.getInstance();
HScene scene = factory.getDefaultHScene();
scene.setBounds(0,0,720,576);
scene.setVisible(true);
scene.requestFocus();
HText text = new HText("Hola Mundo");
text.setBounds(150, 150, 100, 50);
text.setBackground(Color.WHITE);
scene.add(text);
text.setVisible(true);
text.requestFocus();
}
}
```

Este código crea un escena con tamaño 720x576, en el punto 0,0 de pantalla, al que le añadimos un componente de texto.

4 Componentes MHP

4.1 Componentes Básicos

4.1.1 HText

El componente **HText** es un componente de interfaz de usuario utilizado para mostrar contenido de texto estático y que además permite navegar hasta él, es decir, es un componente que puede recibir el foco.

Usabilidad

- Para utilizar este componente, solamente es necesario construir el objeto y establecer el texto que deseamos mostrar en el mismo. Después lo añadiremos al contenedor deseado.

Código de ejemplo

- Inicialización

```
HText _textRed = new HText("");
```

ISBN: 978-84-9012-858-9

Colección Aquilafuente: AQ-238
Ediciones Universidad de Salamanca - cc by nc dc

```
_textRed.setForeground(DVBColor.black);
_textRed.setBackgroundMode(HVisible.NO_BACKGROUND_FILL);
_textRed.setFont(FontRepository.LABEL_BUTTON_FONT);
_textRed.setHorizontalAlignment(0);
_textRed.setTextContent("Complementos", HState.ALL_STATES);
```

- Inclusión en pantalla

```
HContainer container = new HContainer()
container.add(_textRed);
```

4.1.2 HIcon

El componente **HIcon** es un componente de interfaz de usuario utilizado para mostrar contenido gráfico estático y que además permite navegar hasta él, es decir, es un componente que puede recibir el foco.

Usabilidad

- Para utilizar este componente solamente es necesario cargar la imagen que formará el icono, es decir, vamos a crear el componente de icono a partir de la propia imagen y después lo añadiremos al contenedor deseado.

Código de ejemplo

- Inicialización

```
private Image _imgButtonRed = ImageUtil.loadImage(this, "img/pimiento_rojo.png");
private HIcon _icoButtonRed = new HIcon(_imgButtonRed);
```

- Inclusión en pantalla

```
HContainer container = new HContainer()
container.add(_imgButtonRed);
```

5 Uso de Layouts

Los componentes se pueden colocar estableciendo una posición fija, indicándoles un punto de origen y unas dimensiones (alto y ancho). Pero puede darse el caso que queramos que se redimensione o ajuste al tamaño de la pantalla.
Para resolver esta situación, podemos usar los Layouts, especificando la apariencia que tomarán los componentes al colocarlos sobre un Contenedor [21-25].

5.1 BorderLayout

5.1.1 Usando BorderLayout

Para comprender mejor el funcionamiento vamos a explicar los pasos para realizar una separación por regiones de un contenedor.

ISBN: 978-84-9012-858-9
Colección Aquilafuente: AQ-238
Ediciones Universidad de Salamanca - cc by nc dc

Lo primero que necesitamos definir es un contenedor. Para ello generamos un nuevo proyecto siguiendo los pasos del ejemplo “Hola mundo”, creando de este modo un método “run“ tal que:

```
public void run() {
HSceneFactory factory = HSceneFactory.getInstance();
HScene scene = factory.getDefaultHScene();
//Definimos las dimensiones del contenedor
scene.setBounds(5, 5, 710, 566);
//Hacemos que la escena sea visible
scene.setVisible(true);
… }
```

Usaremos HScene como contenedor principal (ya que extiende de Container) y en él iremos definiendo las distintas zonas donde incluir componentes.

Viendo el código, observamos que inicializamos el contendor con coordenadas. Si no seteamos estas dimensiones, por defecto tomará las de toda la pantalla como marco.

Una vez que disponemos del contenedor, lo habilitamos para el uso de Layouts. De tal manera, que podamos definir la región en la que se quiere que se pinten los componentes que añadamos.

```
scene.setLayout(new BorderLayout());
```

Normalmente la forma de trabajar con los Layouts, sería definir distintas regiones en un contenedor, y en cada una de ellas definiríamos nuevos contenedores hasta que quedasen maquetados a nuestro gusto. Para simplificar el ejemplo, usaremos HText, que abarcarán por completo las regiones.

Así pues, construimos un HText que usaremos para definir la zona central.

```
HText textoCentral = new HText("Central");
textoCentral.setBackground(DVBColor.white);
textoCentral.setForeground(DVBColor.black);
```

Y lo añadimos a la scene (que como hemos dicho es nuestro contenedor principal), definiendo la región en la que deseamos colocarlo:

```
scene.add(textoCentral, BorderLayout.CENTER);
```

Por último validamos el contenedor para que calcule las dimensiones y repintamos:

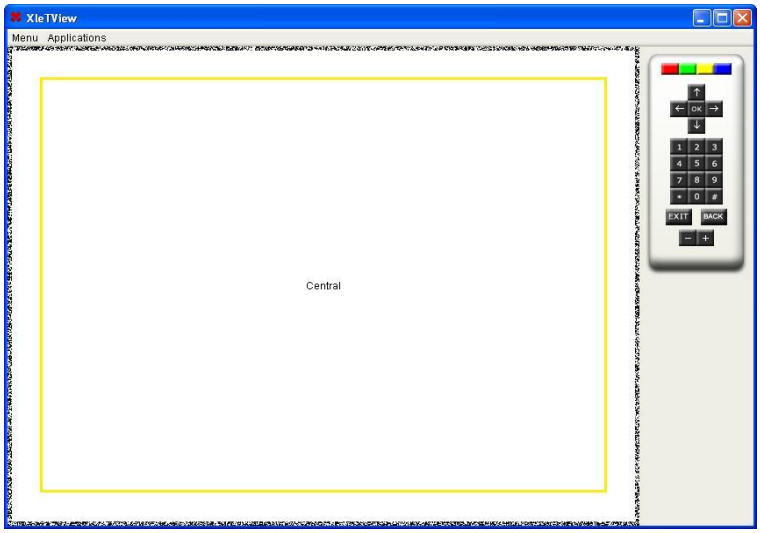
```
// Valida las coordenadas y tamaño en función de las coordenadas
scene.validate();
// Repintamos las escena
scene.repaint();
```

La región central tiene la peculiaridad de que SIEMPRE se expandirá al máximo disponible. Por tanto, ya que por ahora no tenemos definidos componentes en el resto de las regiones, ocuparemos con el HText toda la scene.

Si ejecutamos el código podemos ver claramente lo expuesto:

ISBN: 978-84-9012-858-9

Colección Aquilafuente: AQ-238
Ediciones Universidad de Salamanca - cc by nc dc



Para comprobar cómo redimensiona, vamos a introducir un componente en la región norte. Para ello construimos otro HText:

```
HText textoNorte = new HText("Central");
textoNorte.setBackground(DVBColor.red);
textoNorte.setForeground(DVBColor.black);
//Definimos la altura
textoNorte.setSize(0,100);
```

Si comparamos el nuevo HText norte con el central, podemos ver que le hemos seteado un tamaño. El método recibe dos parámetros, uno para el alto y otro para el ancho (setSize(int width, int height)) indicando mediante enteros los pixeles reservados para el componente.

Para los componentes que se sitúen en las regiones NORTE y SUR, es necesario indicarles una altura, de modo que al realizar el cálculo de las dimensiones sepa el espacio final que tiene que asignar a cada parte. En cambio, el valor del ancho será ignorado, ya que siempre se expandirá al máximo posible [26-30].

Para los componentes que se sitúan en las regiones ESTE y OESTE, lo que tendremos que definir es el valor del ancho, ignorando la altura ya que siempre tomará la máxima disponible.

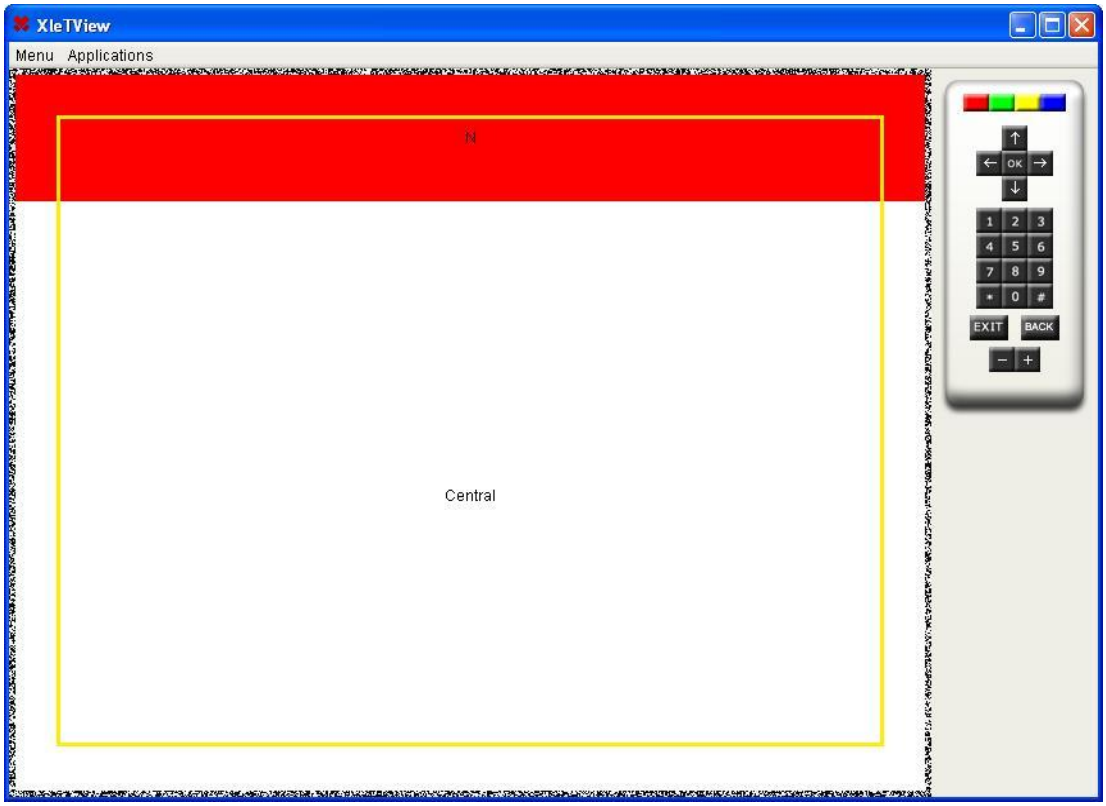
En el caso de la región CENTRAL, como ya hemos comentado anteriormente, no será necesario indicarle ningún tamaño, ya que siempre ocupará el máximo espacio disponible que le quede.

Te tal modo, que si asignamos el nuevo HText norte a la scene:

```
scene.add(textoNorte, BorderLayout.NORTH);
```

Y ejecutamos resultará:

ISBN: 978-84-9012-858-9
Colección Aquilafuente: AQ-238
Ediciones Universidad de Salamanca - cc by nc dc



El código que tendríamos hasta ahora sería:

```
//Obtenemos la escena (contenedor donde pintaremos componentes)
HSceneFactory factory = HSceneFactory.getInstance();
HScene scene = factory.getDefaultHScene();
//Definimos las dimensiones del contenedor
scene.setBounds(5, 5, 710, 566);
//Hacemos que la escena sea visible
scene.setVisible(true);
//Definimos el Layout
scene.setLayout(new BorderLayout());
// *******
// Creamos un componente para la region CENTRAL
HText textoCentral = new HText("Central");
textoCentral.setBackground(DVBColor.white);
textoCentral.setForeground(DVBColor.black);
// *******
// Creamos un componente para la region NORTE
HText textoNorte = new HText("Central");
textoNorte.setBackground(DVBColor.red);
textoNorte.setForeground(DVBColor.black);
//Definimos la altura
textoNorte.setSize(0,100);
//Añadimos los componentes al contenedor
scene.add(textoCentral, BorderLayout.CENTER);
scene.add(textoNorte, BorderLayout.NORTH);
// Valida las coordenadas y tamaño en función de las coordenadas
scene.validate();
// Repintamos las escena
scene.repaint();
```

ISBN: 978-84-9012-858-9

Colección Aquilafuente: AQ-238
Ediciones Universidad de Salamanca - cc by nc dc

5.1.2 Asignación de las zonas

A la hora de añadir un componente a un contenedor que se encuentra habilitado para el uso de BorderLayout disponemos de las siguientes constantes:

- BorderLayout.NORTH
- BorderLayout.SOUTH
- BorderLayout.EAST
- BorderLayout.WEST
- BorderLayout.CENTER

De tal modo que si deseamos añadir un componente en la región norte (en este caso otro contenedor):

```
…
container.setLayout(new Borderlayout());
container.add(new HContainer(), BorderLayout.NORTH);
…
```

Una peculiaridad es que a la hora de asignar en la región central de un contenedor no es necesario que se lo indiquemos obligatoriamente, ya que será la región por defecto. Por tanto, para la región central, las siguientes líneas de código son equivalentes:

```
…
container.setLayout(new Borderlayout());
container.add(new HContainer());
…

…
container.setLayout(new Borderlayout());
container.add(new HContainer(), BorderLayout.CENTRAL);
…
```

Lógicamente, en el resto de las situaciones (NORTE, SUR, ESTE y OESTE) deberemos indicárselo.

5.1.3 BorderLayout con Gap

Al definir el layout de un contenedor podemos definir que deseamos que exista una separación entre cada uno de los componentes asignados a las distintas regiones.

Hasta ahora estábamos utilizando sin parámetros a la hora de definirlo:

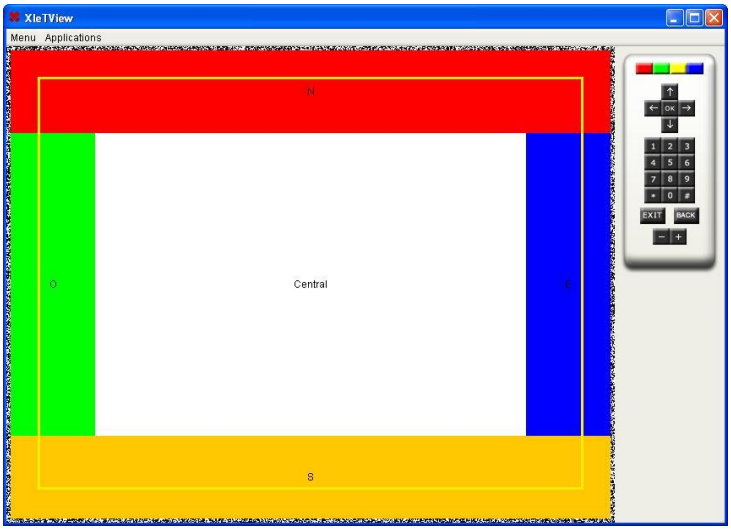
```
container.setLayout(new Borderlayout());
```

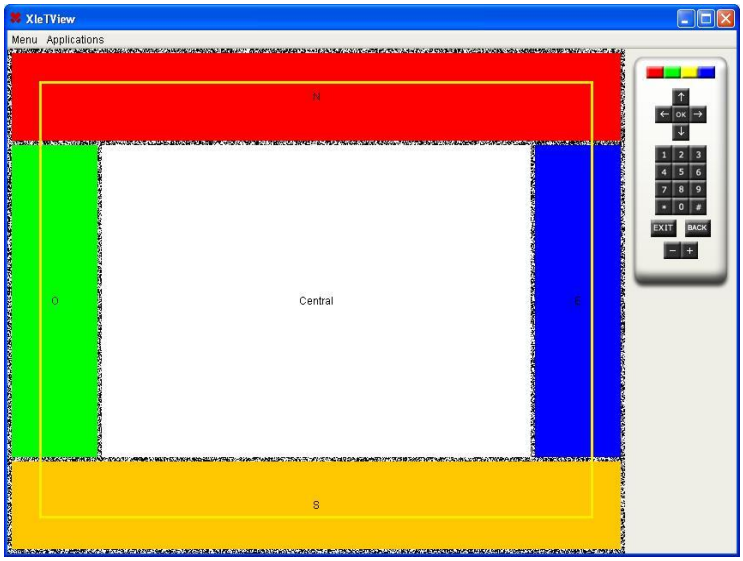
Pero existe un constructor que nos permite pasarle la separación o Gap deseado:

```
container.setLayout(new Borderlayout(5,5));
```

Podemos apreciar la diferencia en las siguientes imágenes:

ISBN: 978-84-9012-858-9
Colección Aquilafuente: AQ-238
Ediciones Universidad de Salamanca - cc by nc dc





5.2 GridLayout

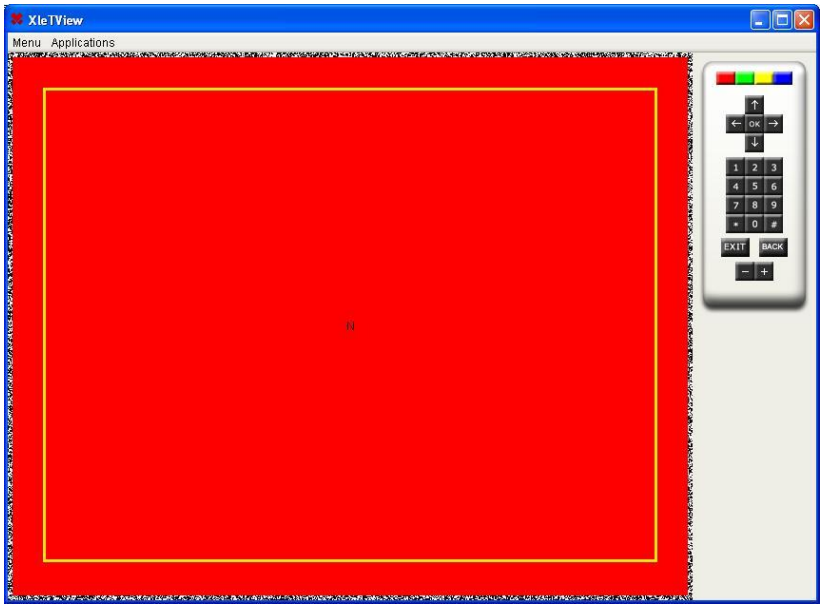
Otra forma de realizar separaciones de componentes es mediante el uso de GridLayout. En este caso separaremos la región que nos interese en celdas que se distribuirán a partir de dos valores que representarán filas y columnas. (Es la representación gráfica de una tabla)
Para realizar este tipo de distribución, al igual que hacíamos con BorderLayout, lo primero que tenemos que hacer es asignárselo al contenedor:

```
scene.setLayout(new GridLayout());
```

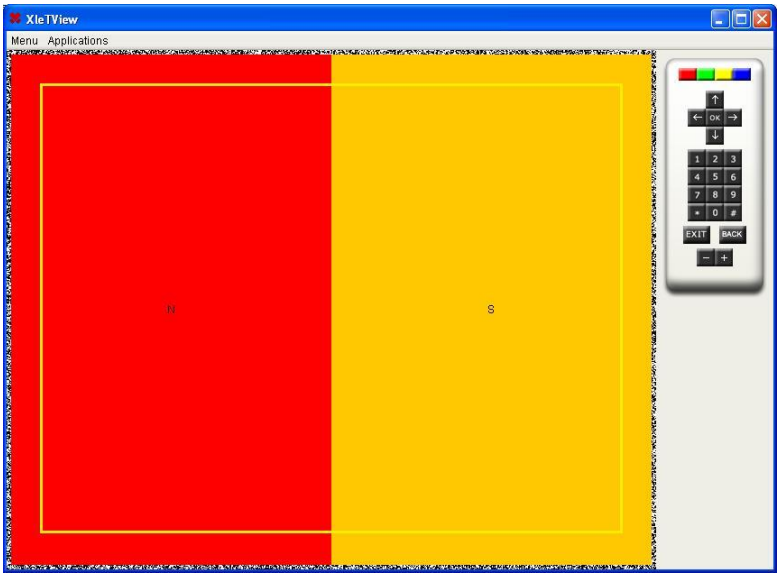
Utilizando el constructor por defecto, definimos que existirá una sola fila, de tal forma que el contenedor se dividirá de forma proporcional según los componentes que le añadamos. Así pues, si añadimos un solo componente, se expandirá hasta abarcar todo el contenedor:

ISBN: 978-84-9012-858-9

Colección Aquilafuente: AQ-238
Ediciones Universidad de Salamanca - cc by nc dc

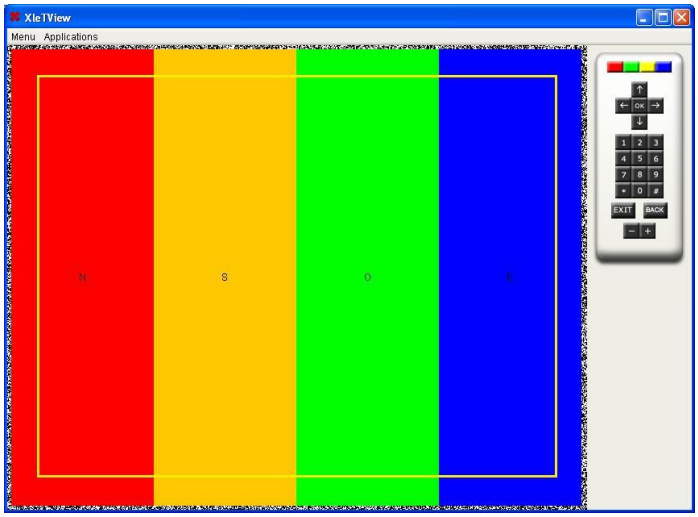


Si añadimos dos componentes:



Y si añadimos cuatro:

ISBN: 978-84-9012-858-9
Colección Aquilafuente: AQ-238
Ediciones Universidad de Salamanca - cc by nc dc



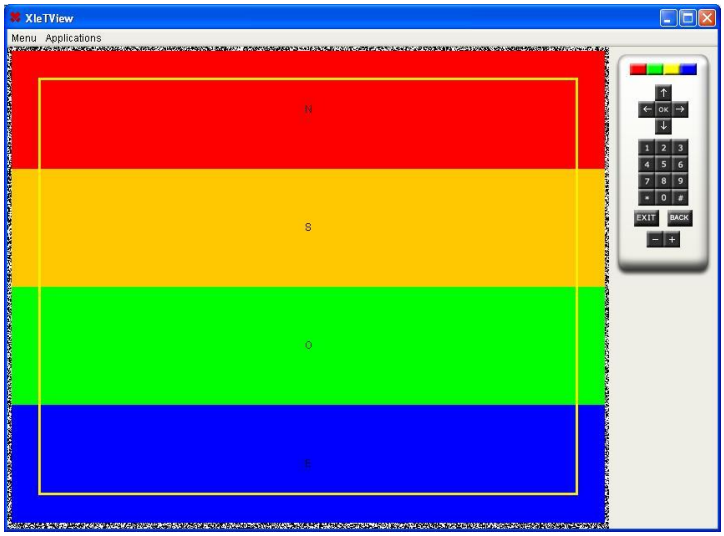
Esta sería la utilización “por defecto”, pero disponemos de dos constructores adicionales por si estas características no se ajustan a nuestras necesidades:

```
public GridLayout(int rows, int cols)
public GridLayout(int rows, int cols, int hgap, int vgap)
```

En un primer caso podemos definir el número de filas y columnas que deseamos que contenga. Y el segundo constructor nos permite definirle un margen horizontal o vertical de separación entre componentes que adjuntemos al contenedor.
De tal modo que si queremos que cada componente se coloque en una fila distinta (sabiendo que son 4 los que queremos adjuntar):

```
scene.setLayout(new GridLayout(4,1));
```

Mostrando por pantalla:

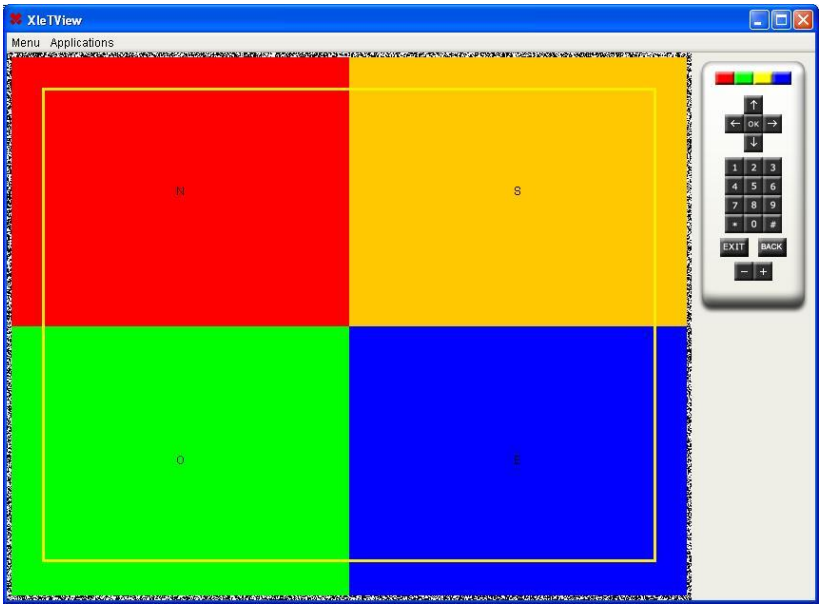


O si queremos que se pinte una tabla de 2x2:

ISBN: 978-84-9012-858-9

Colección Aquilafuente: AQ-238
Ediciones Universidad de Salamanca - cc by nc dc

```
scene.setLayout(new GridLayout(2,2));
```



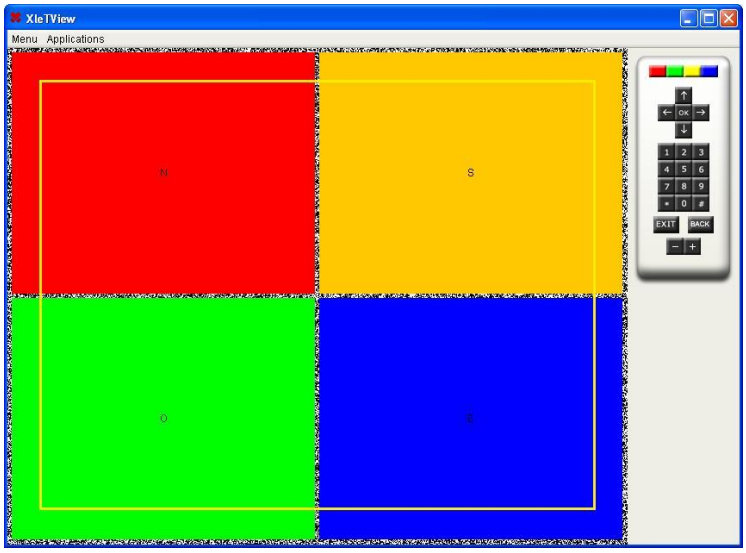
El orden de colocación de los distintos componentes siempre se realizará de forma secuencial, completando primero la fila inicial de izquierda a derecha, y saltando a la siguiente fila en el momento que lleguemos a número de columnas que le hayamos definido.
Por tanto, para este último ejemplo, tenemos que los componentes se han añadido en el siguiente orden:

```
scene.setLayout(new GridLayout(2,2));
scene.add(LayoutComponents.getTextNorth(true, 0)); //Letra N
scene.add(LayoutComponents.getTextSouth(true, 0)); //Letra S
scene.add(LayoutComponents.getTextWest(true, 0)); //Letra O
scene.add(LayoutComponents.getTextEast(true, 0)); //Letra E
```

Si le añadimos un margen de 5 pixeles (horizontal y vertical), se nos mostraría:

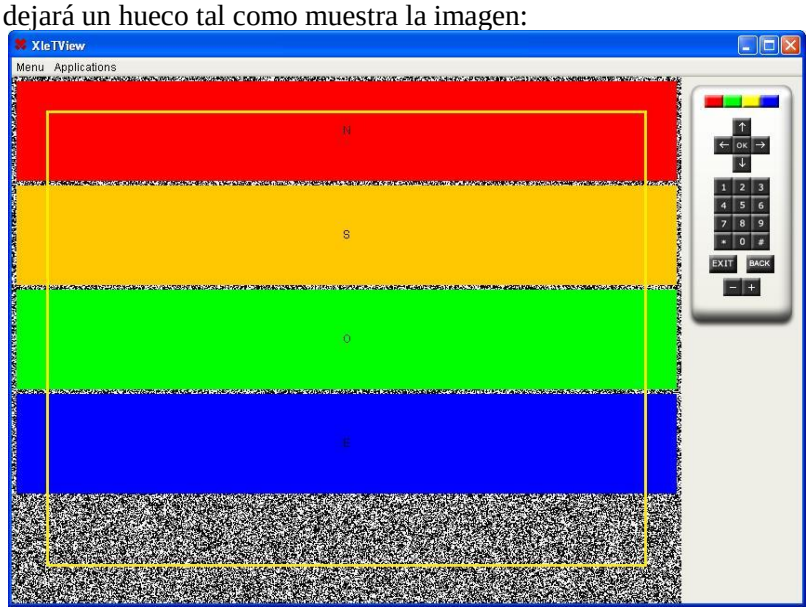
```
scene.setLayout(new GridLayout(2,2,5,5));
scene.add(LayoutComponents.getTextNorth(true, 0)); //Letra N
scene.add(LayoutComponents.getTextSouth(true, 0)); //Letra S
scene.add(LayoutComponents.getTextWest(true, 0)); //Letra O
scene.add(LayoutComponents.getTextEast(true, 0)); //Letra E
```


ISBN: 978-84-9012-858-9
Colección Aquilafuente: AQ-238
Ediciones Universidad de Salamanca - cc by nc dc



5.2.1 Notas sobre GridLayout

Anteriormente mencionamos que cuando usábamos BorderLayout, teníamos que definir un alto para la zona Norte y Sur, y un ancho para la Este y Oeste, de tal modo que pudiese calcular cuánto se tenía que expandir la región central.
Usando GridLayout, esto no será necesario, ya que lo harán las distribuciones automáticamente tomando como referencia tan solo el número de "celdas que deseemos añadir".
Por tanto, si definimos que el contenedor va a disponer de 5 componentes, pero realmente sólo le insertamos 4, dejará un hueco tal como muestra la imagen:



6 Creación de componentes

En primer lugar definamos a qué podemos llamar un componente.

ISBN: 978-84-9012-858-9

Colección Aquilafuente: AQ-238
Ediciones Universidad de Salamanca - cc by nc dc

Para nosotros será cualquier objeto que pueda ser agregado a un Scene o en su defecto a *org.havi.ui.HContainer*. No todos los componentes tienen por qué ser interactivos, es decir, que recojan eventos e interactúen a las teclas del mando y ni siquiera que se pinten en la pantalla, aunque es lo más habitual [31-33].
¿Para qué se podría usar un componente? Lo más obvio es para pintar un texto en la pantalla, el cual una vez pintado, no interactúa con ningún evento. Alguna utilidad más avanzada podría ser un campo de texto el cual nos permitiría introducir texto a través del mando.
Un componente que no llegue a pintarse nunca, podría ser algún componente que quisiéramos que estuviera allí realizando una tarea en background. Pero no es lo más común.
Lo primero para entender cómo crear un componente, es saber un poco de AWT. Por lo que si no sabes nada de AWT, es recomendable leer un poco sobre el tema antes de continuar. Y es que los componentes *org.havi.ui.*HContainer y org.havi.ui.HComponent heredan respectivamente de *java.awt.Container* y *java.awt.Component*.
Debido a esta herencia, se adquiere mucha funcionalidad de AWT, aunque no podemos decir que dispongamos al cien por cien de ella, ya que habrá aspectos que no tendrán sentido en un entorno de televisión, como pueden ser métodos para el uso del ratón, etc.

6.1 Cómo pintar un componente

Todos los componentes tienen un método *public void paint(Graphics g){ ...}*. Este método es llamado cada vez que es necesario pintar o repintar este componente en la pantalla. Al igual que en AWT, esto se produce en cascada. Esto quiere decir que si a un componente se le invoca método *paint*, también será invocado el método *paint* de todos los componentes que contenga. Normalmente la llamada a este método es realizada por el sistema, pero también es posible hacerlo de forma explícita llamando al método *public void repaint().*
Una vez dentro del método *paint*, la clase que tiene todos los métodos de pintado es *java.awt.Graphics***.** Mirando brevemente la API veremos que contiene métodos de pintado de texto, líneas, polígonos, etc.
A través del siguiente ejemplo explicaremos las principales características que hay que tener en cuenta a la hora de pintar un componente.

```
package es.gpm.mhp.ui;
import java.awt.Dimension;
import java.awt.FontMetrics;
import java.awt.Graphics;
public class MiComponente extends HComponent {
private String _text;
public void paint(Graphics g){
Dimension dimension = getSize();
g.setColor(getBackground());
g.fillRect(0, 0, dimension.width, dimension.height);
g.setColor(getForeground());
FontMetrics fontMetrics = g.getFontMetrics();
if(_text!=null)
g.drawString(_text, (dimension.width-fontMetrics.stringWidth(_text))/2, dimension.height/2+fontMetrics.getDescent());
}
public String getText() {
return _text;
}
public void setText(String text) {
_text = text;
}
}
```

Básicamente lo que hace este componente es pintar un texto en pantalla, al cual le podemos especificar el color del fondo, del texto y la fuente. El texto además aparecerá centrado en el rectángulo que lo define.

ISBN: 978-84-9012-858-9
Colección Aquilafuente: AQ-238

Pasos:

- Creación de un rectángulo con color sólido.

```
Dimension dimension = getSize();
g.setColor(getBackground());
g.fillRect(0, 0, dimension.width-1, dimension.height-1);
```

Para obtener el tamaño de un componente se recomienda usar el método ***getSize*** en vez del de *getWidth* o *getHeight*. Ya que no todos los decos lo implementan correctamente. Lo mismo ***para la clase Dimension***, *usar los* ***atributos width*** *y* ***height*** *en vez de los métodos*.
Usamos el método getBackground que es heredado de HComponent para obtener el color del componente y se lo asignamos al objeto Graphics, con lo que a partir de ahí todo lo que pintemos tendrá el color asignado.

- Pintado del Texto.

```
g.setColor(getForeground());
FontMetrics fontMetrics = g.getFontMetrics();
if(_text!=null)
g.drawString(_text, (dimension.width-1-fontMetrics.stringWidth(_text))/2, dimension.height/2+fontMetrics.getDescent());
```

Ahora usamos el método getForeground() para asignar el color del texto.
La clase FontMetrics nos permite saber el tamaño del texto en función de la fuente usada.
Y por último, llamamos al método drawString de Graphics para pintar el texto.
Como se ve, es muy simple pintar un componente, el límite está solamente en la complejidad que queramos darle.
Revisando la API se pueden ver los múltiples métodos de pintado que tiene el Graphics y que nos da más posibilidades de las mostradas en el ejemplo.
Una recomendación es que para fijar el tamaño de un componente, color, fuente y otros parámetros no usar ningún valor en concreto dentro del método paint, sino usar siempre los métodos heredados del HComponent. Esto dará una gran versatilidad a nuestros componentes ya que podremos asignárselos a posteriori en ejecución.

6.2 Manejar eventos

Un componente puede recibir una serie de eventos que le informan de que algo ha sucedido. Si el componente tiene dado de alta un escuchador de un tipo en concreto de eventos podremos realizar alguna acción cuando éstos sean detectados. Como por ejemplo que una tecla del mando se ha pulsado, que el foco está en el componente, o que el foco se ha perdido del componente. Esta parte es fundamental ya que nos permite hacer que nuestros objetos puedan ser interactivos.
En este apartado nos fijaremos sólo en los eventos del mando, y dejando para el siguiente punto todo lo que tiene que ver con el foco. Eso sí, sólo el componente que sea dueño del foco podrá recibir los eventos del pulsado de las teclas del mando.
Para ilustrar esto, lo haremos a través de un ejemplo. Cogeremos el componente de Texto creado anteriormente y haremos que esté escuchando si se ha pulsado las teclas del cursor y la de OK, posicionando el texto según la tecla pulsada.

```
package es.gpm.mhp;
import java.awt.Dimension;
import java.awt.FontMetrics;
import java.awt.Graphics;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
```

ISBN: 978-84-9012-858-9

Colección Aquilafuente: AQ-238
Ediciones Universidad de Salamanca - cc by nc dc

```
import org.havi.ui.HComponent;
public class OldLabel extends HComponent implements KeyListener {
private String _text;
public static int ARRIBA = 0;
public static int ABAJO = 1;
public static int DERECHA = 2;
public static int IZQUIERDA = 3;
public static int CENTRO = 4;
private int _posicion=CENTRO;
private int BORDER =5;
public String getText() {
return _text;
}
public void setText(String text) {
_text = text;
}
public OldLabel(String text) {
_text = text;
addKeyListener(this);
}
public void paint(Graphics g) {
Dimension d = getSize();
g.setColor(getBackground());
g.fillRect(0, 0, d.width-1, d.height-1);
FontMetrics fontMetrics = g.getFontMetrics();
g.setColor(getForeground());
if(_posicion==ARRIBA){
g.drawString(getText(), (d.width-1-fontMetrics.stringWidth(getText()))/2, fontMetrics.getMaxAscent()+BORDER);
}else if(_posicion==ABAJO){
g.drawString(getText(), (d.width-1-fontMetrics.stringWidth(getText()))/2, (d.height-1)-BORDER);
}else if(_posicion==DERECHA){
g.drawString(getText(), (d.width-1-fontMetrics.stringWidth(getText())-BORDER), (d.height-1)/2+fontMetrics.getMaxAscent()/2);
}else if(_posicion==IZQUIERDA){
g.drawString(getText(), BORDER, (d.height-1)/2+fontMetrics.getMaxAscent()/2);
}else if(_posicion==CENTRO){
g.drawString(getText(), (d.width-1-fontMetrics.stringWidth(getText()))/2, (d.height-1)/2+fontMetrics.getMaxAscent()/2);
}
}
<!--[if !supportAnnotations]--
>http://formacion.gpm.int/file.php/25/moddata/scorm/6/52_manejar_eventos.html - _mso-com_1#_msocom_1
public int getPosicion() {
return _posicion;
}
public void setPosicion(int posicion) {
_posicion = posicion;
}
public void keyPressed(KeyEvent e) {
if(e.getKeyCode() == KeyEvent.VK_UP){
setPosicion(ARRIBA);
}else if(e.getKeyCode() == KeyEvent.VK_DOWN){
setPosicion(ABAJO);
}else if(e.getKeyCode() == KeyEvent.VK_LEFT){
setPosicion(IZQUIERDA);
}else if(e.getKeyCode() == KeyEvent.VK_RIGHT){
setPosicion(DERECHA);
}else if(e.getKeyCode() == KeyEvent.VK_ENTER){
setPosicion(CENTRO);
}
repaint();
System.err.println(_posicion);
}
public void keyReleased(KeyEvent e) {
System.err.println("keyReleased");
}
public void keyTyped(KeyEvent e) {
```


ISBN: 978-84-9012-858-9
Colección Aquilafuente: AQ-238
Ediciones Universidad de Salamanca - cc by nc dc

```
// TODO Auto-generated method stub
}
}
```

Las clases que heredan de HComponent tienen un método llamado void addKeyListener(KeyListener). De esta forma podremos dar de alta un escuchador.

Como se ve en el ejemplo el propio componente implementa la interfaz KeyListener que tiene la siguiente definición:

```
package java.awt.event;
import java.util.EventListener;
public interface KeyListener extends EventListener {
public void keyTyped(KeyEvent e);
public void keyPressed(KeyEvent e);
public void keyReleased(KeyEvent e);
}
```

Estos métodos que implementamos serán llamados cuando sea pulsado o soltado un botón del mando. El *KeyEvent* nos indicará qué tecla fue pulsada. Y una vez capturada la tecla podemos decidir qué acción realizar. En nuestro caso, cambiamos la posición del texto, e invocamos al *repaint* para que actualice el componente en la pantalla.
Un detalle interesante es que un componente puede tener dado de alta varios escuchadores a la vez, los cuales serán invocados cuando un evento es lanzado. No se puede garantizar en qué orden será llamados los escuchadores.
Otra opción útil es también que un escuchador sea dado de alta en múltiples componentes, esto permite centralizar el manejo de eventos.

6.3 El uso del foco

El foco es lo que indica cuando un componente puede recibir eventos, o de otra forma dicha, cuando un componente está activo para interactuar con él. Es importante en una aplicación TDT saber en todo momento qué componente es dueño del foco para saber exactamente qué acciones vamos a poder realizar.
Existe un evento llamado *java.awt.event.Focus* que será lanzado cuando el componente reciba el foco o cuando lo pierda. Para capturarlo sólo es necesario dar de alta un escuchador para este tipo de eventos, y para ello implementaremos la interfaz *java.awt.event.FocusListener*.

```
package java.awt.event;
import java.util.EventListener;
public interface FocusListener extends EventListener {
/**
* Invoked when a component gains the keyboard focus.
*/
public void focusGained(FocusEvent e);
/**
* Invoked when a component loses the keyboard focus.
*/
public void focusLost(FocusEvent e);

}
```

Para ver cómo usar esto en un caso práctico, crearemos un componente que pinta un texto, y que mostrará un borde coloreado cuando el foco se encuentre en él.

```
package es.gpm.mhp;
import java.awt.Dimension;
import java.awt.FontMetrics;
import java.awt.Graphics;
import java.awt.event.FocusEvent;
import java.awt.event.FocusListener;
```

ISBN: 978-84-9012-858-9

Colección Aquilafuente: AQ-238
Ediciones Universidad de Salamanca - cc by nc dc

```
import org.dvb.ui.DVBColor;
import org.havi.ui.HComponent;
public class FocusLabel extends HComponent implements FocusListener {
private String _text;
private boolean _isFocus = false;
private int BORDER_FOCUS = 4;
private static final long serialVersionUID = -6865054572942050470L;
public String getText() {
return _text;
}
public void setText(String text) {
_text = text;
}
public FocusLabel(String text) {
_text = text;
addFocusListener(this);
}
public void paint(Graphics g) {
Dimension d = getSize();
pintarFoco(g, d);
g.setColor(getBackground());
g.fillRect(BORDER_FOCUS+1, BORDER_FOCUS+1, d.width - 2 - 2*BORDER_FOCUS , d.height - 2 - 2*BORDER_FOCUS);
FontMetrics fontMetrics = g.getFontMetrics();
g.setColor(getForeground());
g.drawString(getText(), (d.width - 1 - fontMetrics
.stringWidth(getText())) / 2, (d.height - 1) / 2
+ fontMetrics.getMaxAscent() / 2);
}
public void focusGained(FocusEvent e) {
_isFocus = true;
}
public void focusLost(FocusEvent e) {
_isFocus = false;
}
}
```

Como se puede ver en el ejemplo, la clase implementa la interfaz *java.awt.event.FocusListener.*

```
public class FocusLabel extends HComponent implements FocusListener {
```

Y en el constructor da de alta el escuchador pasando una referencia de sí mismo:

```
public FocusLabel(String text) {
_text = text;
addFocusListener(this);
}
```

Lo que se hace es crear una variable que indica si el foco lo tiene el componente.

```
private boolean _isFocus = false;
```

Y en la implementación de la interfaz actualizamos esta variable según sea el caso y realizaremos un repintado para que pinte el nuevo estado del componente:

```
public void focusGained(FocusEvent e) {
_isFocus = true;
repaint();
}
public void focusLost(FocusEvent e) {
_isFocus = false;
repaint();
}
```

Y ya por último hemos agregado unas líneas de código que se encargan de pintar el recuadro en el caso de que el componente sea dueño del foco:

```
if(_isFocus){
g.setColor(DVBColor.red);
}
else{
g.setColor(getBackground());
}
```


ISBN: 978-84-9012-858-9
Colección Aquilafuente: AQ-238
Ediciones Universidad de Salamanca - cc by nc dc

```
g.fillRect(0, 0, d.width - 1, d.height - 1);
```

Hasta ahora hemos explicado qué ocurre cuando un componente recibe el foco o lo pierde, pero ¿cómo podemos hacer que el foco se mueva entre diferentes componentes? Pues sencillo, el mismo componente debe solicitarlo, para ello tiene un método *public void requestFocus()*.

6.3.1 Transferencia del foco entre componentes

Hasta ahora hemos dicho que la forma de transferir el foco de un componente a otro es que el componente que deseemos que lo adquiera lo solicite. Pero, ¿cómo hacer esto cuando sólo el que tiene el foco puede recibir eventos? La solución es sencilla, y se verá mucho mejor con un ejemplo. Para ello usaremos el componente creado en el punto anterior, y lo que haremos es pintar dos de ellos, y haremos que cuando se pulse la tecla del OK se transfiera el foco de uno al otro. Veamos cómo queda la clase con estas modificaciones:

```
package es.gpm.mhp;
import java.awt.Dimension;
import java.awt.FontMetrics;
import java.awt.Graphics;
import java.awt.event.FocusEvent;
import java.awt.event.FocusListener;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import org.dvb.ui.DVBColor;
import org.havi.ui.HComponent;
public class FocusLabel extends HComponent implements FocusListener,KeyListener {
private String _text;
private boolean _isFocus = false;
private int BORDER_FOCUS = 4;
private HComponent _nextFocusComponet;
private static final long serialVersionUID = -6865054572942050470L;
public String getText() {
return _text;
}
public void setText(String text) {
_text = text;
}
public FocusLabel(String text) {
_text = text;
addFocusListener(this);
addKeyListener(this);
}
public void paint(Graphics g) {
Dimension d = getSize();
pintarFoco(g, d);
g.setColor(getBackground());
g.fillRect(BORDER_FOCUS+1, BORDER_FOCUS+1, d.width - 2 - 2*BORDER_FOCUS , d.height - 2 - 2*BORDER_FOCUS);
FontMetrics fontMetrics = g.getFontMetrics();
g.setColor(getForeground());
g.drawString(getText(), (d.width - 1 - fontMetrics
.stringWidth(getText())) / 2, (d.height - 1) / 2
+ fontMetrics.getMaxAscent() / 2);
}
private void pintarFoco(Graphics g, Dimension d) {
if(_isFocus){
g.setColor(DVBColor.red);
}
else{
g.setColor(getBackground());
}
g.fillRect(0, 0, d.width - 1, d.height - 1);
}
public void focusGained(FocusEvent e) {
_isFocus = true;
repaint();
```

ISBN: 978-84-9012-858-9

Colección Aquilafuente: AQ-238
Ediciones Universidad de Salamanca - cc by nc dc

```
}
public void focusLost(FocusEvent e) {
_isFocus = false;
repaint();
}
public void keyPressed(KeyEvent e) {
if(e.getKeyCode() == KeyEvent.VK_ENTER){
if(_nextFocusComponet!=null)
_nextFocusComponet.requestFocus();
}
}
public void keyReleased(KeyEvent e) {
// TODO Auto-generated method stub
}
public void keyTyped(KeyEvent e) {
// TODO Auto-generated method stub
}
public void setNextFocusComponet(HComponent nextFocusComponet) {
_nextFocusComponet = nextFocusComponet;
}
}
```

Hemos agregado a la clase que implemente ese escuchador de teclado para detectar si se ha pulsado la tecla **OK**.

```
public class FocusLabel extends HComponent implements FocusListener,KeyListener
```

El componente tiene una referencia al objeto al que transferirá el foco cuando la tecla **OK** sea pulsada:

```
private HComponent _nextFocusComponet;
public void setNextFocusComponet(HComponent nextFocusComponet) {
_nextFocusComponet = nextFocusComponet;
}
```

Y cuando el componente es informado de que la tecla **OK** ha sido pulsada, traspasa el foco:

```
public void keyPressed(KeyEvent e) {
if(e.getKeyCode() == KeyEvent.VK_ENTER){
if(_nextFocusComponet!=null)
_nextFocusComponet.requestFocus();
}
}
```

6.3.2 Foco Transversal

En el ejemplo anterior hemos visto que si deseamos pasar el foco entre componentes, cada uno de ellos debe implementar qué teclas debe escuchar, y dependiendo de qué tecla sea, a qué componente transferir el foco, lo cual es bastante tedioso. Existe una forma de hacer esto para todos los componentes que usemos y es que nuestro componente implemente la interfaz *org.havi.ui.HNavigable*. Esta implementación hace que podamos definir las teclas de los cursores para decirle a qué objeto debe mover el foco en el caso de que alguna de las teclas sea pulsada. Si esto lo hacemos en un componente padre, cualquier otro que herede de él, tendrá dicha funcionalidad.

La definición de esta interfaz es:

```
package org.havi.ui;
public interface HNavigable extends HNavigationInputPreferred{
public void setMove(int keyCode, HNavigable target);
public HNavigable getMove(int keyCode);
public void setFocusTraversal(HNavigable up, HNavigable down, HNavigable left, HNavigable right);
public boolean isSelected();
public void setGainFocusSound(HSound sound);
public void setLoseFocusSound(HSound sound);
public HSound getGainFocusSound();
public HSound getLoseFocusSound();
public void addHFocusListener(org.havi.ui.event.HFocusListener l);
```


ISBN: 978-84-9012-858-9
Colección Aquilafuente: AQ-238
Ediciones Universidad de Salamanca - cc by nc dc

```
public void removeHFocusListener(org.havi.ui.event.HFocusListener l);
}
```

Los métodos importantes son:

```
// Este asigna el código de tecla a un component navegable.
public void setMove(int keyCode, HNavigable target);
//Obtiene el objeto navegable para esa tecla determinada
public HNavigable getMove(int keyCode);
//Asigna los componentes para las teclas del cursor
public void setFocusTraversal(HNavigable up, HNavigable down, HNavigable left, HNavigable right);
```

Por último, si queremos saber si un componente tiene asignado el foco existen dos métodos que lo indican, isFocusOwner(), hasFocus(). Desgraciadamente estos métodos no funcionan en todos los decodificadores, con lo que la recomendación es implementarse uno mismo su propio método que almacene si tiene el foco.

Es muy importante que nuestros componentes sobrescriban el método:

```
public boolean isFocusTraversable() {
return true;
}
```

6.4 Creación de Look&Feel

Algo importante en una aplicación es poder cambiar el aspecto de la misma, sin tener que crear desde cero todos los componentes. Esto sólo es posible si separamos lo que es la lógica del componente de su forma de pintarse.

Para ello MHP tiene definida una interfaz llamada *org.havi.ui.HLook* que usada en conjunto con el *org.havi.ui.HVisible* desacoplará la parte del pintado de la lógica que pueda tener un componente.

La definición de la interfaz es:

```
package org.havi.ui;
import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.Insets;
public interface HLook extends Cloneable{
public abstract void showLook(Graphics g, HVisible hvisible, int i);
public abstract void widgetChanged(HVisible hvisible, HChangeData ahchangedata[]);
public abstract Dimension getMinimumSize(HVisible hvisible);
public abstract Dimension getPreferredSize(HVisible hvisible);
public abstract Dimension getMaximumSize(HVisible hvisible);
public abstract boolean isOpaque(HVisible hvisible);
public abstract Insets getInsets(HVisible hvisible);
}
```

El método principal a implementar será *showLook*, que recibe el *Graphics (*Componente gráfico que permite el pintado de los componentes*)*. Algo a tener en cuenta, viendo la definición del método, es que necesita recibir un *org.havi.ui.HVisible.* Esto supone que todos nuestros componentes que usen el *org.havi.ui.HLook*, deben implementar esta interfaz.

Veamos un ejemplo:

Vamos a usar el componente de texto que hemos creado en el punto anterior, y hacer que use dos tipos implementaciones diferentes de HLook.

```
package es.gpm.mhp;
import java.awt.Dimension;
import java.awt.FontMetrics;
import java.awt.Graphics;
import java.awt.Insets;
import org.havi.ui.HChangeData;
import org.havi.ui.HLook;
import org.havi.ui.HVisible;
public class LabelRectHLook implements HLook{
private static Insets _insets = new Insets(2, 2, 2, 2);
public Insets getInsets(HVisible hvisible) {
```

ISBN: 978-84-9012-858-9

Colección Aquilafuente: AQ-238
Ediciones Universidad de Salamanca - cc by nc dc

```
return _insets;
}
public Dimension getMaximumSize(HVisible hvisible) {
return hvisible.getMaximumSize();
}
public Dimension getMinimumSize(HVisible hvisible) {
return hvisible.getMaximumSize();
}
public Dimension getPreferredSize(HVisible hvisible) {
return hvisible.getPreferredSize();
}
public boolean isOpaque(HVisible hvisible) {
return hvisible.isOpaque();
}

public void showLook(Graphics g, HVisible hvisible, int i) {
Dimension d = hvisible.getSize();
g.setColor(hvisible.getBackground());
g.fillRect(0, 0, d.width-1, d.height-1);
FontMetrics fontMetrics = g.getFontMetrics();
g.setColor(hvisible.getForeground());
g.drawString(((Label)hvisible).getText(), (d.width-1- fontMetrics.stringWidth(((Label)hvisible).getText()))/2, (d.height-1)/2+font-
Metrics.getMaxAscent()/2);
}
public void widgetChanged(HVisible hvisible, HChangeData[] ahchangedata) {
hvisible.repaint();
}
}
```

Como se puede ver ahora para nosotros, el método *showLook* será el equivalente al método *paint* del *HComponent.* Algo importante es que este método sólo se use para pintar, eliminando de él toda lógica. Esta parte es sólo la vista del componente.

Ahora veamos cómo cambia nuestro componente de texto usando el HLook.

```
package es.gpm.mhp;
import org.havi.ui.HInvalidLookException;
import org.havi.ui.HVisible;
public class Label extends HVisible {
private String _text;
protected static HLook _defaultHLook = new LabelRectHLook();
public Label() {
try {
setLook(_defaultHLook);
} catch (HInvalidLookException e) {
}
}
public Label(String text) {
this();
_text = text;
}
public String getText() {
return _text;
}
public void setText(String text) {
_text = text;
}
public static void setDefaultHLook(HLook defaultHLook) {
_defaultHLook = defaultHLook;
}
}
```

Si nos fijamos bien se verá que el cambio fundamental es que en vez de extender del *HComponent* extendemos de *HVisible.* Y que no tenemos que implementar por tanto el método *paint*.

Supongamos que ahora queremos que nuestro componente tenga los bordes redondeados. Lo único que debemos hacer es crear una nueva implementación de un *HLook.* Como por ejemplo:

```
package es.gpm.mhp;
```

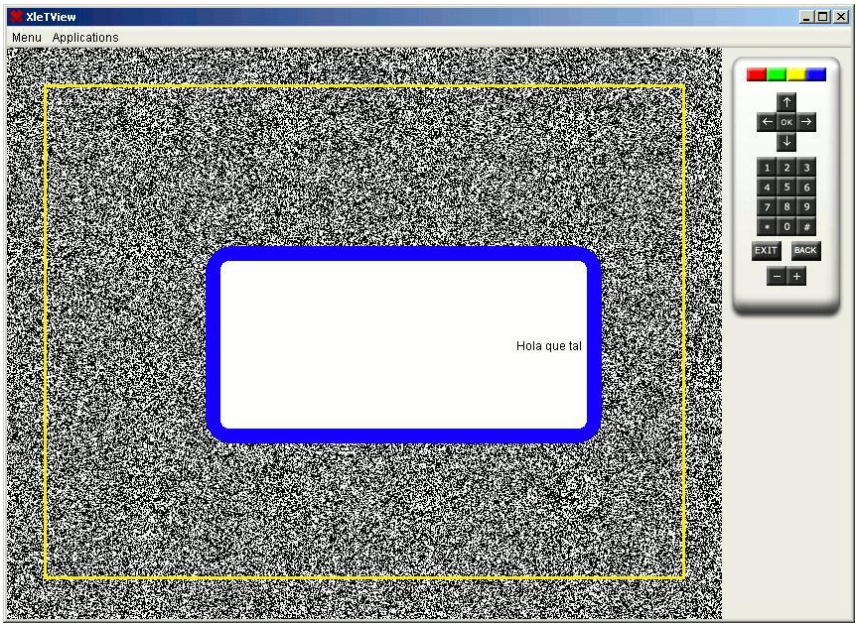

ISBN: 978-84-9012-858-9
Colección Aquilafuente: AQ-238
Ediciones Universidad de Salamanca - cc by nc dc

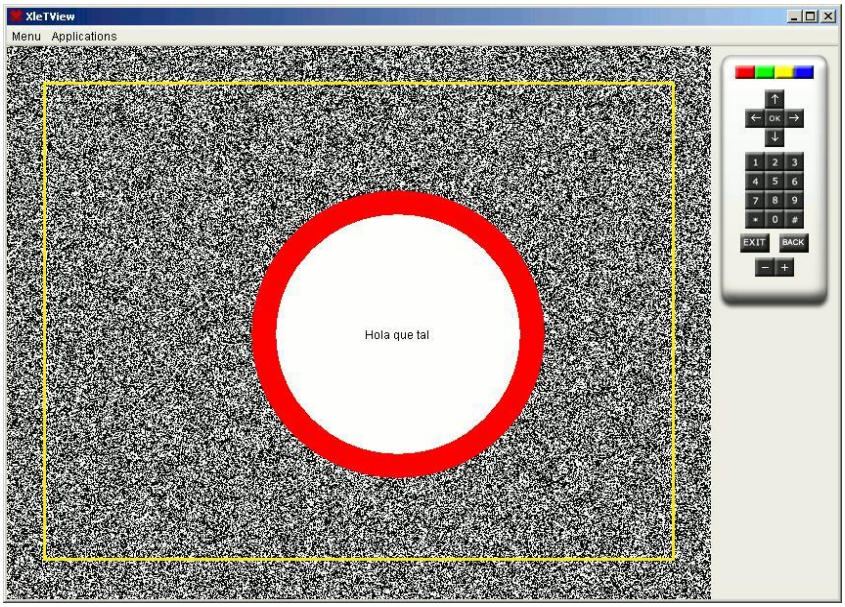
```
import java.awt.Dimension;
import java.awt.FontMetrics;
import java.awt.Graphics;
import java.awt.Insets;
import org.havi.ui.HChangeData;
import org.havi.ui.HLook;
import org.havi.ui.HVisible;
public class LabelMoothHLook implements HLook{
private static Insets _insets = new Insets(2, 2, 2, 2);
public Insets getInsets(HVisible hvisible) {
return _insets;
}
public Dimension getMaximumSize(HVisible hvisible) {
return hvisible.getMaximumSize();
}
public Dimension getMinimumSize(HVisible hvisible) {
return hvisible.getMaximumSize();
}
public Dimension getPreferredSize(HVisible hvisible) {
return hvisible.getPreferredSize();
}
public boolean isOpaque(HVisible hvisible) {
return hvisible.isOpaque();
}
public void showLook(Graphics g, HVisible hvisible, int i) {
Dimension d = hvisible.getSize();
g.setColor(hvisible.getBackground());
g.fillRoundRect(0, 0, d.width-1, d.height-1,10,10);
FontMetrics fontMetrics = g.getFontMetrics();
g.setColor(hvisible.getForeground());
g.drawString(((Label)hvisible).getText(), (d.width-1-fontMetrics.stringWidth(((Label)hvisible).getText()))/2, (d.height-1)/2+font-
Metrics.getMaxAscent()/2);
}
public void widgetChanged(HVisible hvisible, HChangeData[] ahchangedata) {
hvisible.repaint();
}
}
```

Se ve como ahora usamos el método *fillRoundRect* en vez del *fillRect*.
A continuación podemos ver dos pantallas que muestran a un mismo componente, pero con diferentes HLook:

ISBN: 978-84-9012-858-9

Colección Aquilafuente: AQ-238
Ediciones Universidad de Salamanca - cc by nc dc





Si ahora quisiéramos usar este nuevo *HLook*, sólo deberíamos hacer lo siguiente:

```
Label.setDefaultHLook(new LabelMoothHLook());
Label text = new Label("Hola");
```

Como se puede ver, si implementamos nuestros componentes de esta forma podremos separar de una forma fácil y sencilla lo que es la lógica de nuestro componente con la forma en la que queremos que se pinte en la pantalla, permitiendo una gran reutilización de componentes y una gran facilidad para cambiar la apariencia de nuestra aplicación.

ISBN: 978-84-9012-858-9
Colección Aquilafuente: AQ-238
Ediciones Universidad de Salamanca - cc by nc dc

7 Usos Avanzados

Crear una Lanzadera

Esta es una aplicación auto-ejecutable (en cuento se recuperan todos los datos desde el carrusel se inicia automáticamente), que se usa para avisar de que hay aplicaciones disponibles, y poder así lanzarlas. Cuando accedes a un canal, la aplicación se ejecutará y mostrará un menú que indicará las aplicaciones a lanzar y cómo hacerlo.

Para poder acceder a la lista de las aplicaciones disponibles en el canal, usaremos la clase AppsDatabase, la cual tiene un método estático (getAppsDatabase()) que nos da una referencia a una instancia de sí misma. En la clase AppsDatabase podemos dar de alta también un listener que recibirá notificaciones de los cambios que pueda haber en esta base de datos con referencia a las aplicaciones disponibles. Para ello, sólo hay que implementar la interfaz java.util.EventListener cuya definición podemos ver a continuación:

```
package org.dvb.application;
import java.util.EventListener;
public interface AppsDatabaseEventListener extends EventListener{
public void newDatabase(AppsDatabaseEvent evt);
public void entryAdded(AppsDatabaseEvent evt);
public void entryRemoved(AppsDatabaseEvent evt);
public void entryChanged(AppsDatabaseEvent evt);
}
```

AppsDatabase tiene una enumeración de los atributos de cada una de las aplicaciones, lo que nos permitirá acceder a su nombre, ID y otros datos relevantes. Con la información obtenida es posible acceder a una referencia de la aplicación a través del método de AppsDatabase,

```
(AppProxy) dataBase.getAppProxy(info.getIdentifier());
```

AppProxy es una interfaz que es un proxy a la aplicación, con todos los métodos necesarios para su manejo como se puede ver en la definición de abajo:

```
package org.dvb.application ;
public interface AppProxy {
public static final int STARTED = 0;
public static final int DESTROYED = 1;
public static final int NOT_LOADED = 2;
public static final int PAUSED = 3;
public int getState () ;
public void start ();
public void start (String args[]);
public void stop(boolean forced);
public void pause();
public void resume () ;
public void addAppStateChangeEventListener (AppStateChangeEventListener listener) ;
public void removeAppStateChangeEventListener (AppStateChangeEventListener listener) ;
}
```

Si queremos saber si nuestra aplicación ha cambiado de estado podemos dar de alta el listener AppStateChangeEventListener, donde se informará si ha habido un cambio en el estado, pasando por ejemplo de STARTED a DESTROYED.

```
package org.dvb.application;
import java.util.EventListener;
public interface AppStateChangeEventListener extends EventListener{
public void stateChange(AppStateChangeEvent evt);
}
```

Los métodos más importantes son load, init, start, los cuales cargarán la aplicación en memoria, la inicializarán y la arrancarán respectivamente.

Veamos un ejemplo de esta clase:

```
import java.awt.Graphics;
import java.awt.event.KeyEvent;
import java.util.Enumeration;
import java.util.Vector;
```

ISBN: 978-84-9012-858-9

Colección Aquilafuente: AQ-238
Ediciones Universidad de Salamanca - cc by nc dc

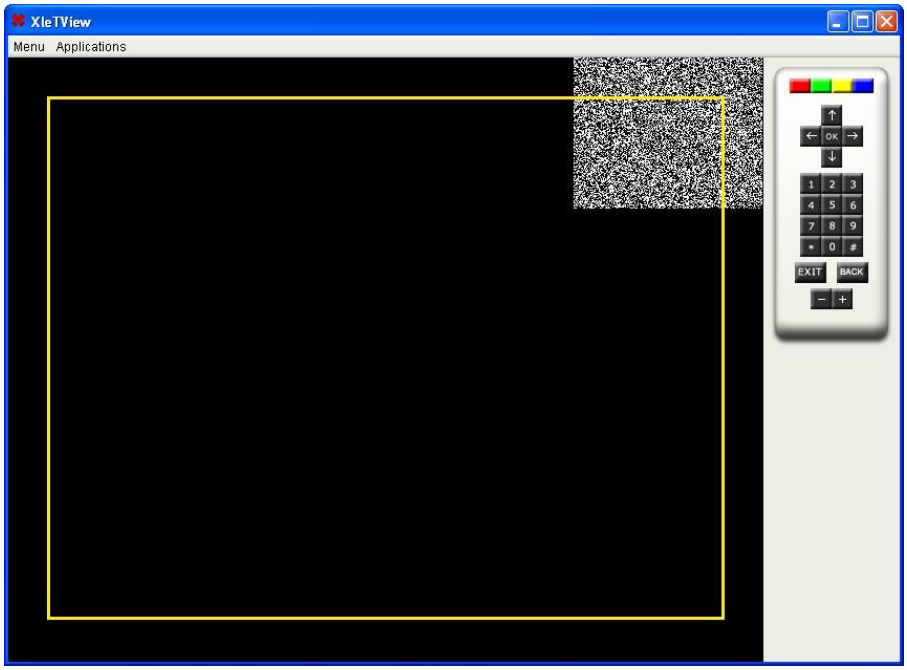
```
import org.dvb.application.AppAttributes;
import org.dvb.application.AppProxy;
import org.dvb.application.AppStateChangeEvent;
import org.dvb.application.AppStateChangeEventListener;
import org.dvb.application.AppsDatabase;
import org.dvb.application.AppsDatabaseEvent;
import org.dvb.application.AppsDatabaseEventListener;
import org.dvb.application.CurrentServiceFilter;
import org.dvb.application.DVBJProxy;
import org.dvb.ui.DVBColor;
import org.havi.ui.event.HKeyEvent;
public class Lanzadera extends HContainer implements AppsDatabaseEventListener,
AppStateChangeEventListener {
AppsDatabase dataBase;
Vector nombreApps = new Vector();
Vector proxyApps = new Vector();
String message = ".";
public Lanzadera() {
}
public void paint(Graphics g) {
super.paint(g);
g.setColor(DVBColor.blue);
g.fillRect(90, 35, 500, 20);
g.setColor(DVBColor.cyan);
g.drawString(message, 100, 75 - 25);
for (int i = 0; i < nombreApps.size(); i++) {
g.setColor(DVBColor.blue);
g.fillRect(90, 60, 500, 20 + i * 25);
g.setColor(DVBColor.cyan);
g.drawString(i + ". " + (String) nombreApps.elementAt(i), 100,
75 + i * 25);
}
}
public void keyPressed(KeyEvent e) {
if (e.getKeyCode() == HKeyEvent.VK_COLORED_KEY_0) {
//Aquí muestr la lanzadera
xlet.showView("org.vicomtech.mhp.Indice");
return;
} else if (e.getKeyCode() == HKeyEvent.VK_COLORED_KEY_1) {
//Ocultamos la lanzadera
xlet.hideView();
//Ejecutamos la aplicación (en este caso siempre la //primera)
((DVBJProxy) proxyApps.elementAt(0)).load();
((DVBJProxy) proxyApps.elementAt(0)).init();
((DVBJProxy) proxyApps.elementAt(0)).start();
}
}
public void setTheXlet(TheXlet xlet) {
super.setTheXlet(xlet);
dataBase = AppsDatabase.getAppsDatabase();
if(dataBase!=null){
dataBase.addListener(this);
}
parseDataBase();
repaint();
}
public void parseDataBase() {
if (dataBase != null) {
Enumeration attributes = dataBase
.getAppAttributes(new CurrentServiceFilter());
if (attributes != null) {
AppAttributes info;
AppProxy proxy;
while (attributes.hasMoreElements()) {
info = (AppAttributes) attributes.nextElement();
this.nombreApps.add(info.getName());
```


ISBN: 978-84-9012-858-9
Colección Aquilafuente: AQ-238
Ediciones Universidad de Salamanca - cc by nc dc

```
proxy = (AppProxy) dataBase.getAppProxy(info
.getIdentifier());
proxy.addAppStateChangeEventListener(this);
proxyApps.add(proxy);
}
}
}
}
public void entryAdded(AppsDatabaseEvent evt) {
}
public void entryChanged(AppsDatabaseEvent evt) {
}
public void entryRemoved(AppsDatabaseEvent evt) {
}
public void newDatabase(AppsDatabaseEvent evt) {
}
public void stateChange(AppStateChangeEvent evt) {
message = "App changed state";
}
}
```

7.1 Redimensionar la capa de Vídeo

En toda aplicación MHP se definen tres capas en el siguiente orden de la más inferior hasta la más superior o cercana al usuario: background, vídeo y aplicación. Una aplicación puede modificar el tamaño de la capa de vídeo.
Podemos ver una redimensión en la siguiente imagen:



Para ello:

```
// Factoría de contexto de servicio
ServiceContextFactory scf = null;
// Contexto servicio
ServiceContext sc = null;
try {
```

ISBN: 978-84-9012-858-9

Colección Aquilafuente: AQ-238

```
// Obtención de ServiceContextFactory
scf = ServiceContextFactory.getInstance();
// Obtención del contexto
XletContext context = MainXlet._context;
// Obtención del ServiceContext
sc = scf.getServiceContext(context);
} catch (Exception e) {
e.printStackTrace();
}
// Si el ServiceContext no es nulo
if (sc != null) {
// Manejadores de contenido
ServiceContentHandler[] sch = sc.getServiceContentHandlers();
// Player JMF
Player player = null;
if (sch.length > 0) {
player = (Player) sch[0];
}
if (player != null) {
// Obtención de AWTVideoSizeControl
AWTVideoSizeControl avsc = (AWTVideoSizeControl) player.getControl("javax.tv.media.AWTVideoSizeControl");
// Redimensión
avsc.setSize(new AWTVideoSize(new Rectangle(720, 576),
new Rectangle(200, 200)));
}
}
```

7.2 Lectura del Carrusel

No hay ninguna restricción para un servicio DVB para contener más de un carrusel. No hay incluso restricciones para que una aplicación acceda a carruseles de objetos diferentes al que pertenecen. Por tanto, si podemos acceder a otros carruseles, ¿cómo hacerlo?

Como en un sistema de ficheros de Unix, los carruseles de objetos pueden ser montados en una posición cualquiera de la jerarquía de ficheros. Antes de ver cómo hacer esto, hablemos un poco sobre la terminología. Un carrusel de objetos está a veces referenciado como un Service Domain. Para ser preciso, un Service Domain es en realidad un grupo de objetos DSM-CC relacionados. En una red de broadcast, éstos están en un carrusel de objetos y transmitidos al cliente. En una network interactiva, un cliente puede manipularlos usando el protocolo user-to-user DSM_CC. Aquí no se hablará sobre este protocolo, ya que lo que nos concierne básicamente son las broadast networks. Sin embargo, la operación básica es la misma en ambos casos.

La API MHP DSM-CC representa un Service Domain usando la clase ServiceDomain. Antes de que un Service Domain pueda ser usado, debemos adjuntarlo. Esto monta el domino de servicio a la jerarquía del sistema de ficheros, de la misma forma que el comando de Unix mount lo hace. Un Service Domain se adjunta usando el método attach(). Hay tres diferentes versiones de este método, cada uno toma diferentes parámetros.

```
public void attach(Locator l)
public void attach(Locator service, int carouselId)
public void attach(byte[] NSAPAddress)
```

Algo que se puede observar es que ninguno de los métodos anteriores permite especificar dónde montar el Service Domain. Esto asegura que el receptor puede evitar conflictos y montar el Service Domain en el punto donde mejor considere. Una aplicación puede usar el getMountPoint() para obtener un objeto que represente el punto de montaje.

Después de que un servicio ha sido montado, los ficheros son accesibles. Una aplicación puede desmontar el Service Domain usando el método detach(). Esto permitirá saber al receptor que cualquier fichero cacheado de ese domino puede ser eliminado.

ISBN: 978-84-9012-858-9
Colección Aquilafuente: AQ-238
Ediciones Universidad de Salamanca - cc by nc dc

El hecho de que una aplicación tengo montado un Service Domain no significa que este Service Domain sea siempre accesible. Si el receptor cambió la sintonía a otro que no contenga el transport stream que contenía el carrusel, entonces los ficheros puede que no sean accesibles. Si el receptor decide que nunca se podrá conectar al carrusel de nuevo, puede elegir desmontarlo.
Cuando el Service Domain no es accesible, entonces cualquier intento de acceder a un fichero fallará, como si el fichero no existiera. A nivel de API, impedir el acceso a un objeto del carrusel causara una MPEGDeliveryException. Esto podría no ser un fallo permanente, ya que futuros intentos podrían ser exitosos.

7.2.1 Ejemplo

```
// create a new ServiceDomain object to represent
// the carousel we will use
ServiceDomain carousel = new ServiceDomain();
// now create a Locator that refers to the service
// that contains our carousel
org.davic.net.Locator locator;
locator = new org.davic.net.Locator
("dvb://123.456.789");
// finally, attach the carousel to the ServiceDomain
// object (i.e. mount it) so that we can actually
// access the carousel. In this case, we don't specify
// the stream containing the carousel in the locator, so
// we pass in the carousel ID as part of the attach
// request
carousel.attach(locator, 1);
// we have to create our DSMCCObject with an absolute
// path name, which means we need to get the mount point
// for the service domain
DSMCCObject dsmccObj;
dsmccObj = new DSMCCObject(carousel.getMountPoint(),
"graphics/image1.jpg");
// now we create a FileInputStream instance to access
// our DSM-CC object. Alternatively, we could create
// a RandomAccessFile instance if we wanted random
// instead of sequential access to the file.
FileInputStream inputStream;
inputStream = new FileInputStream(dsmccObj);
// we can now use the FileInputStream just like any
// other FileInputStream
```

ISBN: 978-84-9012-858-9

Colección Aquilafuente: AQ-238
Ediciones Universidad de Salamanca - cc by nc dc

References

1. Alejandro Jiménez-Rodríguez, Luis Fernando Castillo, Manuel González (2012). Studying the mechanisms of the Somatic Marker Hypothesis in Spiking Neural Networks (SNN). ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 1, n. 2
2. Amir Hosein Keyhanipour, Behzad Moshiri (2013). Designing a Web Spam Classifier Based on Feature Fusion in the Layered Multi-Population Genetic Programming Framework. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 2, n. 3
3. André Santos, Regina Nogueira, Anália Lourenço (2012). Applying a text mining framework to the extraction of numerical parameters from scientific literature in the biotechnology domain. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 1, n. 1
4. Anna Závodská, Veronika Šramová, Anne-Maria AHO (2012). Knowledge in Value Creation Process for Increasing Competitive Advantage. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 1, n. 3
5. Casado-Vara, R., Chamoso, P., De la Prieta, F., Prieto J., & Corchado J.M. (2019). Non-linear adaptive closed-loop control system for improved efficiency in IoT-blockchain management. Information Fusion.
6. Casado-Vara, R., Novais, P., Gil, A. B., Prieto, J., & Corchado, J. M. (2019). Distributed continuous-time fault estimation control for multiple devices in IoT networks. IEEE Access.
7. Casado-Vara, R., Vale, Z., Prieto, J., & Corchado, J. (2018). Fault-tolerant temperature control algorithm for IoT networks in smart buildings. Energies, 11(12), 3430.
8. Casado-Vara, R., Prieto-Castrillo, F., & Corchado, J. M. (2018). A game theory approach for cooperative control to improve data quality and false data detection in WSN. International Journal of Robust and Nonlinear Control, 28(16), 5087-5102.
9. Chamoso, P., González-Briones, A., Rivas, A., De La Prieta, F., & Corchado J.M. (2019). Social computing in currency exchange. Knowledge and Information Systems.
10. Chamoso, P., González-Briones, A., Rivas, A., De La Prieta, F., & Corchado, J. M. (2019). Social computing in currency exchange. Knowledge and Information Systems, 1-21.
11. Chamoso, P., González-Briones, A., Rodríguez, S., & Corchado, J. M. (2018). Tendencies of technologies and platforms in smart cities: A state-of-the-art review. Wireless Communications and Mobile Computing, 2018.
12. Chamoso, P., Rivas, A., Martín-Limorti, J. J., & Rodríguez, S. (2018). A Hash Based Image Matching Algorithm for Social Networks. In Advances in Intelligent Systems and Computing (Vol. 619, pp. 183–190). https://doi.org/10.1007/978-3-319-61578-3_18
13. Chamoso, P., Rodríguez, S., de la Prieta, F., & Bajo, J. (2018). Classification of retinal vessels using a collaborative agent-based architecture. AI Communications, (Preprint), 1-18.
14. Di Mascio, T., Vittorini, P., Gennari, R., Melonio, A., De La Prieta, F., & Alrifai, M. (2012, July). The Learners' User Classes in the TERENCE Adaptive Learning System. In 2012 IEEE 12th International Conference on Advanced Learning Technologies (pp. 572-576). IEEE.
15. Felicitas Mokom, Ziad Kobti (2013). Interventions via Social Influence for Emergent Suboptimal Restraint Use. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 2, n. 2
16. Gonzalez-Briones, A., Chamoso, P., De La Prieta, F., Demazeau, Y., & Corchado, J. M. (2018). Agreement Technologies for Energy Optimization at Home. Sensors (Basel), 18(5), 1633-1633. doi:10.3390/s18051633
17. González-Briones, A., Chamoso, P., Yoe, H., & Corchado, J. M. (2018). GreenVMAS: virtual organization-based platform for heating greenhouses using waste energy from power plants. Sensors, 18(3), 861.
18. Gonzalez-Briones, A., Prieto, J., De La Prieta, F., Herrera-Viedma, E., & Corchado, J. M. (2018). Energy Optimization Using a Case-Based Reasoning Strategy. Sensors (Basel), 18(3), 865-865. doi:10.3390/s18030865
19. K. S. Jasmine, Gavani Prathviraj S., P Ijantakar Rajashekar, K. A. Sumithra Devi (2013). Inference in Belief Network using Logic Sampling and Likelihood Weighing algorithms. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 2, n. 3
20. Manuel Rodrigues, Sérgio Gonçalves, Florentino Fdez-Riverola (2012). E-learning Platforms and E-learning Students: Building the Bridge to Success. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 1, n. 2
21. Muñoz, M., Rodríguez, M., Rodríguez, M. E., & Rodríguez, S. (2012). Genetic evaluation of the class III dentofacial in rural and urban Spanish population by AI techniques. Advances in Intelligent and Soft Computing (Vol. 151 AISC). https://doi.org/10.1007/978-3-642-28765-7_49

ISBN: 978-84-9012-858-9
Colección Aquilafuente: AQ-238
Ediciones Universidad de Salamanca - cc by nc dc

22. Nadia Alam, Munira Sultana, M.S. Alam, M. A. Al-Mamun, M. A. Hossain (2013). Optimal Intermittent Dose Schedules for Chemotherapy Using Genetic Algorithm. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 2, n. 2
23. Naoufel Khayati, Wided Lejouad-Chaari (2013). A Distributed and Collaborative Intelligent System for Medical Diagnosis. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 2, n. 2
24. Nicholas Beliz, José Carlos Rangel, Chi Shun Hong (2012). Detecting DoS Attack in Web Services by Using an Adaptive Multiagent Solution. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 1, n. 2
25. Rodríguez, S., Tapia, D. I., Sanz, E., Zato, C., De La Prieta, F., & Gil, O. (2010). Cloud computing integrated into service-oriented multi-agent architecture. IFIP Advances in Information and Communication Technology (Vol. 322 AICT). https://doi.org/10.1007/978-3-642-14341-0_29
26. Saadi Bin Ahmad Kamaruddin, Nor Azura Md Ghanib, Choong-Yeun Liong, Abdul Aziz Jemain (2012). Firearm Classification using Neural Networks on Ring of Firing Pin Impression Images. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 1, n. 3
27. Sittón-Candanedo, I., Alonso, R. S., Corchado, J. M., Rodríguez-González, S., & Casado-Vara, R. (2019). A review of edge computing reference architectures and a new global edge proposal. Future Generation Computer Systems, 99, 278-294.
28. Sérgio Matos, Hugo Araújo, José Luís Oliveira (2013). Biomedical Literature Exploration through Latent Semantics. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 2, n. 2
29. Sittón, I., & Rodríguez, S. (2017). Pattern Extraction for the Design of Predictive Models in Industry 4.0. In International Conference on Practical Applications of Agents and Multi-Agent Systems (pp. 258–261).
30. Sumit Goyal, Gyanendra Kumar Goyal (2013). Machine Learning ANN Models for Predicting Sensory Quality of Roasted Coffee Flavoured Sterilized Drink. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 2, n. 3
31. Vasileios Efthymiou, Maria Koutraki, Grigoris Antoniou (2012). Real-Time Activity Recognition and Assistance in Smart Classrooms. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 1, n. 1
32. Vincenza Cofini, Fernando De La Prieta, Tania Di Mascio, Rosella Gennari, Pierpaolo Vittorini (2012). Design Smart Games with requirements, generate them with a Click, and revise them with a GUIs. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 1, n. 3
33. Yi Ying Leong, Chuii Khim Chong, Lian En Chai, Safaai Deris, Rosli Illias, Sigeru Omatu, Mohd Saberi Mohamad (2012). Simulation of Fermentation Pathway Using Bees Algorithm. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 1, n. 2

ISBN: 978-84-9012-858-9

Colección Aquilafuente: AQ-238
Ediciones Universidad de Salamanca - cc by nc dc