ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN
UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Proyecto Fin de Carrera

SIMUTEMP: Simulador de un Conmutador Temporal

AUTOR: Francisco Javier Martínez Mercader
DIRECTOR: Francesc Burrull i Mestres

Julio / 2004

Autor	Francisco Javier Martínez Mercader
E-mail del Autor	fjmmercader@hotmail.com
Director	Francesc Burrull i Mestres
E-mail del Director	francesc.burrull@upct.es
Título del PFC	SIMUTEMP: Simulador de un Conmutador Temporal
Descriptores	Simulador, Conmutación Temporal

Resumen

Un elemento esencial de las redes de conmutación de circuitos son los conmutadores propiamente dichos. Como complemento a su estudio se pueden utilizar simuladores didácticos, que permiten una mejor comprensión de los conmutadores, así como la obtención de datos estadísticos.

En este proyecto se ha realizado un simulador de un conmutador temporal, herramienta que servirá para la docencia en la Escuela Técnica Superior de Ingeniería de Telecomunicación (E.T.S.I.T.) de la Universidad Politécnica de Cartagena (U.P.C.T.), disponiendo además del código fuente de la herramienta.

Titulación	Ingeniero Técnico de Telecomunicación
Intensificación	Telemática
Departamento	Departamento de Tecnologías de la Información y las Comunicaciones
Fecha de Presentación	Julio - 2004

Índice General

Capítulo 1
Introducción

Un elemento esencial en las redes de conmutación de circuitos son los conmutadores propiamente dichos. Como complemento a su estudio se pueden utilizar simuladores didácticos, que permiten una mejor comprensión de los conmutadores, así como la obtención de datos estadísticos.

En este proyecto se ha relizado un simulador de un conmutador temporal, herramienta que servirá para la docencia en la Escuela Técnica Superior de Ingeniería de Telecomunicación (E.T.S.I.T.) de la Universidad Politécnica de Cartagena (U.P.C.T.), disponiendo además del código fuente de la herramienta.

Capítulo 2
Objetivos

Nuestro objetivo es implementar una aplicación didáctica consistente en un simulador de conmutación en el tiempo, partiendo como base el simulador ya realizado *TIMSWIT*. Con ello conseguiremos una aplicación de la que se dispondrá el código fuente y las siguientes mejoras respecto al *timswit*:

- Introducir mejoras en el control del algoritmo de transmisión, como retroceder en los pasos dados, es decir, la marcha atrás en todos los posibles modos de transmisión (ciclo, slot y trama).

- Facilitar la comprensión del programa mediante una ayuda que el alumno podrá usar cuando la necesite, viendo así la utilidad y la función de todos los componentes del programa.

- En la opción de multiplexación por división en el tiempo, además de la opción conocida de 4 canales, se incluyen 3 apartados más: simulación con 8 canales, con 16, y con 32, adaptando así al simulador aún más a la realidad de lo que es TDM.

- Se mejora la portabilidad del programa, encontrándose éste en un archivo tipo *"*.jar"*, donde se podrá ejecutar en cualquier sistema operativo, como Windows, LINUX, etc.

- La interfaz gráfica se mejora considerablemente al pasar de un programa en MS-DOS a una ventana tipo *JFrame*, donde el entorno visual es mucho más agradable, permitiendo la interacción del alumno con el ratón, y facilitando la comprensión del algoritmo mediante elementos gráficos y ventanas de ayuda emergentes.

- El software será propio de la Universidad Politécnica de Cartagena, sin tener que pagar ningún tipo de licencia o tener problemas con ésta, y al ser software de la propia Universidad, estará en castellano, facilitando así su comprensión al alumno.

- El código se dejará abierto para posibles mejoras futuras del programa.

Capítulo 3
Análisis y diseño

3.1. Ingeniería Inversa

Nuestro programa, a pesar de partir de cero en cuanto a código se refiere, el comportamiento y los algoritmos los sacamos partiendo de otro programa ya hecho, por lo que para su implementación y diseño recurriremos a la ingeniería inversa.

La ingeniería inversa consiste en desmontar un objeto para ver cómo funciona y, de ese modo, duplicar o mejorar el mismo. En el ámbito informático, existen 2 maneras de llevarla a cabo:

1. Empleando métodos de “caja negra”, que consiste en observar desde fuera el comportamiento de un programa al que se somete a una serie de casos de uso. Esto es factible en programas pequeños.

2. Descompilando un programa de modo que se obtenga el código fuente, legible por un programador, del mismo a partir del código que ejecutamos en nuestro ordenador. Es la única manera con programas grandes, como sistemas operativos.

En el ámbito del software, esta práctica está prohibida, tanto por las licencias con las que se vende como por leyes (aunque en este último caso la primera opción se permite).

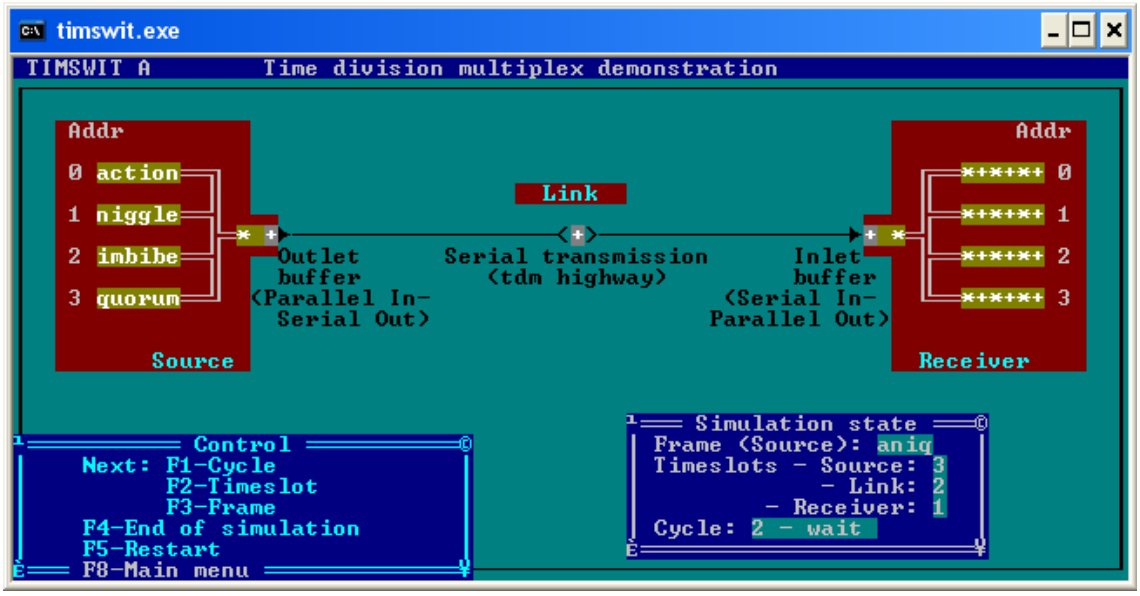
Es por esta última razón y por la relativa simplicidad del programa por lo que se ha decidido que se usará el método de “caja negra”, viendo cómo actúa el algoritmo y plasmándolo en el código empezando desde cero.

Ahora vamos a pasar a comentar las características del programa de partida, el *timswit*, pero muy por encima, para pasar a profundizar más posteriormente en nuestro simulador:

En el simulador hay 3 opciones, Opción A y Opción B que serán las que tendremos en cuenta a la hora de la implementación, y una tercera Opción C que no se ha incluido en este proyecto, que es una mezcla de conmutación espacial y temporal usando TDM, que se dejará para posibles futuros proyectos.

3.2. Opción A

La Opción A es la demostración de una multiplexación por división en el tiempo de 4 canales, donde arriba encontramos el emisor, el receptor, y la línea de transmisión. En la parte inferior vemos el mecanismo de control de la ejecución, que se controla mediante F1, F2 y F3. Para ir al punto final de la simulación pulsamos F4, y para reiniciar le damos a F5. El F8 es para volver al menú principal:



Como hemos dicho antes, las teclas F1, F2, y F3 controlan la ejecución paso a paso del programa, pero antes de explicar qué hace cada una, debemos saber los 3 diferentes ciclos en los que se desarrolla la transmisión en un slot completo de tiempo:

1. **“0 – Output”**

   Es el momento en el que una letra de la fuente pasa al buffer de salida de la misma, y en la línea se desplazan las letras, donde el buffer de entrada pasa a estar ocupado por la letra que estaba en la línea. Está representado como “0 – Output”.

2. **“1 – Input”**

   La letra que estaba en el buffer de entrada del receptor pasa a formar parte del canal que le corresponde en el mismo receptor, mientras que en la fuente desaparece la letra que queda almacenada en el buffer de salida. Este ciclo se representa por “1 – Input”.

3. **“2 – Wait”**

   Es un tiempo de espera necesario para la correcta simulación, dado que en el canal existe un retardo temporal que es el que simula este ciclo (tiempo de transmisión). Se representa mediante “2 – Wait”.

Dicho ya en qué consiste cada uno de los 3 ciclos, podemos explicar para qué funcionan las distintas teclas:

1. **F1**

   “Cycle”, simula la ejecución de un ciclo, pulsándolo reiteradamente veremos cada uno de los 3 ciclos de los que se compone una transmisión en un slot de tiempo.

2. **F2**

   “Timeslot”, se transmite el slot completo, es decir, se produce simultáneamente la ejecución de los 3 ciclos.

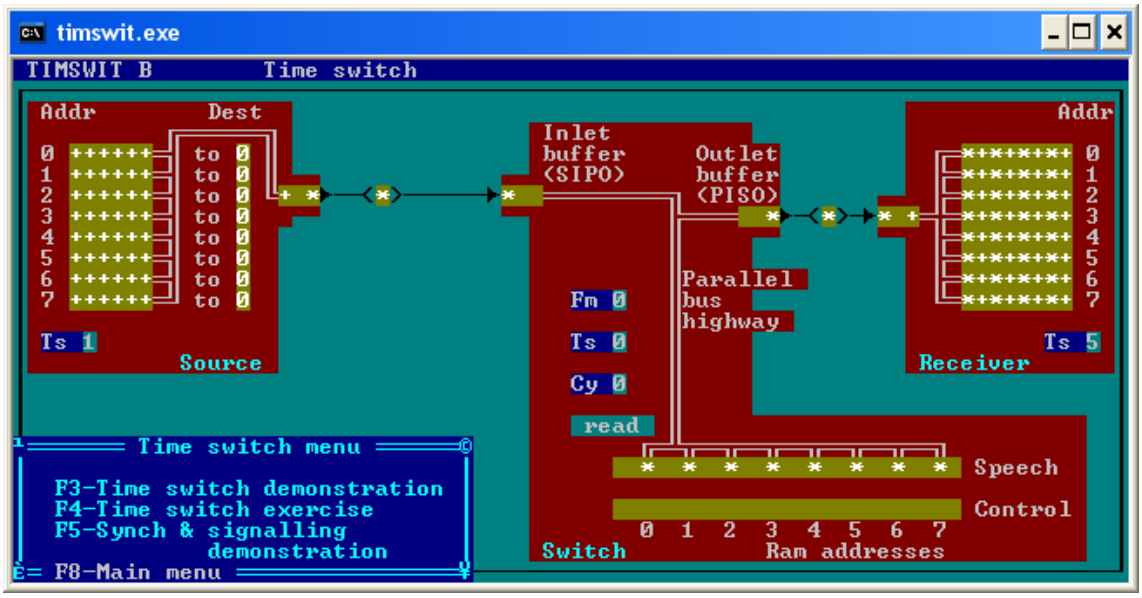
3. **F3**

   Se transmite una trama entera, es decir, 4 letras, que equivale a 4 slots y a 12 ciclos.

En cuanto a la línea, lo que inicialmente aparece también se transmite, aunque sea basura. Estos residuos están representados por los caracteres “*” y “+”, que van directamente a parar a los canales 2 y 3 (porque estos residuos estaban finalizando ya su transmisión, es como si se hubiera hecho previamente otra TDM).

3.3. Opción B

La opción B nos ofrece 3 posibilidades:



La opción F3 es una demostración de conmutación en el tiempo, es decir, una transmisión de emisor a receptor, ambos de 8 canales, y con un Switch encaminando las letras, guiándose éste por los destinos a donde debe ir cada canal.

La opción F4 es un ejercicio de lo anteriormente dicho, que para este proyecto no se va a implementar.

La opción F5 es una demostración de cómo funciona el sincronismo y la señalización, es decir, se dispone de un emisor con 8 canales, 1 de ellos destinado a la sincronización y otro para la señalización.

3.3.1. Time switch demonstration

Es la opción de la etapa T simple, donde las letras de la fuente serán encaminadas hacia el receptor a través del switch. Las letras se almacenan en speech, mientras que en control se decide cuándo se mandan al outlet buffer.

En la etapa T hay tres registros: Fm, Ts y Cy, que sirven para indicar cuántas tramas están pasando por la memoria de voz, qué slot de dicha trama se está almacenando en ese momento, y en qué ciclo se encuentra, respectivamente.

También nos encontramos con un módulo cuya única función es la de señalización, que está conectado de la fuente al switch.

Los ciclos ahora son read, write, wait:

1. **Read**

   Es el ciclo de lectura de la memoria de voz. La memoria de control es la que decide qué posición de la memoria de voz pasa al outlet buffer.

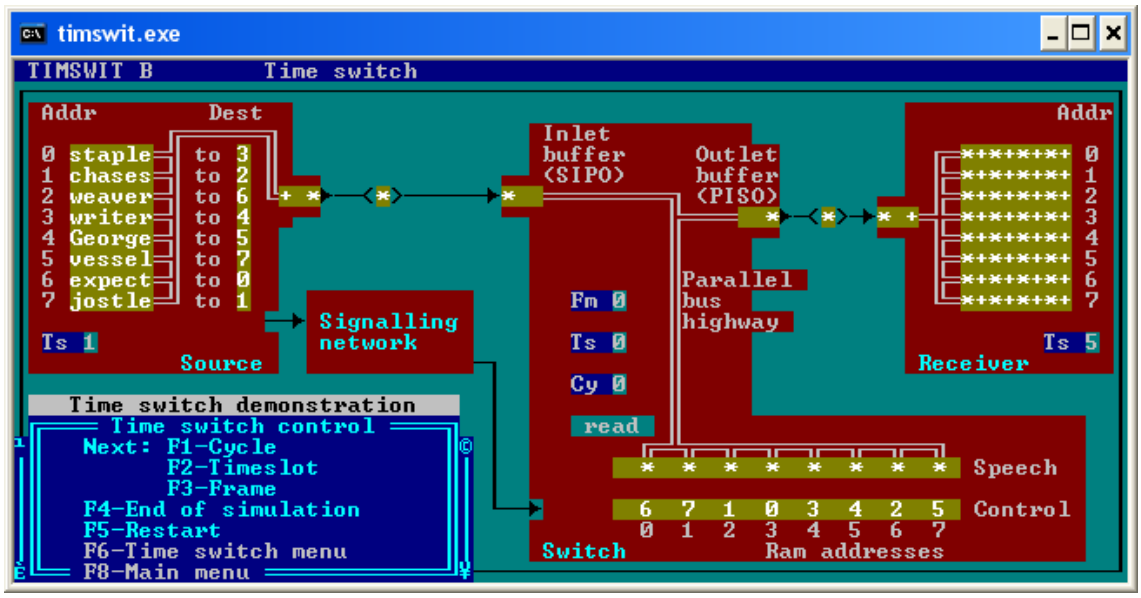
2. **Write**

   Es el ciclo de escritura en la memoria de voz. La dirección RAM que toque en ese momento es la que decide en qué posición de la memoria de voz se escribe.

3. **Wait**

   Es el ciclo de espera que hay que realizar para la correcta simulación.

La fuente y el receptor tienen un registro cada uno, el registro Ts, que indica de qué canal es la letra que se está transmitiendo (fuente) o en qué canal se está recibiendo la letra (receptor).



3.3.2. Synch & Signalling demonstration

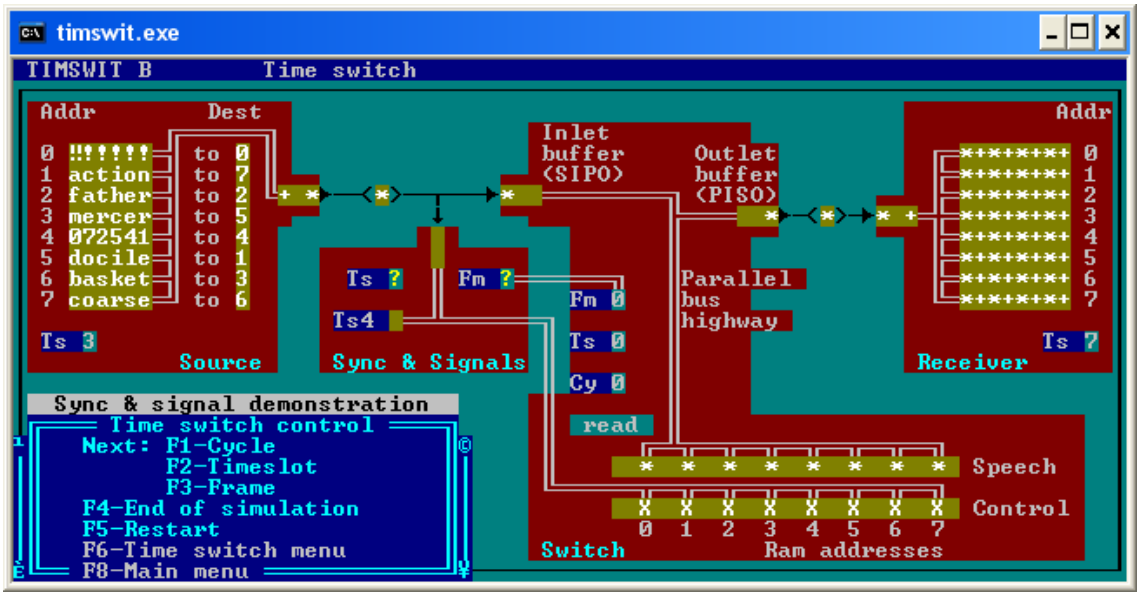
Tiene básicamente el mismo aspecto que la opción anterior, pero la funcionalidad cambia considerablemente.

Ya no hay 8 canales de datos, sino 6, los 2 restantes son para sincronismo (canal 0) y señalización (canal 4), y la memoria de control del switch inicialmente no está definida, se irá completando paso a paso con la ayuda del canal de señalización.

Otra novedad es el módulo de sincronismo y señalización, que por un lado se encarga de sincronizar los registros Fm, Ts y Cy, y por otro se encarga de rellenar la memoria de control para así poder mandar al outlet buffer el contenido de la memoria de voz. Si en una memoria de control está la letra X, quiere decir que ésta aún no ha sido definida, y lo que se enviará al outlet buffer será ruido.

El canal de señalización queda definido por los destinos que tomarán las letras que hay en la fuente, destinos que se forman aleatoriamente, con la excepción de los destinos 0 y 4, que están dirigidos, y sólo los tomarán los canales de sincronismo y señalización, respectivamente.

El sincronismo de multitrama se representa mediante “!!”, y el sincronismo de trama es “!”. El módulo de sincronismo y señalización se resetea cuando llega por primera vez “!!”, lo mismo pasa con el receptor.



Capítulo 4
Ámbito de aplicación

4.1. Elección del lenguaje de desarrollo

Se han barajado los siguientes lenguajes para desarrollar la implementación del programa:

C

El lenguaje por excelencia de programación de sistemas. Es un lenguaje de nivel medio (no se puede decir que sea de alto nivel, por incoporar muchos elementos propios del ensamblador), tremendamente ligado a UNIX (aunque es portable y hay compiladores para casi cualquier sistema operativo). Casi podríamos considerarlo como un ensamblador estructurado y portable. Difícil de aprender (no es recomendable como primer lenguaje, como sí lo podría ser *Pascal*), aunque tremendamente flexible. Bastante dado a errores, sobre todo entre programadores novatos.

C++

Evolución sobre el C. C++ sí se puede considerar de alto nivel. Es orientado a objetos, relativamente difícil de aprender (muchas características, muy complicado), pero combina la potencia y flexibilidad de C con orientación a objetos. Bastante utilizado. Dado a errores, aunque no tanto como C.

Java

Un lenguaje "de moda", parecido al C (mayormente por la sintaxis). Orientado a objetos (Java te obliga; C++ sólo te da la posibilidad), mucho menos flexible que C++, pensado para hacer aplicaciones interactivas más que controladores de dispositivos y sistemas operativos. Mucha aceptación popular, muchos recursos, y posibilidad de incluir programas en Java en páginas HTML (los llamados "applets").

Es un lenguaje multiplataforma, los programas Java se pueden ejecutar en cualquier plataforma soportada (Windows, UNIX, Mac, etc.), además es un lenguaje compilado e interpretado, ya que el compilador produce un código intermedio independiente del sistema llamado *bytecode*. Se necesita instalar en el ordenador la JVM (Java Virtual Machine), que es el intérprete que convierte el bytecode en código máquina. *Sun Microsystems* distribuye de forma gratuita el producto base, el llamado JDK (Java Depelopment Kit), también llamado J2SE (Java 2 Standard Edition), y se puede encontrar en la siguiente dirección : http://java.sun.com/

Elección final

Finalemente se ha elegido el lenguaje **Java** para la implementación del programa por las siguientes razones:

1. Se ha elegido en detrimento de C al ser un lenguaje orientado a objetos y por su portabilidad.

2. Frente a C++ tiene la ventaja de tener más portabilidad, se pueden usar las clases compiladas en cualquier plataforma (Windows, UNIX, etc.).

3. Una razón muy importante fue el entorno gráfico resultante, ya que Java proporciona elementos gráficos muy atractivos, sobre todo usando los componentes *swing*.

4.2. Elección del entorno de desarrollo

Para el lenguaje Java hay múltiples entornos de desarrollo, entre los que se han barajado:

JBuilder

Dirección: (Borland) http://www.borland.com/jbuilder/
Versión actual: 6.0
Plataformas: Windows, Linux, Solaris.
Licencia: La versión de evaluación, la Personal, es gratis, las avanzadas, Profesional y Enterprise son de pago.

JBuilder Foundation está diseñado para desarrolladores Java que quieran una alta productividad IDE (Entorno de Desarrollo Integrado) para crear más facilmente aplicaciones multiplataforma para Linux, Solaris y Windows.

JBuilder Foundation permite desarrollar rápidamente, compilar, ejecutar, y encontrar errores, usando las aplicaciones visuales de JBuilder o con métodos tradicionales de código.

Adicionalmente, JBuilder permite que los usuarios retoquen a su gusto y extiendan el entorno según sus necesidades de desarrollo usando Open Tools API, el cual facilita la integración de otros componentes adicionales.

Tiene la gran utilidad de visualizar los diagramas UML, además de poder desarrollar aplicaciones web con JSP y servlets.

Kawa

Versión actual: 5.0
Plataforma: Windows
Licencia: Como es habitual, versiones Profesional y Enterprise. Disponen de versión de evaluación.

NetBeans

Versión actual: 3.6
Plataforma: Todas con JVM
Licencia: Opensource

Dirección: http://www.netbeans.org/

NetBeans es una aplicación *open source* (el código del entorno está abierto a posibles modificaciones) de desarrollo escrita en java, esto quiere decir que se puede modificar el entorno de acuerdo a ciertos parámetros de licencia. Soporta, aparte de java, otros lenguajes, como C++.

Algunas de sus funciones y características son:

- Completado de código
- Soporte para escritura de servlets
- Ayudas con el código
- Ayuda on-line

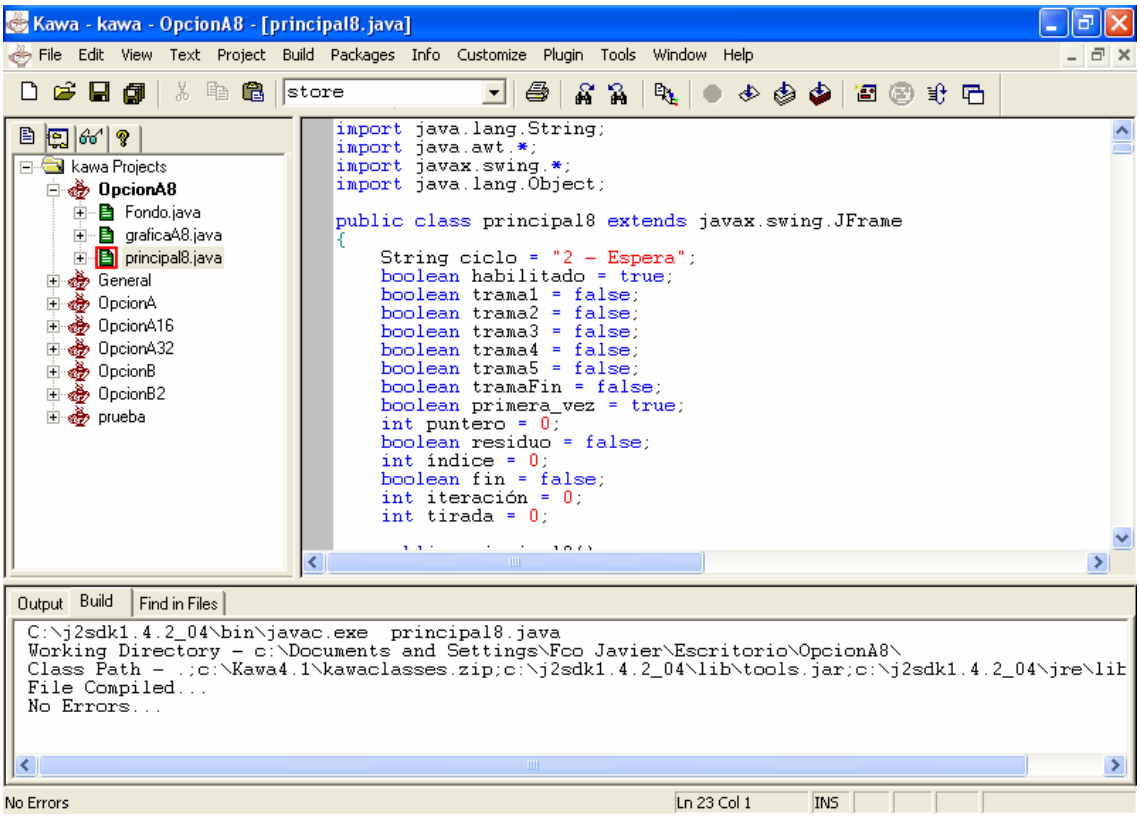
Es un entorno muy cómodo en cuanto a interfaz gráfica se refiere, ya que nos ahorra poner código tan sólo con arrastrar los componentes gráficos que soporta java y que están representados mediante iconos. Al añadir estos componentes, en la aplicación se genera el código correspondiente a dicha inclusión del elemento gráfico. Está recomendado por *Sun Microsystems*, aunque una de sus desventajas es que ocupa muchos recursos de memoria.

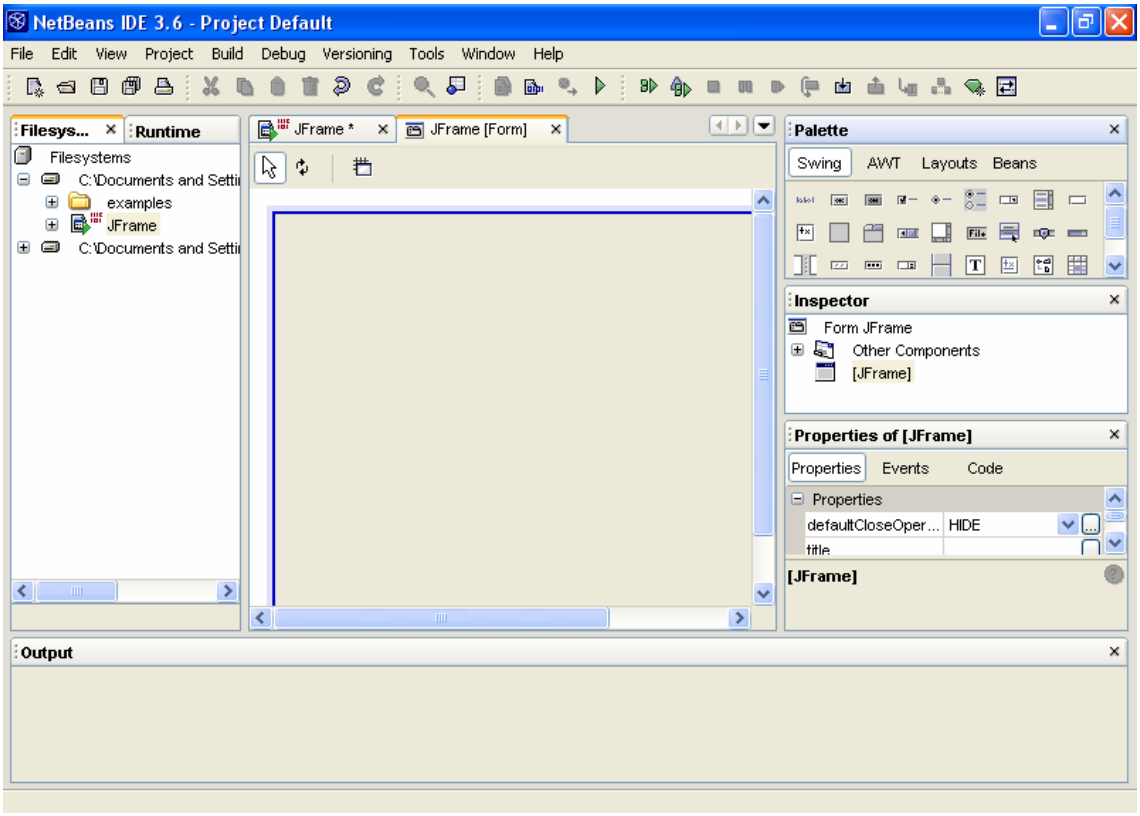
Elección final

El entorno principal elegido es el Kawa. Por su facilidad de uso y potencia, es el ideal para implementar los algoritmos, además, al ser un entorno de ventanas pero no muy sobrecargado, la programación se hace llevadera.

Para diseñar la interfaz gráfica, se ha decidido por el NetBeans, también por su gran facilidad de uso, aunque en este caso, el código que se genera al añadir componentes gráficos se ha copiado en el NetBeans y posteriormente pegado en el Kawa, ya que programar todo desde el NetBeans conllevaría mucho tiempo, por lo dicho anteriormente: consume muchos recursos de memoria, mientras que el Kawa es mucho más rápido y más accesible.

Finalmente, se ha escogido el JBuilder única y exclusivamente para realizar los diagramas UML de las clases que componen el programa.


Kawa - kawa - OpcionA8 - [principal8.java]
File Edit View Text Project Build Packages Info Customize Plugin Tools Window Help
store
kawa Projects
OpcionA8
Fondo.java
graficaA8.java
principal8.java
General
OpcionA
OpcionA16
OpcionA32
OpcionB
OpcionB2
prueba
import java.lang.String;
import java.awt.*;
import javax.swing.*;
import java.lang.Object;
public class principal8 extends javax.swing.JFrame
{
String ciclo = "2 - Espera";
boolean habilitado = true;
boolean trama1 = false;
boolean trama2 = false;
boolean trama3 = false;
boolean trama4 = false;
boolean trama5 = false;
boolean tramaFin = false;
boolean primera_vez = true;
int puntero = 0;
boolean residuo = false;
int indice = 0;
boolean fin = false;
int iteración = 0;
int tirada = 0;
Output Build Find in Files
C:\j2sdk1.4.2_04\bin\javac.exe principal8.java
Working Directory - c:\Documents and Settings\Fco Javier\Escritorio\OpcionA8\
Class Path - .;c:\Kawa4.1\kawaclasses.zip;c:\j2sdk1.4.2_04\lib\tools.jar;c:\j2sdk1.4.2_04\jre\lib
File Compiled...
No Errors...
No Errors
Ln 23 Col 1
INS


NetBeans IDE 3.6 - Project Default
File Edit View Project Build Debug Versioning Tools Window Help
Filesys...
Runtime
Filesystems
C:\Documents and Setti
examples
JFrame
C:\Documents and Setti
JFrame *
JFrame [Form]
Palette
Swing AWT Layouts Beans
Inspector
Form JFrame
Other Components
[JFrame]
Properties of [JFrame]
Properties Events Code
Properties
defaultCloseOper...
HIDE
title
[JFrame]
Output

Capítulo 5
Implementación de la aplicación

5.1. Menú principal

La aplicación comienza mediante un menú de inicio donde se pueden elegir 2 opciones: Opción A: Multiplexación por división en el tiempo, y Opción B: Conmutación temporal con TDM. Si se elige la Opción A, aparecen a la izquierda del menú 4 botones, donde se seleccionará con cuántos canales queremos hacer la simulación, y si se elige la Opción B, a la derecha de la pantalla aparecen 2 botones, donde seleccionaremos una conmutación con TDM simple, o con sincronismo y señalización.

La clase que implementa el menú es la clase principal del programa `General`, que previamente hemos señalado como 'Main Class' mediante el archivo MANIFEST.MF, posteriormente explicaremos con todo detalle cómo se construye el archivo *.jar* final. Ésta es una clase que extiende de JFrame, donde según el botón pulsado, se mostrarán los distintos JFrames que componen el programa.

Por ejemplo, si se elige la opción de TDM de 4 canales, el botón pulsado tiene asociado un método manejador de eventos que hará lo que se pida cuando en dicho botón ocurra un evento, en este caso, de tipo ActionEvent, es decir, una pulsación. En este método se hará lo siguiente:

```
private void opcionA4Handler(java.awt.event.ActionEvent
evt)
{
      a4 = new OpcionA4();

      a4.show();
      (this).hide();
}
```

En el manejador ocurren 3 cosas:

1. Se crea un objeto de tipo OpcionA4, es decir, un JFrame con toda la funcionalidad del TDM de 4 canales.

2. Dicho JFrame creado, se muestra en pantalla por medio del método `show()`

3. Mediante la referencia `(this)` nos referimos al JFrame del propio menú, de manera que como ya no nos hará falta, ocultamos el menú con el método `hide()`.

De esta manera, todos los botones del menú tienen asociado un manejador que creará y mostrará JFrames del tipo que corresponda según el tipo de simulación elegida, y finalmente, cerrará el menú.

Con esta porción de código, el aspecto de la ventana cambia, sobre todo en los botones, es una forma de adaptarla gráficamente a Windows XP:

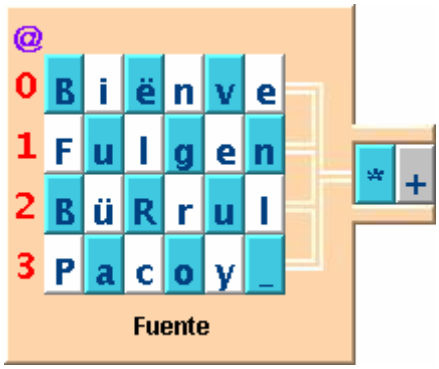
```
try
{
      UIManager.setLookAndFeel("com.sun.java.swing.plaf.window
      s.WindowsLookAndFeel");
      SwingUtilities.updateComponentTreeUI(this);
}
catch (Exception e){}
```

A continuación, pasaremos a explicar la implementación de todas las opciones del programa.

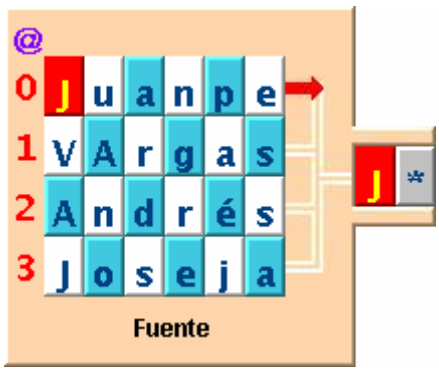
5.2. Opción A

En la opción A nos encontramos con 4 formas de hacer un multiplexado por división en el tiempo, con 4 canales, con 8, con 16, y con 32. Explicaremos la de 4 canales, ya que es la más simple, y por no redundar en las otras, ya que el funcionamiento es exactamente igual.

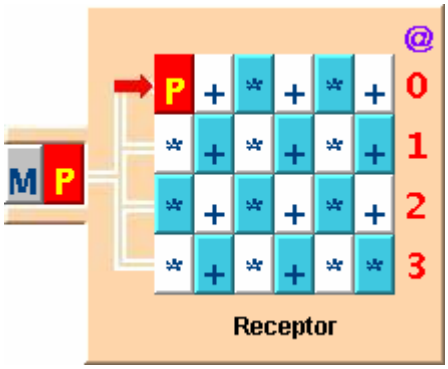
Lo primero que nos encontramos es el módulo fuente, donde se encuentran las letras a transmitir, las cuales se han separado en paneles de distinto color para hacer hincapié en el tratamiento individualizado de cada letra, y así facilitar la comprensión del algoritmo.



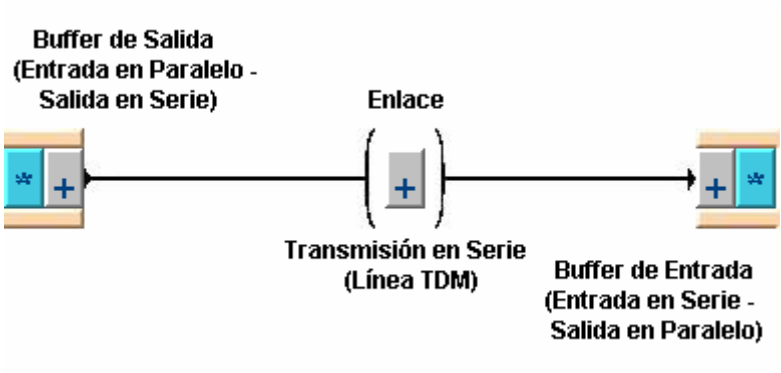
En el momento en que en la fuente se produce una transmisión, aparece una flecha roja en las líneas que van desde el canal que esté transmitiendo hasta el buffer de salida, para indicar que la letra está saliendo de la fuente.



El módulo receptor tiene básicamente el mismo aspecto, y aquí se visualizan las flechas rojas en el instante en que el receptor esté recibiendo la letra. Éstas flechas se pueden ver en las líneas que van desde el buffer de entrada hasta el principio del canal que está recibiendo.



En la línea de transmisión nos encontramos con los buffers de salida de la fuente y de entrada del receptor, y con 3 visualizaciones en distintos lugares de la letra que se encuentra en el enlace. En el buffer de salida, los datos llegan a él en paralelo, es decir, primero un dato de un canal, luego de otro, etc., y salen de él para viajar como una trama en serie por la línea. A este proceso se le llama PISO (parallel in, serial out), y cuando los datos llegan al receptor, se produce lo contrario, una conversión SIPO (serial in, parallel out).



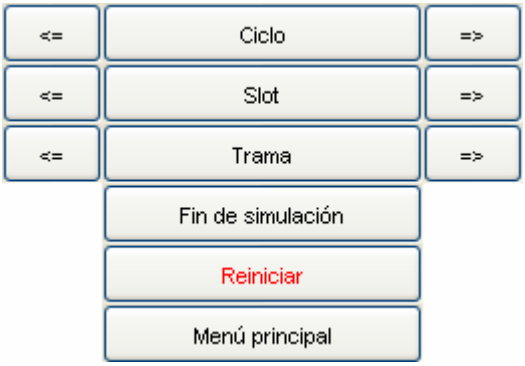
Estado de la simulación

Estos indicadores nos muestran la trama que en ese instante se está transmitiendo desde la fuente, a qué canales corresponden las letras que se encuentran en el buffer de entrada, línea de transmisión y línea de salida, y finalmente, en qué ciclo se encuentra la transmisión de un slot temporal, es decir, 0 – Salida, 1 – Entrada, y 2 – Espera.

Como vemos, primero visualiza la trama que en ese instante se está transmitiendo, y se sustituirá por la siguiente cuando empiece a transmitirse ésta, no cuando haya terminado la transmisión de la anterior. En slots, vemos los canales a los que en ese momento pertenecen las letras que ocupan los buffers de salida y entrada, y el enlace. Finalmente, vemos en cuál de los 3 ciclos se encuentra la transmisión en un slot de tiempo.

Botones de control

Con estos botones, el usuario interactuará con el simulador. Cuando estén deshabilitados, querrá decir que no tiene sentido pulsar el botón en cuestión, por ejemplo, al iniciar la aplicación todos los botones de marcha atrás están deshabilitados, incluido el de reinicio, porque no tiene sentido ir hacia atrás cuando todavía ni se ha empezado. Lo mismo ocurre con la marcha adelante y el botón fin de simulación, cuando ésta llega al final.



El botón **ciclo adelante** (=>) transmite 1/3 de slot, es decir, hace, o un ciclo de salida, o un ciclo de entrada, o un ciclo de espera. Si el ciclo es de salida, la letra que se vaya a coger de la fuente se ilumina, y al mismo tiempo pasa a ocupar el buffer de salida, la letra que estaba en el buffer de salida pasa a transmitirse por la línea TDM, y la que estaba en la línea TDM, pasa al buffer de entrada del receptor. Si el ciclo es de entrada, la letra que se encuentra en el buffer de entrada, sigue ahí pero además se posiciona en su lugar correspondiente en el receptor, iluminándose dicho lugar, y si el ciclo es de espera, no se hace nada.

El botón **slot adelante** transmite un slot completo, es decir, 3 ciclos. Si nos encontramos en un ciclo de salida o de entrada porque previamente hemos estado utilizando el botón de ciclo adelante, no adelanta 3 ciclos, sino que adelanta 2 en el caso de que estemos en 0 – Salida o 1 ciclo en el caso de que estemos en 1 – Entrada, esto se hace para dar por terminada la transmisión de ese slot para en las siguientes pulsaciones avanzar 3 ciclos con total normalidad, siempre empezando y finalizando en el ciclo de espera. Esto no sucede así con el botón **slot atrás**, ya que la finalidad de dicho botón no es ya la correcta simulación hacia atrás, sino la rectificación de un paso dado, o para volver a un estado concreto.

El botón **trama adelante** transmite una trama entera de 4 letras (en esta opción de 4 canales, porque en la de 8 canales serían 8 letras, en 16 canales 16 letras, etc.), lo que supone unos 12 ciclos. Aquí pasa lo mismo que con el botón slot adelante, estemos en el estado que estemos y en el ciclo que estemos, al pulsar trama adelante llegaremos al punto en el que deja de enviarse la trama que se está transmitiendo, la cual la podemos apreciar en el estado de la simulación, y así acabar también en el ciclo de espera, para luego reanudar la marcha de forma normal, es decir, de 4 en 4 slots. Esto tampoco es aplicable al botón **trama atrás** por lo dicho en el párrafo anterior.

El botón **Fin de Simulación** transmite los 4 canales enteros, y se puede contemplar en el receptor cómo están ya todos situados en sus correspondientes

direcciones. Es como si recorriéramos 78 ciclos, 26 slots, o 6 tramas (son 6 las que se transmiten totalmente, aunque en realidad la simulación termina con la trama número 7 transmitiéndose).

El botón **Reiniciar** devuelve todo a su estado inicial.

El botón **Menú Principal** nos permite acceder a la ventana del menú que aparece al iniciarse el programa.

5.2. Opción B1

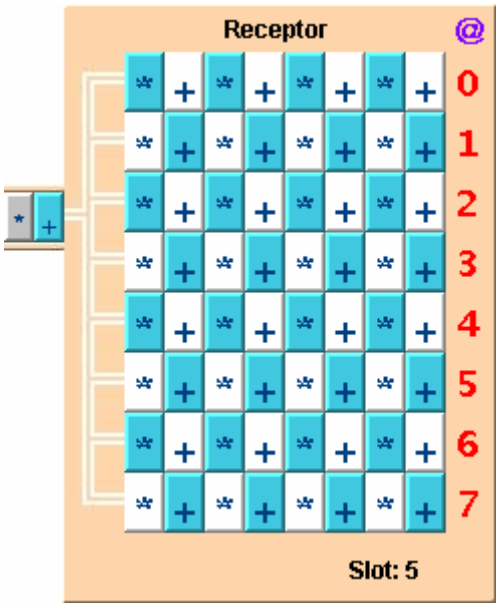
En esta opción contamos con 8 canales a transmitir, pero la diferencia con el TDM simple radica en que cada canal tiene un destino aleatorio predefinido que no tiene por qué ser su misma posición en el receptor. Se hace necesaria así la presencia de un conmutador o Switch, que se encargue de almacenar las letras hasta que les llegue su turno de salir del switch, mirando la memoria de control.

A continuación se describirán las partes de un TDM con una etapa T:

En la fuente podemos distinguir 8 canales, como en la opción del TDM de 8 canales simple, pero cada uno tiene su propio destino, cosa que también ocurría en el modo simple, pero de forma que cada canal tenía el destino en el receptor en la misma posición que ocupaba en la fuente, aquí también puede darse ese caso, pero lo más usual es que vaya a otra posición distinta:

El receptor tiene 8 canales, cada uno con 8 posiciones para guardar voz, en vez de 6, esto se debe a que hay que guardar los caracteres que ya había en la memoria de voz justo al principio de la simulación, representados mediante “*”, y

también a la transmisión del carácter “+”, que se produce cuando en la fuente ya no hay información útil que transmitir, y se transmite “+” de relleno, para terminar la simulación.



Tanto en la fuente como en el receptor, hay dos etiquetas que indican cuál es el canal que está transmitiéndose (caso de la fuente), y qué canal está recibiendo (caso del receptor). Se puede saber el canal porque, en realidad este registro indica qué slot de la trama que se está transmitiendo se está enviando en ese momento, y si es el slot 5, este slot pertenece al canal 5.

La etapa T, es decir, el switch, empieza en el buffer de entrada, donde se realiza una conversión SIPO (serial in, parallel out), donde las letras entran en serie y se depositan en paralelo en la memoria de voz. Hay un panel donde se indica el número de trama **que se está despositando en la memoria de voz**, el slot de dicha trama que en ese momento se está almacenando, y el tipo de ciclo que se está produciendo, donde el 0 representa “Lectura”, el 1 “Escritura” y el 2 “Espera”.

Abajo tenemos las direcciones RAM, que se seguirán secuencialmente según se vaya iterando, y son fijas. Arriba de las direcciones RAM tenemos la memoria de control, cuyo contenido queda fijado por los destinos a los que irán las letras de los distintos canales de la fuente. Por ejemplo, si el canal **1** de la fuente tiene como destino el canal **7** del receptor, la memoria de control que está justo encima de la dirección RAM número **7** del switch, tendrá como contenido un **1**. Arriba de la memoria de control tenemos la memoria de voz, donde se irán almacenando las letras que lleguen del buffer de entrada, hasta que les toque ir al buffer de salida y posteriormente ser reemplazadas.

Lo primero que ocurre en el switch es un ciclo de Lectura. En él, se producen cuatro sucesos:

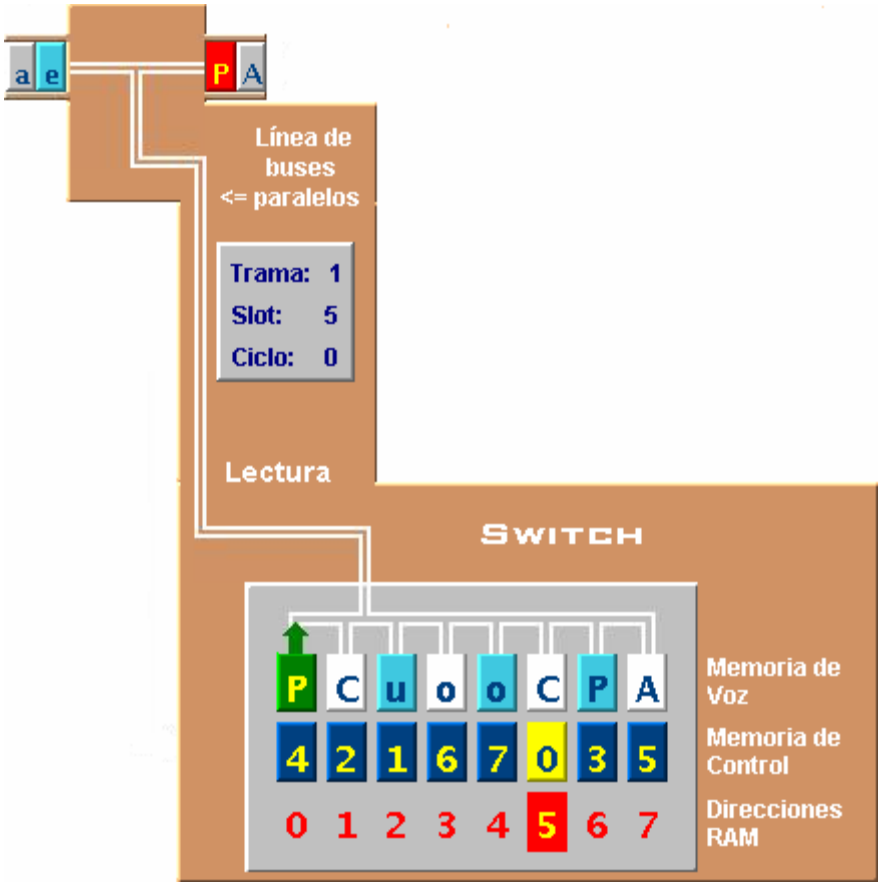
1. Hay un desplazamiento tanto en la primera línea TDM como en la segunda.

2. Se atiende a la dirección RAM que toque, se empieza en la 0, y después de un slot, se continúa en la 1, y luego en la 2, y así sucesivamente.

3. En la memoria de control que corresponda con la dirección RAM que toque, se lee su contenido, que es un número.

4. Éste número leído de la memoria de control será la posición en la memoria de voz donde se **leerá** la letra que en este mismo ciclo, pasa al buffer de salida del switch, que hace una conversión PISO (parallel in, serial out), donde los datos llegan en paralelo desde la memoria de voz, y se transmiten en serie por la segunda línea TDM.

El ciclo de lectura gráficamente se puede apreciar por una flecha verde en cualquiera de las 8 posiciones de la memoria de voz.



En la siguiente iteración se produce un ciclo de escritura, que tiene las siguientes fases:

1. Se atiende a la dirección RAM que toque.

2. En la memoria de voz que corresponda con la dirección RAM seleccionada, se sustituirá la letra que había antes por la que está en el buffer de entrada.

El ciclo de escritura se puede apreciar gráficamente por una flecha roja en cualquiera de las 8 posiciones de la memoria de voz.

En la siguiente iteración, en el ciclo de espera, no se hace nada, para simular el retardo.

5.2. Opción B2

Esta opción es similar a la anterior, pero con diferencias sustanciales. Hay 6 canales de datos en vez de 8, los otros 2 restantes son para sincronismo y señalización, para los que se les ha reservado las posiciones 0 y 4 respectivamente. La memoria de control no está definida inicialmente, se irá completando a lo largo de la simulación mediante la señalización.

El módulo fuente tiene el siguiente aspecto:

Como vemos, en el canal 0 se encuentra el canal de sincronismo, donde el sincronismo de multitrama viene representado por el carácter “^”, y el sincronismo de trama es el carácter “!”. El canal 4 es el de señalización, vemos que coincide con los destinos a los que irán a parar las letras de la fuente. Los destinos se han tomado de forma aleatoria, a excepción de los destinos 0 y 4, que están reservados para los canales de sincronismo y señalización (0 y 4 respectivamente).

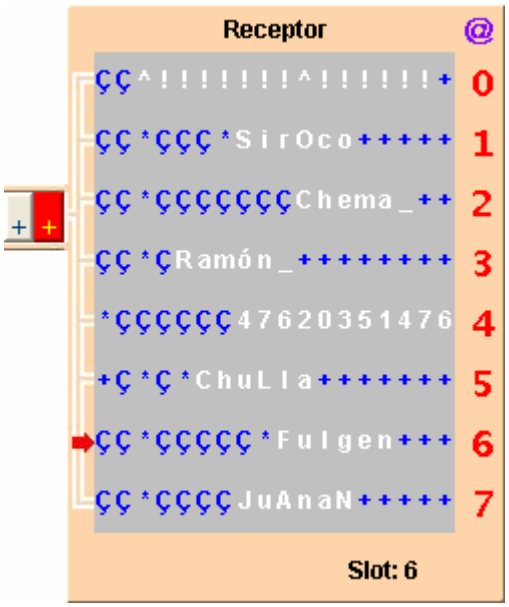
La simulación empieza con fuente y receptor desincronizados, cosa que no ocurría en la anterior opción, donde receptor y fuente tenían un desfase de unos 4 slots.

El funcionamiento en la fuente es el siguiente:

Al estar fuente y receptor desincronizados, en la etiqueta “slot” puede aparecer cualquier número, empezamos en un canal aleatorio. Por mucho que iteremos no se mandan datos útiles hasta que el registro slot marque 0, es entonces cuando enviamos por primera vez el sincronismo de multitrama. Se va iterando y no se mandan datos, hasta que llegamos al canal 4, donde se manda la señalización del canal 0, de tal forma que cuando volvemos a la transmisión del canal 0, se manda el sincronismo de trama. Se sigue sin mandarse datos hasta que nuevamente llegamos al canal 4. Tomando el ejemplo del dibujo, se manda señalización del canal 7, luego el canal de la fuente que tiene por destino el canal 7 del receptor, es decir, el canal 1, ya puede transmitir con normalidad sus datos. De esta forma, los canales de voz de la

fuente tendrán permiso para transmitir en cuanto se haya mandado la señalización del canal que tienen por destino en la fuente. De no tener permiso, lo que se mandará será relleno, representado por el carácter “*”. Cuando se terminan los datos útiles en los canales de la fuente, pero aún así hay que seguir transmitiendo para terminar la simulación, se manda el carácter “+”, que también es relleno.

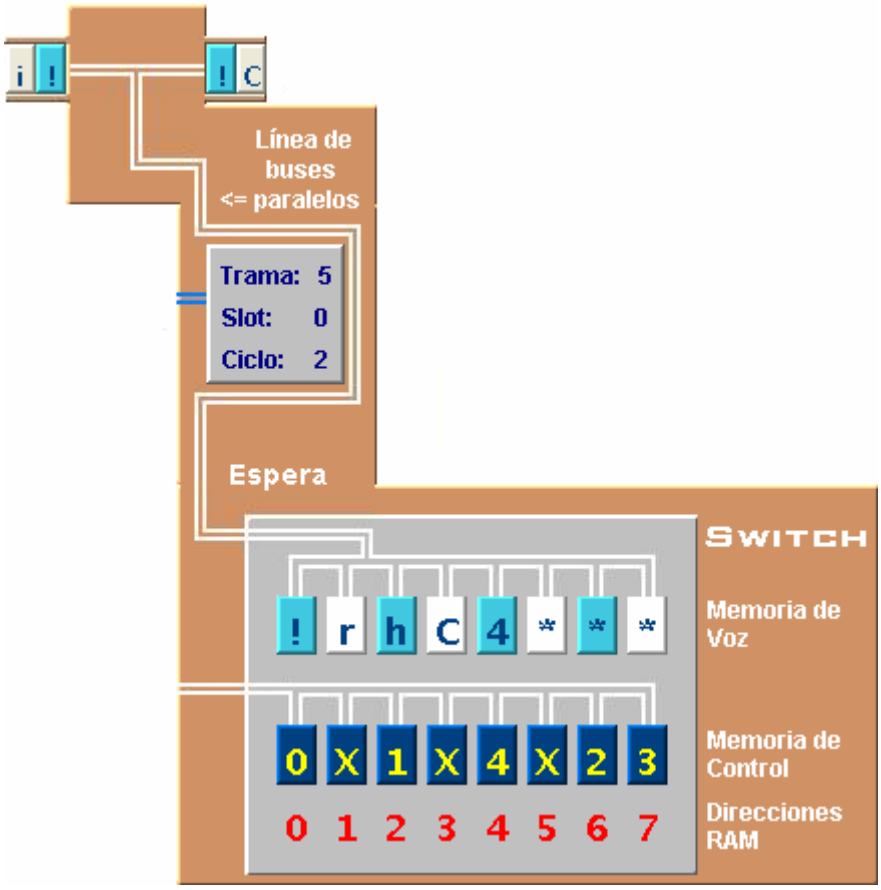
A contunuación, veremos el módulo receptor, una vez finalizada la simulación, para que se vea con más claridad



Cada canal tiene muchos más espacios que en la fuente para ver toda la progresión. Se puede ver cómo los primeros caracteres de cada canal son ruido, representados por el carácter “Ç”, esto se debe a que aproximadamente durante el primer tercio de la simulación la memoria de control se está completando. A continuación tenemos ya la información útil (resaltada en blanco), y los demás caracteres representados por “+” son la consecuencia de transmitir relleno desde la fuente cuando ya no hay información útil que transmitir, pero que son necesarios para terminar correctamente la simulación.

El receptor inicialmente está desincronizado con la fuente, ya que los registros Slot de cada uno toman un valor aleatorio. El receptor se resetea, es decir, se sincroniza con la fuente cuando a éste le llega el sincronismo de multitrama, de forma que el desfase entre fuente y receptor es de 4 slots, como en la opción anterior.

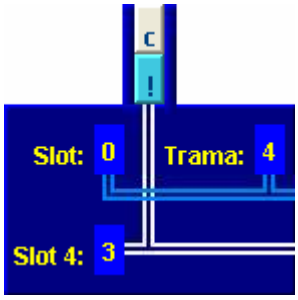
En esta opción, el switch tiene el siguiente aspecto:



La memoria de voz y las direcciones RAM son iguales que en la Opción B1. El cambio más significativo lo encontramos en la memoria de control, donde inicialmente están todos los valores a X. Eso quiere decir que la memoria de control no está definida inmediatamente después de que se generen los destinos, sino que se debe ir completando con la información que llegue del canal de señalización. Podemos ver que la memoria de control está conectada a unos buses que vienen desde el módulo de sincronismo y señalización que veremos posteriormente, módulo que también controla los registros contadores de tramas, de slots y de ciclos del switch.

En el ejemplo del dibujo, si llegara el turno de la dirección RAM 1, observamos que la memoria de control a la que corresponde esta dirección aún no está definida, luego a la segunda línea TDM lo que se envía es ruido.

De rellenar la memoria de control y de las labores de sincronismo se encarga el módulo de sincronismo y señalización:



A este módulo le llegan los datos desde la primera línea TDM (la que va del buffer de salida de la fuente al buffer de entrada del switch). En principio, los registros Slot y Trama aparecen con signo de interrogación, mientras que los registros contadores del switch van independientes. En el momento en que a este módulo le llega el sincronismo de multitrama, se produce un reset en dicho módulo y también en los registros contadores del switch, siendo los registros Slot y Trama del switch y del módulo idénticos a partir de ese momento.

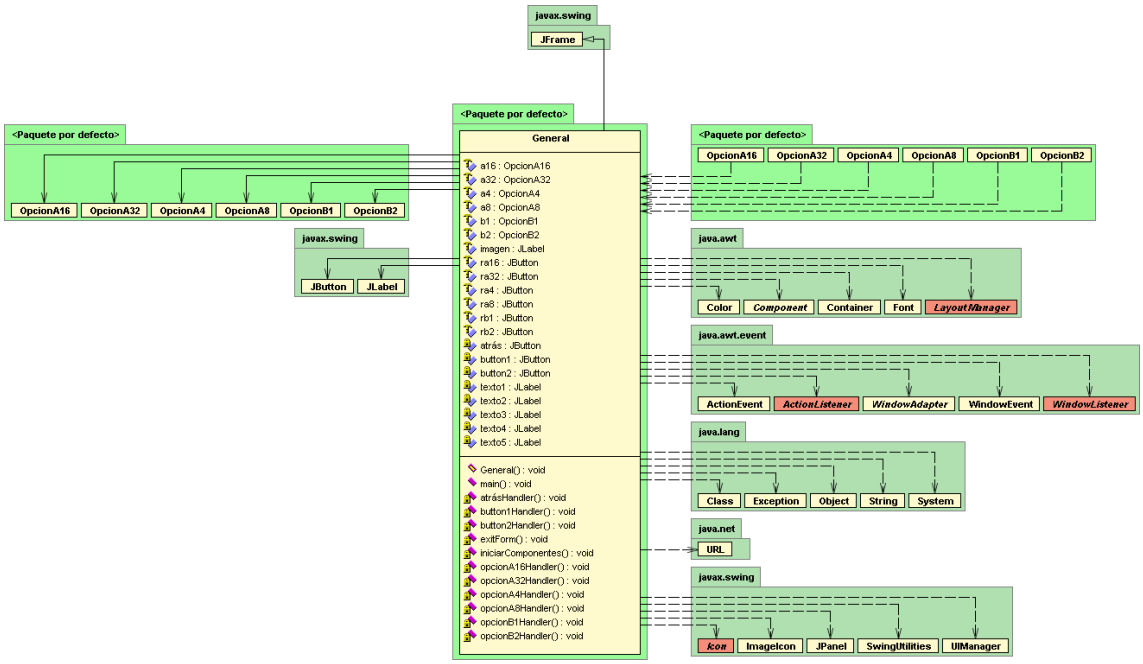
El proceso de señalización empieza cuando al módulo le llega por primera vez información de señalización, es decir, un 0. Este 0 pasa a ocupar el registro Slot 4, inicialmente vacío, a la misma vez que se rellena la posición 0 de la memoria de control con el número 0. Este último número 0 no tiene nada que ver con el 0 que llega del canal de señalización, esta posición se rellena con dicho valor porque se atiende al registro Trama del módulo de sincronismo y señalización. Por ejemplo, si el siguiente dato de señalización fuera un 5, en la memoria de control que tenga la posición 5 se pondrá como contenido un 1, que es el número de trama que se está transmitiendo.

Todo esto ocurre en el ciclo de lectura, en el ciclo de escritura lo que se hace es escribir dicho dato de señalización en la memoria de voz, y la posición la decide el resgistro Slot del módulo de sincronismo y señalización, que será siempre la posición 4.

Ahora procederemos a explicar detalladamente las clases que componen el programa, para ello nos ayudaremos de los diagramas UML de cada clase, con los que podremos ver las variables utilizadas, los métodos, y las clases que interactúan con dicha clase, ya sea por herencia o composición. Empezaremos con la clase General.

General

Ésta es la clase principal desde donde se llaman a las otras clases mediante composición, es decir, la clase General está compuesta de objetos del tipo OpcionA4, OpcionA8, OpcionA16, OpcionA32, OpcionB1 y OpcionB2. La clase General, al igual que todas las demás, es un JFrame (hereda de JFrame).



Vamos a ver los métodos:

- `General():`

  Es el método constructor, que además de configurar el tamaño de la ventana, llama al método iniciarComponentes().

- `iniciarComponentes():`

  Crea todos los botones y las etiquetas de texto.

- `opcionA4Handler(java.awt.event.ActionEvent evt)`

Es el manejador que se ejecuta cuando pulsamos el botón del que escucha. En este manejador se crea el objeto del tipo OpcionA4, se muestra por pantalla, y cierra la ventana del menú principal. Los manejadores de eventos de los otros botones funcionan de manera análoga a éste con sus correspondientes objetos del resto de clases.

- `button1Handler(java.awt.event.ActionEvent evt)`

Este manejador sustituye los botones que aparecen nada más abrirse el menú principal por los del correspondiente submenú, en este caso, por las 4 opciones del TDM. En el otro manejador ocurre lo mismo con sus correspondientes opciones.

- `exitForm(java.awt.event.WindowEvent evt)`

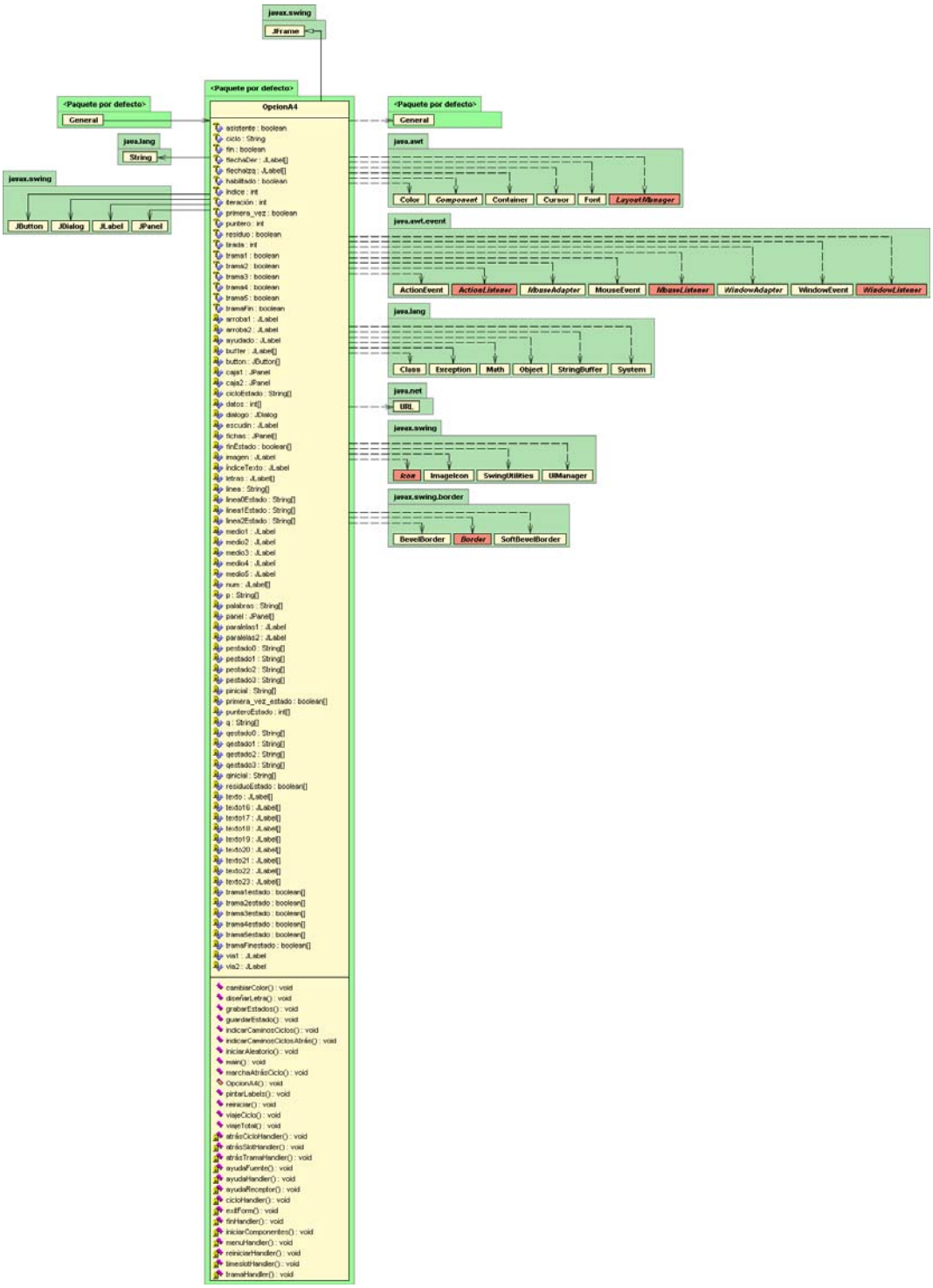
  Este método sirve para salir de la aplicación.

- `main(String args[])`

Éste es el método main de todo programa, aquí se crea un objeto de tipo General, y se muestra por pantalla.

OpcionA4

Es una clase que extiende de JFrame, y donde están tanto el algoritmo de TDM de 4 canales como la interfaz gráfica. Las clases OpcionA8, OpcionA16 y OpcionA32 tienen unas características análogas a la clase OpcionA4, de modo que sólo comentaremos esta última.



En esta clase, destacamos los siguientes métodos:

- `public void cambiarColor(JPanel p, JLabel l)`

Ilumina una letra, es decir, pone el panel donde está situada en rojo y el color de la letra se vuelve amarillo.

- `public void diseñarLetra(JPanel p[], JLabel l[], int x, int y, int n, int f, int c)`

Se encarga de diseñar cada letra según las circunstancias, es decir, su colocación, su color, y a qué JPanel le añadimos el JLabel donde está ubicada la letra. Este método nos ahorra muchísimo código, ya que cada letra puede requerir 8 líneas o más; con este método tan sólo basta una sola línea.

- `public void guardarEstado()`

Este método guarda todos los posibles estados de todas las variables en su array correspondiente, es decir, todos los posibles estados de q[0] se guardarán en el array qestado0[], pero este método sólo lo hace una vez, para hacerlo todas las posibles iteraciones se ayudará del método grabarEstados().

- `public void grabarEstados()`

Este método se llama nada más iniciarse la aplicación, y hace un viaje completo hasta el final de la simulación, guardando los estados de todas variables en cada iteración, y reiniciando posteriormente. Primero llama al método guardarEstado(), con lo que se graban todas las variables en su estado inicial, luego llama al método viajeCiclo() para pasar al siguiente estado, y el contador de los arrays de los estados se incrementa en uno, y vuelve a hacer el proceso hasta el final de la simulación. Finalmente reinicia el sistema para que nosotros podamos simular, pero las variables de estado no quedan reiniciadas, quedarán ahí grabadas para cuando necesitemos utilizar la marcha atrás.

- `public void marchaAtrásCiclo()`

El valor actual de las variables pasará a ser el valor que tienen todas sus variables de estado en el instante anterior, es decir, en la posición contador – 1 del array (índice – 1). Posteriormente, se decrementa en uno este contador.

- `public void indicarCaminosCiclos()`

Sirve para ver de qué canales son las letras que están en el buffer de la fuente, en la línea, y en el buffer del receptor, y esto se mostrará en las etiquetas del estado de la simulación.

- `public void iniciarAleatorio()`

Aquí se generan de forma aleatoria las 4 palabras de entre las 32 posibles y se sitúan en la fuente. En java, para obtener un número aleatorio se hace uso de la función **Math.random(),** que devuelve un número aleatorio entre 0 (inclusive) y 1(exclusive), de tal forma que si queremos un número aleatorio de 0 a 32, el resultado de la función Math.random() lo multiplicamos por 32, y al resultado le hacemos un casting a entero. Por ejemplo:

```
int num = (int)(Math.random()*32);
```

- `public void reiniciar()`

  Devuelve todas las variables a su estado inicial, excepto a las de estado, que seguirán con sus valores hasta que el usuario cierre la ventana o vuelva al menú principal.

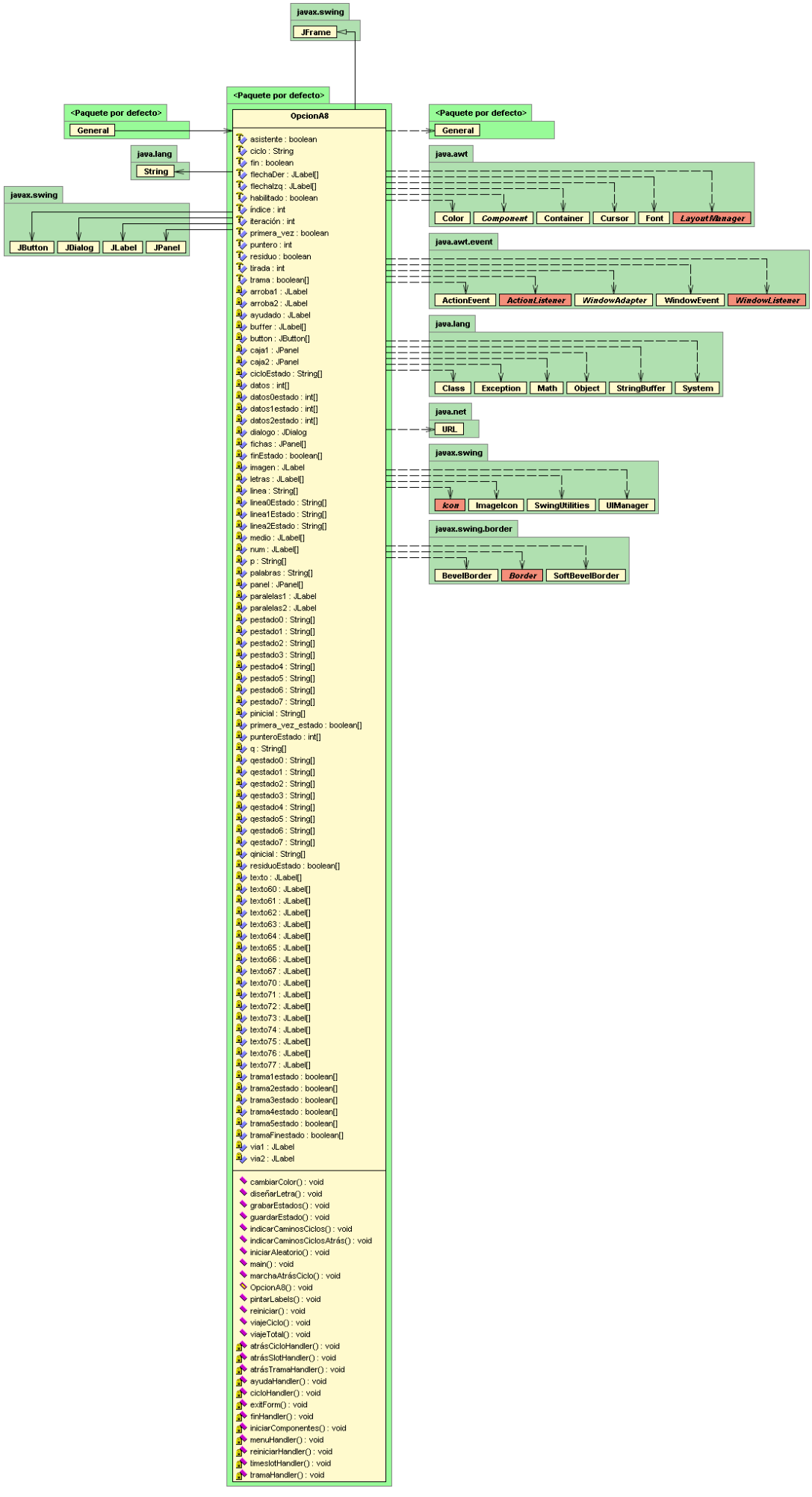
- `public void viajeTotal()`

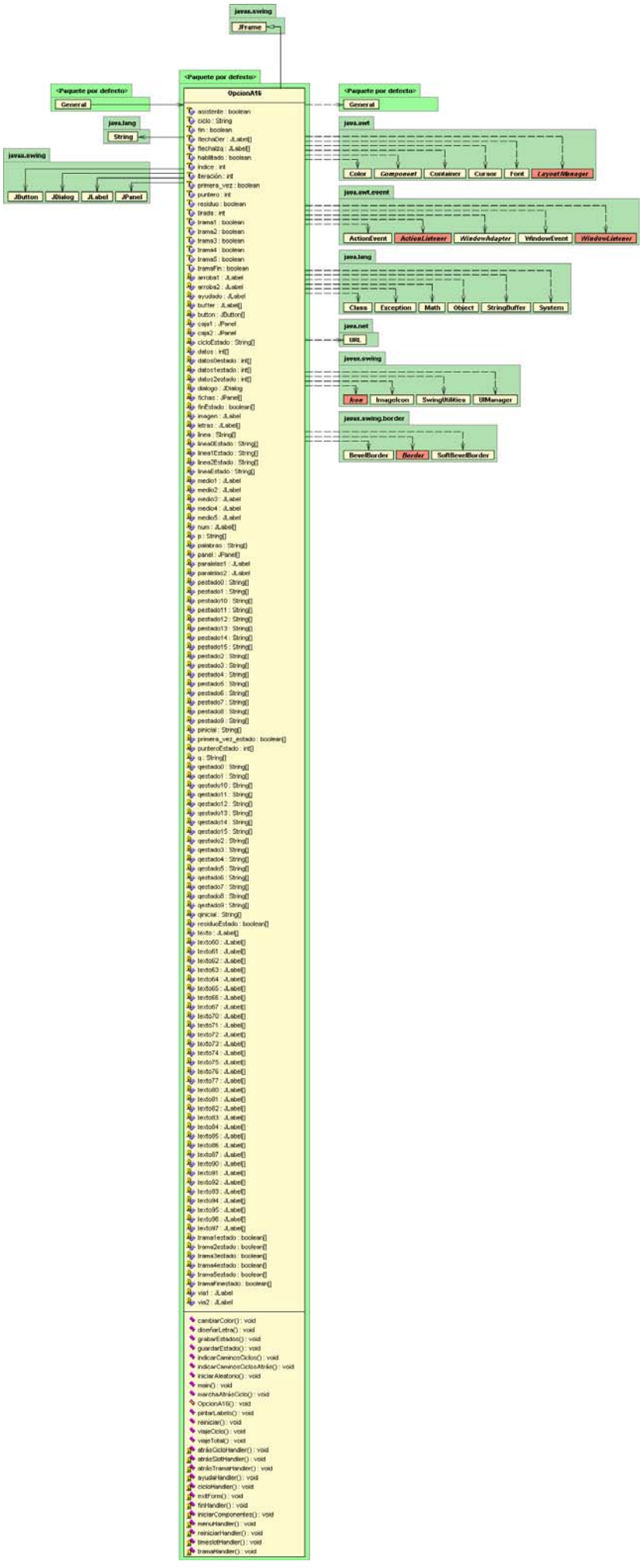
  Llama al método viajeCiclo() tantas veces como falte para llegar al estado final.

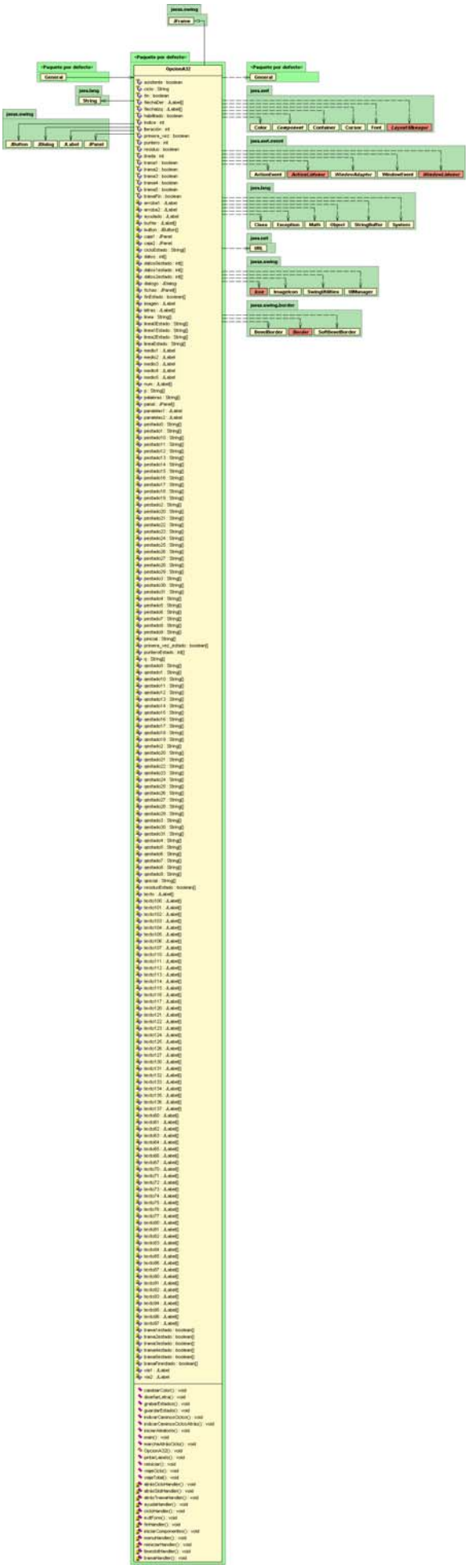
- `public void pintarLabels()`

  En cada iteración, las variables tienen un estado, luego necesitan ser también refrescadas en pantalla cada vez que se produce dicha iteración, de hacerlo sólo una vez, el usuario no podría ver los cambios. En cualquier acción que hagamos que suponga un cambio en las variables, se llamará al método pintarLabels(), que diseñará la ventana según el estado de la simulación.

Ahora vamos a ver los diagramas UML de las clases OpcionA8, OpcionA16, OpcionA32:


javax.swing
JFrame
<Paquete por defecto>
<Paquete por defecto>
General
java.lang
String
javax.swing
JButton
JDialog
JLabel
JPanel
OpcionA8
asistente : boolean
ciclo : String
fin : boolean
flechaDer : JLabel[]
flechaIzq : JLabel[]
habilitado : boolean
indice : int
iteración : int
primera_vez : boolean
puntero : int
residuo : boolean
tirada : int
trama : boolean[]
arroba1 : JLabel
arroba2 : JLabel
ayudado : JLabel
buffer : JLabel[]
button : JButton[]
caja1 : JPanel
caja2 : JPanel
cicloEstado : String[]
datos : int[]
datos0estado : int[]
datos1estado : int[]
datos2estado : int[]
dialogo : JDialog
fichas : JPanel[]
finEstado : boolean[]
imagen : JLabel
letras : JLabel[]
linea : String[]
linea0Estado : String[]
linea1Estado : String[]
linea2Estado : String[]
medio : JLabel[]
num : JLabel[]
p : String[]
palabras : String[]
panel : JPanel[]
paralelas1 : JLabel
paralelas2 : JLabel
pestado0 : String[]
pestado1 : String[]
pestado2 : String[]
pestado3 : String[]
pestado4 : String[]
pestado5 : String[]
pestado6 : String[]
pestado7 : String[]
pinicial : String[]
primera_vez_estado : boolean[]
punteroEstado : int[]
q : String[]
qestado0 : String[]
qestado1 : String[]
qestado2 : String[]
qestado3 : String[]
qestado4 : String[]
qestado5 : String[]
qestado6 : String[]
qestado7 : String[]
qinicial : String[]
residuoEstado : boolean[]
texto : JLabel[]
texto60 : JLabel[]
texto61 : JLabel[]
texto62 : JLabel[]
texto63 : JLabel[]
texto64 : JLabel[]
texto65 : JLabel[]
texto66 : JLabel[]
texto67 : JLabel[]
texto70 : JLabel[]
texto71 : JLabel[]
texto72 : JLabel[]
texto73 : JLabel[]
texto74 : JLabel[]
texto75 : JLabel[]
texto76 : JLabel[]
texto77 : JLabel[]
trama1estado : boolean[]
trama2estado : boolean[]
trama3estado : boolean[]
trama4estado : boolean[]
trama5estado : boolean[]
tramaFinestado : boolean[]
via1 : JLabel
via2 : JLabel
cambiarColor() : void
diseñarLetra() : void
grabarEstados() : void
guardarEstado() : void
indicarCaminosCiclos() : void
indicarCaminosCiclosAtrás() : void
iniciarAleatorio() : void
main() : void
marchaAtrásCiclo() : void
OpcionA8() : void
pintarLabels() : void
reiniciar() : void
viajeCiclo() : void
viajeTotal() : void
atrásCicloHandler() : void
atrásSlotHandler() : void
atrásTramaHandler() : void
ayudaHandler() : void
cicloHandler() : void
exitForm() : void
finHandler() : void
iniciarComponentes() : void
menuHandler() : void
reiniciarHandler() : void
timeslotHandler() : void
tramaHandler() : void
<Paquete por defecto>
General
java.awt
Color
Component
Container
Cursor
Font
LayoutManager
java.awt.event
ActionEvent
ActionListener
WindowAdapter
WindowEvent
WindowListener
java.lang
Class
Exception
Math
Object
StringBuffer
System
java.net
URL
javax.swing
Icon
ImageIcon
SwingUtilities
UIManager
javax.swing.border
BevelBorder
Border
SoftBevelBorder


javax.swing
JFrame
<Paquete por defecto>
OpcionA16
asistente : boolean
ciclo : String
fin : boolean
flechaDer : JLabel[]
flechaIzq : JLabel[]
habilitado : boolean
indice : int
iteración : int
primera_vez : boolean
puntero : int
residuo : boolean
trads : int
trama1 : boolean
trama2 : boolean
trama3 : boolean
trama4 : boolean
trama5 : boolean
tramaFin : boolean
arroba1 : JLabel
arroba2 : JLabel
ayudado : JLabel
buffer : JLabel[]
button : JButton[]
caja1 : JPanel
caja2 : JPanel
cicloEstado : String[]
datos : int[]
datos0estado : int[]
datos1estado : int[]
datos2estado : int[]
dialogo : JDialog
fichas : JPanel[]
finEstado : boolean[]
imagen : JLabel
letras : JLabel[]
linea : String[]
linea0Estado : String[]
linea1Estado : String[]
linea2Estado : String[]
lineaEstado : String[]
medio1 : JLabel
medio2 : JLabel
medio3 : JLabel
medio4 : JLabel
medio5 : JLabel
num : JLabel[]
p : String[]
palabras : String[]
panel : JPanel[]
paralelas1 : JLabel
paralelas2 : JLabel
pestado0 : String[]
pestado1 : String[]
pestado10 : String[]
pestado11 : String[]
pestado12 : String[]
pestado13 : String[]
pestado14 : String[]
pestado15 : String[]
pestado2 : String[]
pestado3 : String[]
pestado4 : String[]
pestado5 : String[]
pestado6 : String[]
pestado7 : String[]
pestado8 : String[]
pestado9 : String[]
pinicial : String[]
primera_vez_estado : boolean[]
punteroEstado : int[]
q : String[]
qestado0 : String[]
qestado1 : String[]
qestado10 : String[]
qestado11 : String[]
qestado12 : String[]
qestado13 : String[]
qestado14 : String[]
qestado15 : String[]
qestado2 : String[]
qestado3 : String[]
qestado4 : String[]
qestado5 : String[]
qestado6 : String[]
qestado7 : String[]
qestado8 : String[]
qestado9 : String[]
qinicial : String[]
residuoEstado : boolean[]
texto : JLabel[]
texto60 : JLabel[]
texto61 : JLabel[]
texto62 : JLabel[]
texto63 : JLabel[]
texto64 : JLabel[]
texto65 : JLabel[]
texto66 : JLabel[]
texto67 : JLabel[]
texto70 : JLabel[]
texto71 : JLabel[]
texto72 : JLabel[]
texto73 : JLabel[]
texto74 : JLabel[]
texto75 : JLabel[]
texto76 : JLabel[]
texto77 : JLabel[]
texto80 : JLabel[]
texto81 : JLabel[]
texto82 : JLabel[]
texto83 : JLabel[]
texto84 : JLabel[]
texto85 : JLabel[]
texto86 : JLabel[]
texto87 : JLabel[]
texto90 : JLabel[]
texto91 : JLabel[]
texto92 : JLabel[]
texto93 : JLabel[]
texto94 : JLabel[]
texto95 : JLabel[]
texto96 : JLabel[]
texto97 : JLabel[]
trama1estado : boolean[]
trama2estado : boolean[]
trama3estado : boolean[]
trama4estado : boolean[]
trama5estado : boolean[]
tramaFinestado : boolean[]
via1 : JLabel
via2 : JLabel
cambiarColor() : void
diseñarLetra() : void
grabarEstados() : void
guardarEstado() : void
indicarCaminosCiclos() : void
indicarCaminosCiclosAtrás() : void
iniciarAleatorio() : void
main() : void
marchaAtrásCiclo() : void
OpcionA16() : void
pintarLabels() : void
reiniciar() : void
viajarCiclo() : void
viajarTotal() : void
atrásCicloHandler() : void
atrásSlotHandler() : void
atrásTramaHandler() : void
ayudaHandler() : void
cicloHandler() : void
exitForm() : void
finHandler() : void
iniciarComponentes() : void
menuHandler() : void
reiniciarHandler() : void
timerslotHandler() : void
tramaHandler() : void
<Paquete por defecto>
General
java.lang
String
javax.swing
JButton
JDialog
JLabel
JPanel
<Paquete por defecto>
General
java.awt
Color
Component
Container
Cursor
Font
LayoutManager
java.awt.event
ActionEvent
ActionListener
WindowAdapter
WindowEvent
WindowListener
java.lang
Class
Exception
Math
Object
StringBuffer
System
java.net
URL
javax.swing
Icon
ImageIcon
SwingUtilities
UIManager
javax.swing.border
BevelBorder
Border
SoftBevelBorder



OpcionB1

Es la clase que implementa la funcionalidad y la interfaz gráfica de la demostración de conmutación en el tiempo. Tiene bastantes métodos iguales que las anteriores opciones, de modo que se obviarán y se mostrarán los nuevos métodos:

- `public void contarSlots()`

  Hace la cuenta en los registros Slot de la fuente y del receptor.

- `public void diseñarLetraReceptor(JPanel p[], JLabel l[], int x, int y, int n, int f, int c)`

  Tiene la misma funcionalidad que el método diseñarLetra(), pero éste está adaptado a un receptor con 2 espacios más para datos.

- `public void situarControles()`

  Aquí se fija el contenido de la memoria de control según los destinos elegidos

OpcionB2

Contiene funcionalidad (algoritmo) e interfaz gráfica. Los métodos más destacados son:

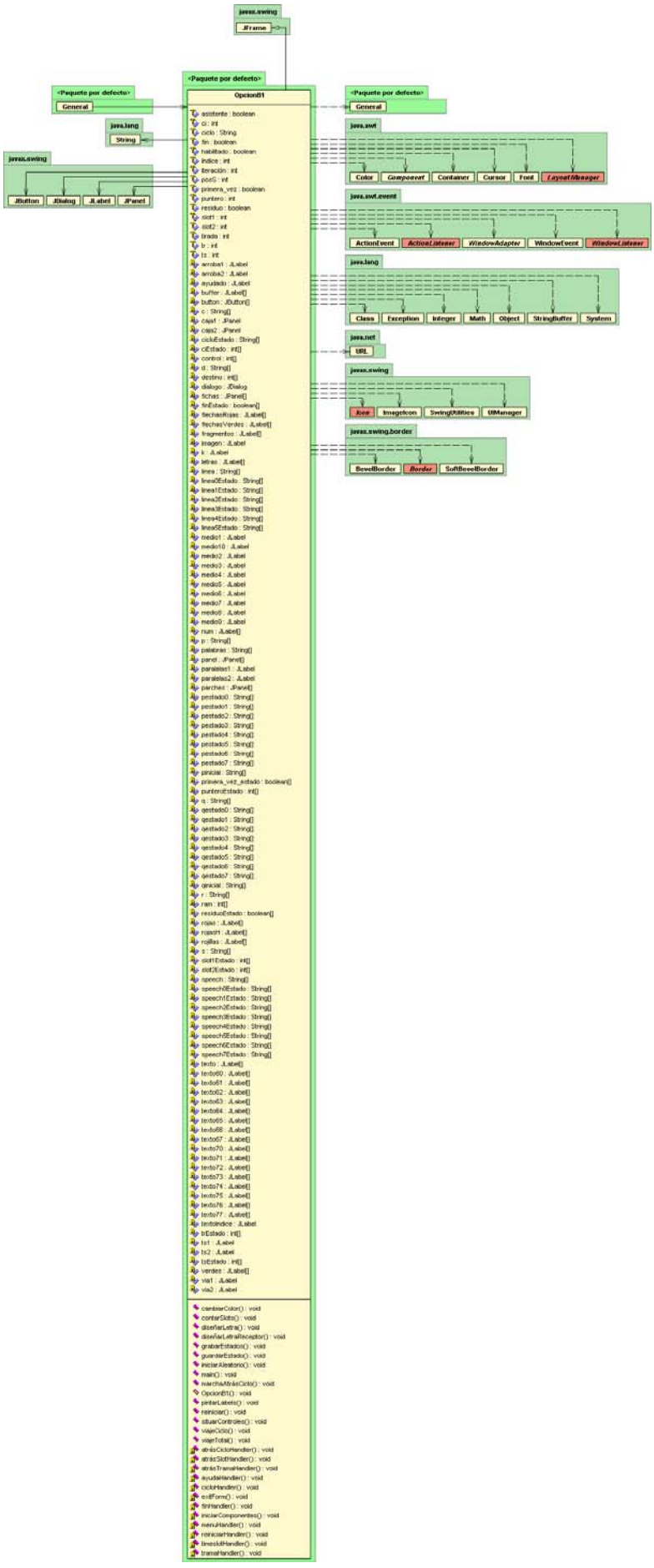
- `public void esLetra(String s, JLabel l[])`

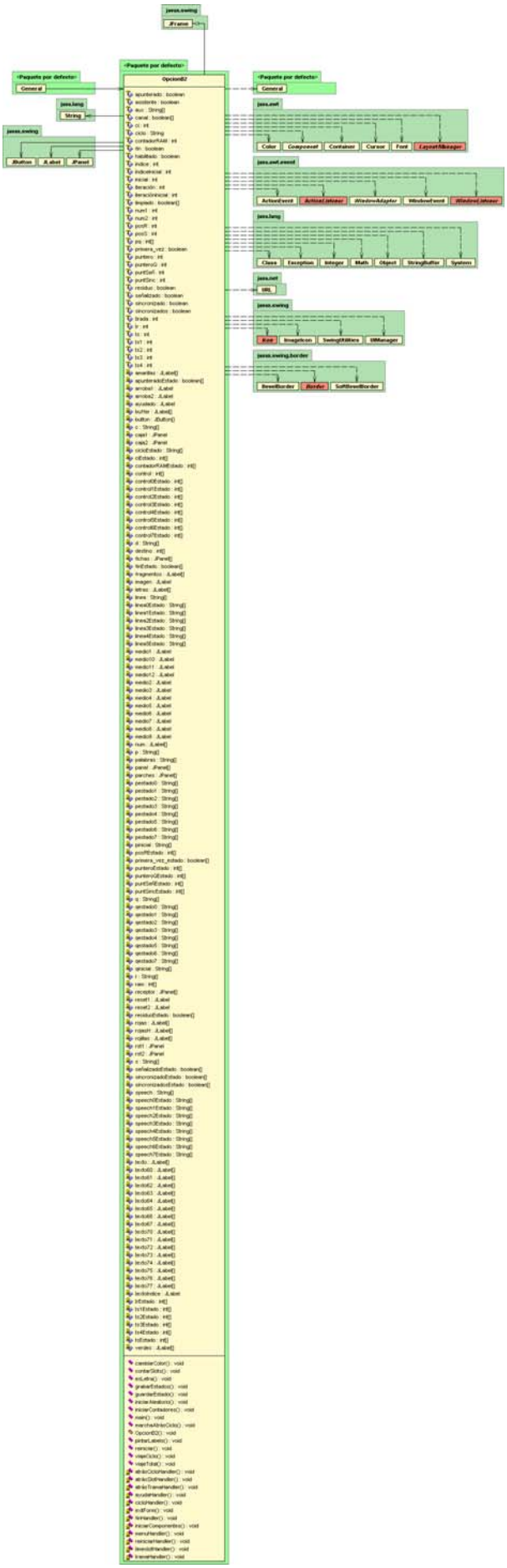
  En el receptor hay ruido, información útil e información de relleno, este método resalta la información útil en blanco.

- `public void iniciarContadores()`

  Genera los valores aleatorios de los registros Slot de fuente y receptor.

A contuniación podemos ver los diagramas UML de las clases OpcionB1 y OpcionB2:


javax.swing
JFrame
<Paquete por defecto>
Opcion01
<Paquete por defecto>
General
<Paquete por defecto>
General
java.lang
String
javax.swing
JButton
JDialog
JLabel
JPanel
java.awt
Color
Component
Container
Cursor
Font
LayoutManager
java.awt.event
ActionEvent
ActionListener
WindowAdapter
WindowEvent
WindowListener
java.lang
Class
Exception
Integer
Math
Object
StringBuffer
System
java.net
URL
javax.swing
Icon
ImageIcon
SwingUtilities
UIManager
javax.swing.border
BevelBorder
Border
SoftBevelBorder



Capítulo 6
Uso académico de la aplicación

6.1. Teoría de la conmutación

Antes de empezar a comentar el uso académico del programa, se comentarán brevemente conceptos básicos teóricos de conmutación.

Conmutación

La conmutación de circuitos es la técnica que permite que dos terminales, emisor y receptor, se comuniquen a través de un circuito único y específico, establecido para tal propósito antes del inicio de la misma y liberando una vez que ha terminado, quedando en este caso a disposición de otros usuarios para su utilización.

La conmutación de circuitos es orientada a la conexión y por tanto consta de las 3 fases siguientes:

- Fase de establecimiento de la llamada.
- Fase de transferencia de la información.
- Fase de desconexión de la llamada.

Durante el establecimiento de la llamada se reservan recursos físicamente (canales dentro de la trama TDM) de forma que los bits que entran por un puerto son conmutados “instantáneamente” a un canal de un puerto de salida.

Este tipo de redes son de nivel 1, es decir, no hay propiamente protocolo, y así se trata de transportar información analógica, fundamentalmente voz, de forma analógica o digital. La transmisión de información digital como fax y datos, necesita de una transformación digital/analógica y analógica/digital en sus extremos.

Tipos de conmutación

Según su operación:

- Manual y automática
- Analógica y digital
- Espacial y temporal

Según el tipo de servicio:

- De circuitos
- De paquetes

Según su utilización:

- Privadas

- Públicas

En nuestro simulador, los tipos de conmutación a utilizar son la conmutación de circuitos y la conmutación temporal.

Conmutación de circuitos

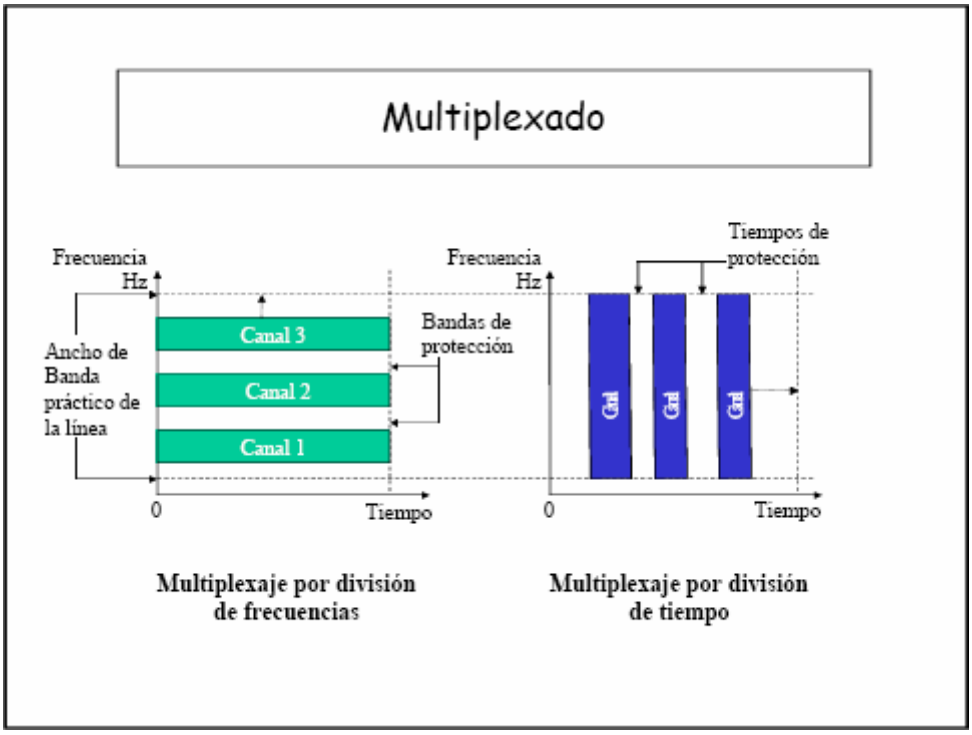
Es una técnica en la que los equipos que se comunican entre sí utilizan un canal físico dedicado extremo a extremo que se mantiene durante el tiempo de duración de la llamada o por el periodo de contatación.

Conmutación temporal

Método de conmutación de circuitos en que el tiempo es dividido en ranuras temporales y la información debe ir por un solo medio, de tal forma que la información es divida en dichas ranuras temporales para viajar por dicho canal y se reordena cuando llega a su destino.

6.2. ¿Qué es el multiplexado?

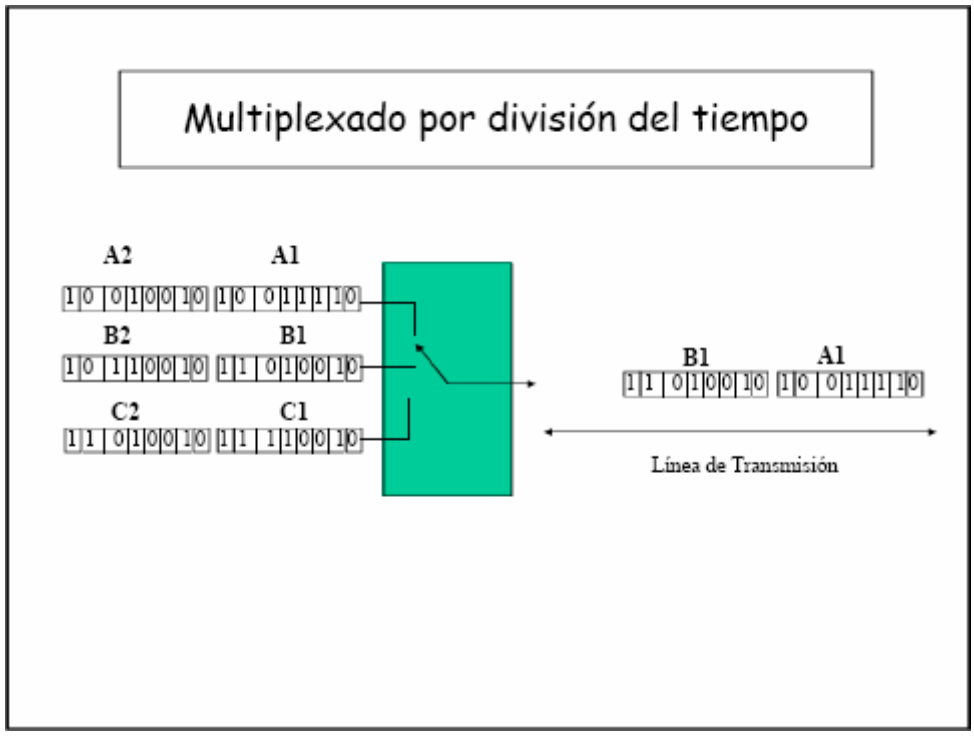
El multiplexado es la técnica de combinar varias señales diferentes y transmitirlas simultáneamente sobre un solo canal o medio sin que se interfieran. El principal objetivo de la multicanalización es la economía.



El multiplexado consiste en compartir una línea por muchos canales telefónicos y puede realizarse con base en la frecuencia, el llamado **multiplexado por división de frecuencia**, o con base en el tiempo, llamado **multiplexado por división en el tiempo.**

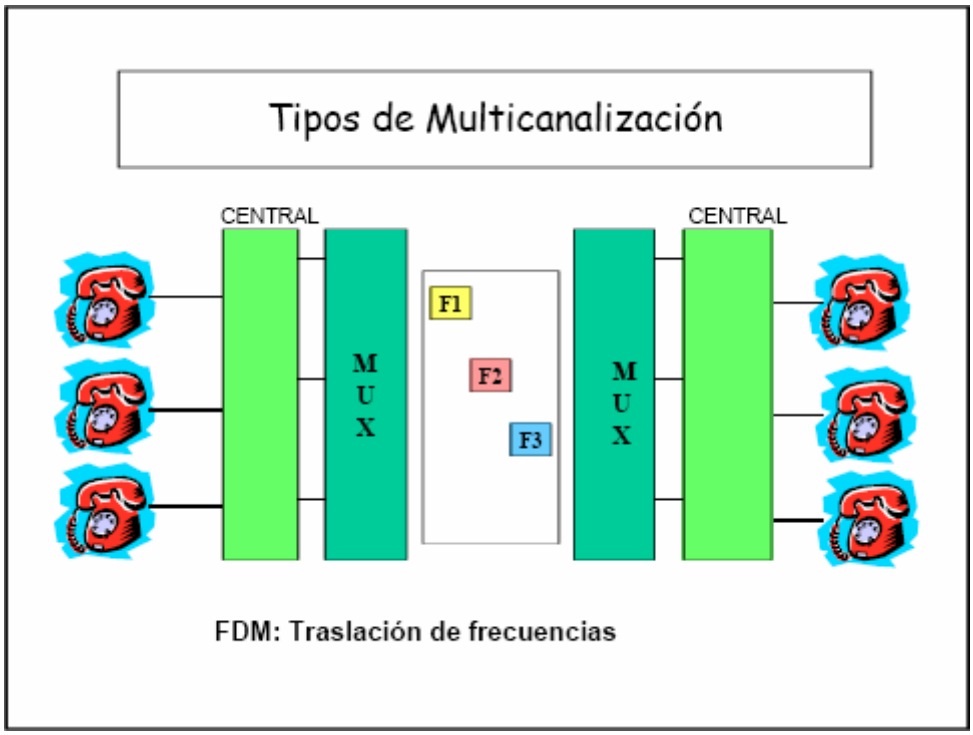
Multiplexado por división de frecuencia (FDM), cada canal de voz se asigna a una porción única del ancho de banda disponible en la línea, y todo el tiempo utiliza en forma exclusiva el ancho de banda asignado.

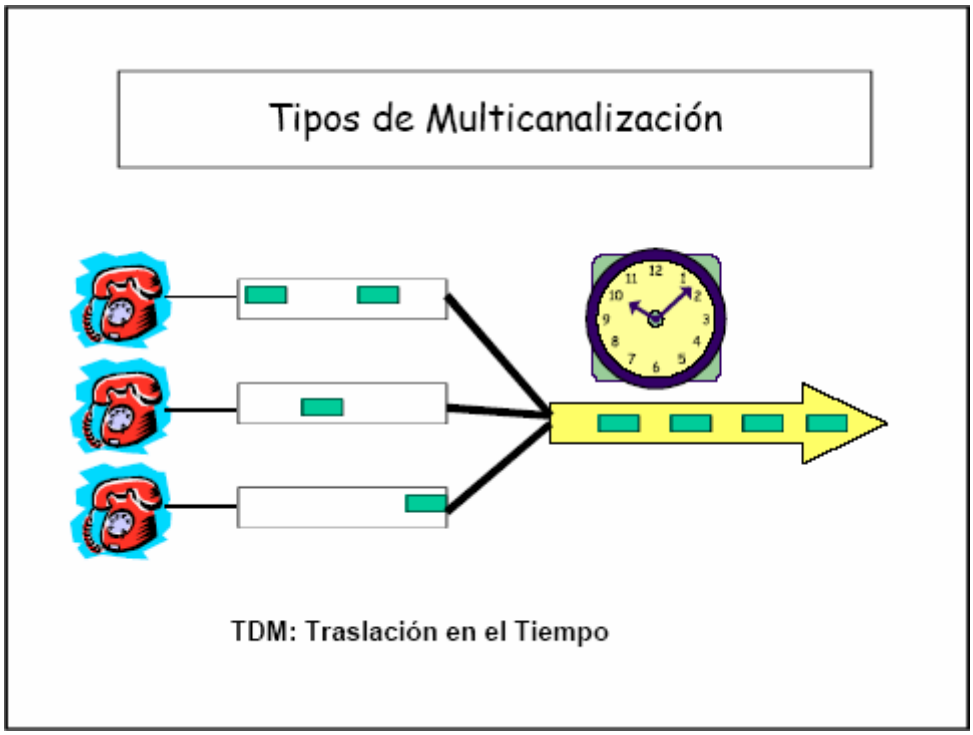
Multiplexado por división en el tiempo (TDM), a cada canal se le asigna la totalidad del ancho de banda de la línea por períodos de tiempo regulares específicos; los intervalos de tiempo dependen, obviamente, del número de canales que comparten la línea. En cada caso hay bandas de protección que separan los canales adyacentes para evitar interferencia mutua (lo que es nuestro simulador se traduce en el ciclo de espera).



La transmisión digital es a base de pulsos discretos y no de señales continuas, es posible transmitir sobre la misma trayectoria la información. En la práctica esto se lleva a cabo intercalando los pulsos de los diferentes canales de tal manera que la secuencia de 8 bits procedente del primer canal sea seguida de la secuencia de ocho pulsos que procede del segundo canal y así sucesivamente. El equipo multiplexor se puede considerar como un interruptor giratorio que capta por vez 8 bits de cada uno de los canales de entrada A, B, C. Así, el tren de bits de salida del equipo Multiplexor comprenderá, a su vez, el byte A1, el byte B1, el byte C1, despues, reiniciando el ciclo, el byte A2, el byte B2, el byte C2, etc.

Los diferentes canales comparten en tiempo la trayectoria de salida de transmisión.


Tipos de Multicanalización
CENTRAL
CENTRAL
M
U
X
F1
F2
F3
M
U
X
FDM: Traslación de frecuencias


Tipos de Multicanalización
TDM: Traslación en el Tiempo

Diferencias entre FDM y TDM

FDM

Es utilizado en redes analógicas. Divide el enlace en varios canales por asignación en ranuras de frecuencias separadas.

TDM

Es usado en redes digitales y analógicas. Divide el enlace en varios canales por asignación en ranuras de tiempo separadas.

6.3. Digitalización de señales

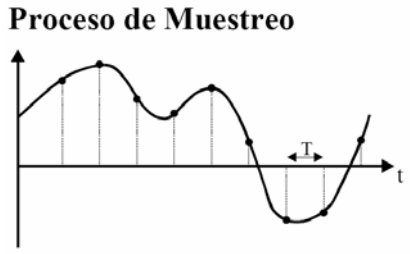
Se realiza mediante dispositivos electrónicos denominados conversores analógico/digitales, fundamental para transmitir información de voz por un sistema digital.

La conversión de una señal analógica en otra digitales “equivalente” se realiza en dos partes:

- Muestreo
- Cuantificación

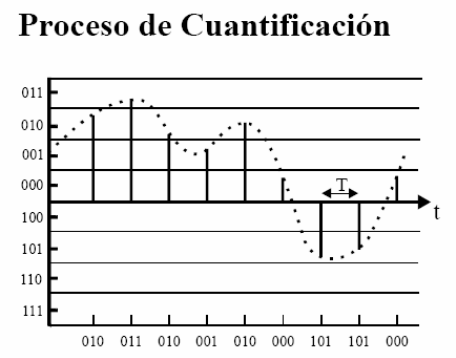
Conversión analógico/digital

El proceso de **muestreo** consiste en tomar “muestras” del valor de la amplitud de la señal a intervalos regulares de tiempo.



El proceso de **cuantificación** convierte las muestras en palabras binarias de n bits. Para ello se divide el margen dinámico en 2^n intervalos iguales.

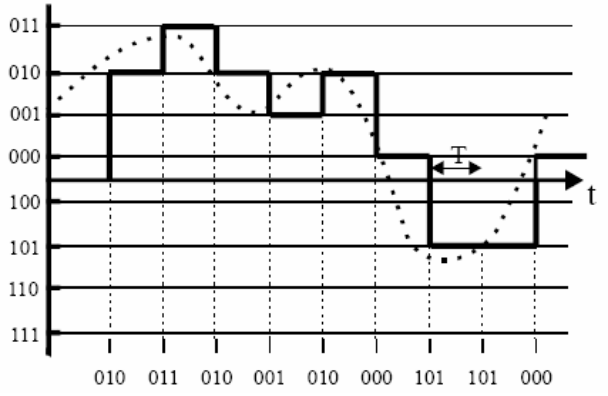
En el ejemplo, n =3 y las palabras “0XX” y “1XX” corresponden a muestras positivas y negativas respectivamente:



Conversión analógico/digital

Se realiza mediante un conversor digital/analógico en recepción. Convierte la secuencia de palabras binarias en una señal analógica con un perfil de tipo escalón. Esta señal debe ser suavizada mediante un filtro paso bajo.

La calidad de la señal recuperada depende de la frecuencia de muestreo y del número de bits por muestra.

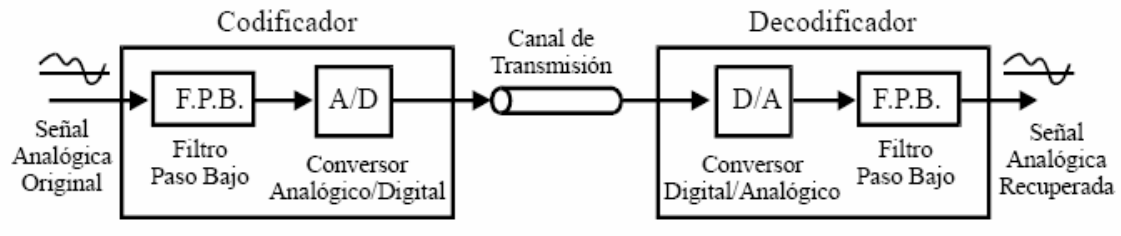


Teorema del muestreo (Nyquist)

Para poder recuperar una señal a partir de sus muestras, es necesario que la señal analógica original haya sido muestreada a una frecuencia mayor o igual al doble de su frecuencia máxima.

La mayor parte de información de la señal de voz está por debajo de los 3400 Hz → la frecuencia de muestreo es 8000 Hz (muestras/s), lo que se traduce en una muestra cada T = 125 μs.

En general, se realiza un filtrado paso bajo de la señal antes de pasarla al conversor analógico/digital.

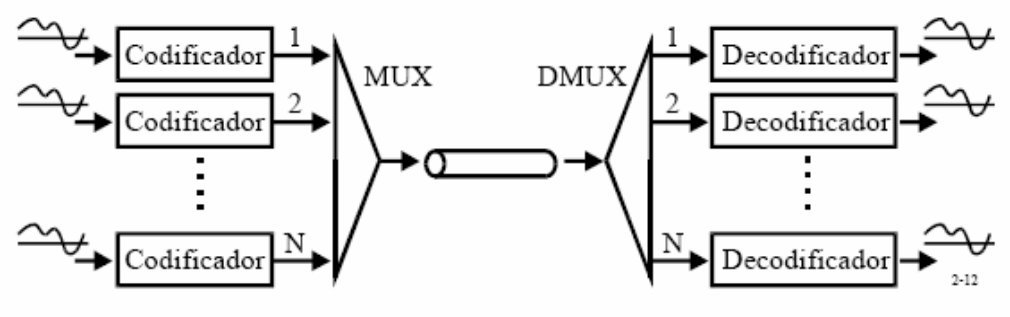


Para las señales de ancho de banda telefónico, 8 bits por muestra permiten conseguir una calidad suficiente para un oído humano. Las muestras deben llegar al decodificador con un periodo exaacto de T = 125 µs.

6.4. Multiplexación por división en el tiempo

Con la multiplexación por división en el tiempo (TDM) se permite compartir el medio de transmisión entre el conjunto de señales digitalizadas que se desean transmitir.

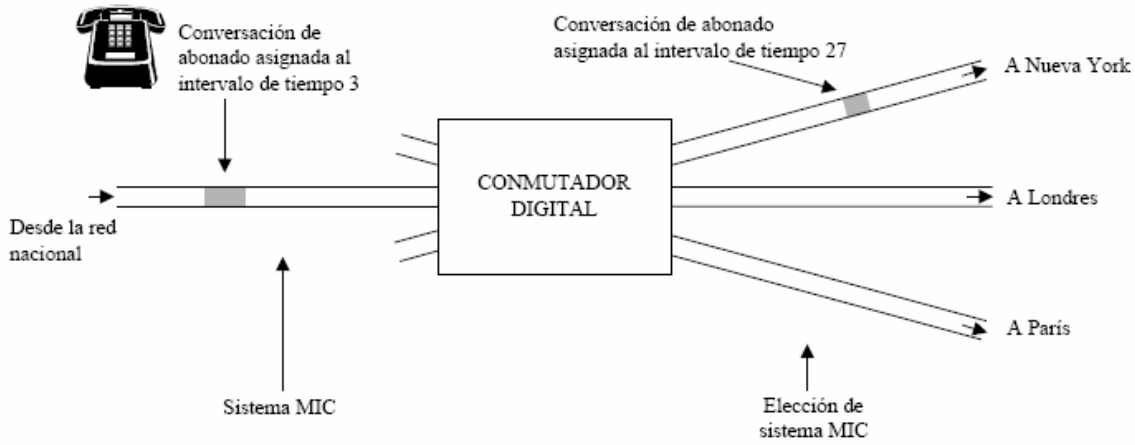
Se define un intervalo de tiempo T, éste se divide en N partes iguales y cada uno de los subintervalos o ranuras temporales (slots) se asignan a cada una de las N señales a transmitir.



Se denomina canal a la repetición periódica (cada 125 µs) de ranuras temporales asociadas a una misma pareja codificador-decodificador, es decir, a una misma señal de voz → 8 bits/ 125 µs = 64 Kbit/s.

6.5. Sistema MIC 30+2

La tecnología digital está constantemente disminuyendo el coste de las soluciones de los problemas de telecomunicaciones. Por razones económicas, se introdujo ampliamente la transmisión MIC (Modulación por impulsos codificados, PCM) durante los años 60. La transmisión de conversaciones entre las centrales, se logró poniendo voz analógica dentro de paquetes digitales o intervalos de tiempo, siendo necesaria la conversión nuevamente a analógica con propósitos de conmutación. El desarrollo ha hecho ahora posible y económico conmutar directamente intervalos de tiempo de entrada al intervalo de tiempo de salida requerido. Con propósitos de encaminamiento, una conversación entrante en un cierto intervalo de tiempo de un sistema MIC necesita conectarse a un circuito de salida, otro determinado intervalo de tiempo en otro sistema MIC. El conmutador digital conmuta una muestra de voz digital en un intervalo de tiempo de entrada al intervalo de tiempo de salida escogido, hacia el siguiente punto en la red telefónica.



El abonado se va a conectar a Nueva York; un intervalo de tiempo libre ha de ser seleccionado en el sistema MIC de salida hacia Nueva York (Por ejemplo, el intervalo de tiempo 27).

Para hacer posible el interfuncionamiento de equipos de transmisión de equipos de transmisión diseñados por diferentes fabricantes, se estandarizó un conjunto de recomendaciones de transmisión que se denominan **Jerarquía Digital Plesiócrona (PDH)**.

Europa			Norte América		
Nivel Jerárquico	Número Canales	Tasa Binaria (Mbit/s)	Nivel Jerárquico	Número Canales	Tasa Binaria (Mbit/s)
E1	30	2.048	DS-1	24	1.544
E2	120	8.448	DS-1C	48	3.152
E3	480	34.368	DS-2	96	6.312
E4	1920	139.264	DS-3	672	44.736
E5	7680	565.148	DS-4	4032	374.176

En Europa cada nivel jerárquico se forma agregando la carga de cuatro enlaces del nivel jerárquico inferior.

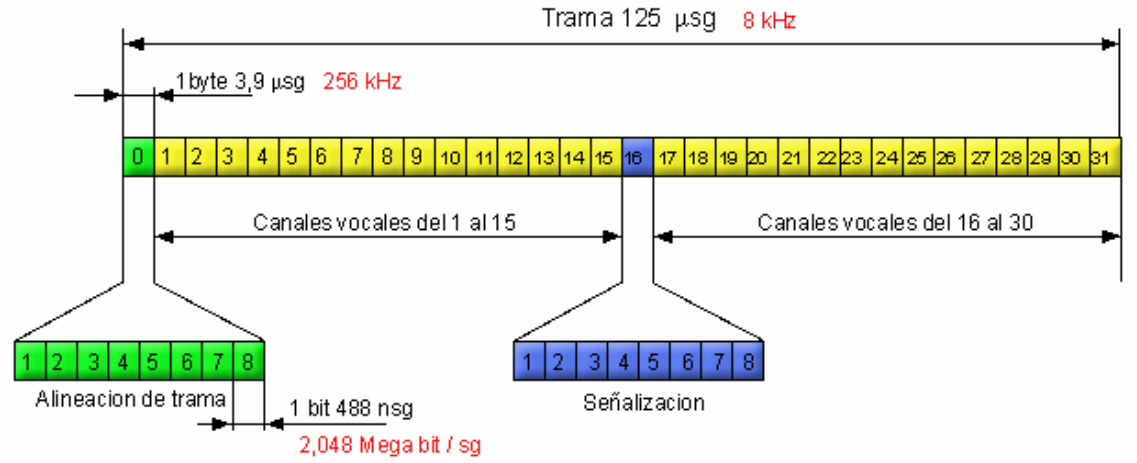
Se diseñaron inicialmente para interconectar centrales telefónicas, por lo que se crea un canal de señalización.

También se añadió un canal de control para soportar diferentes funciones, como la de sincronismo de trama y múltiples señales de alarma.

El sistema E1 está compuesto por 30 canales de voz más 1 de sincronismo más 1 de señalización, lo que supone:

32 canales x 64 Kbit/s de cada canal = **2.048 Mbit/s**

Al sistema E1 también se le llama **MIC 30+2**, y la trama de este sistema es la siguiente:



Podemos observar que el primer slot sirve para la alineación o sincronismo de trama (todos los slots 0 pertenecen al canal 0), y que el slot 16 sirve para la señalización (todos los slots 16 pertenecen al canal 16). El resto de slots (del 1 al 15 y del 17 al 31) pertenecen todos a canales de tráfico de abonado (voz o datos).

En nuestro simulador, el MIC 30+2 se puede simular en la opción B: Demostración de sincronismo y señalización, sólo que en vez de haber 30 canales vocales, sólo hay 6. Se hace de esta forma para ahorrar espacio y tiempo en la comprensión del algoritmo. Podría decirse que el sistema del simulador es una especie de “MIC 6+2”

Multiplexor MIC

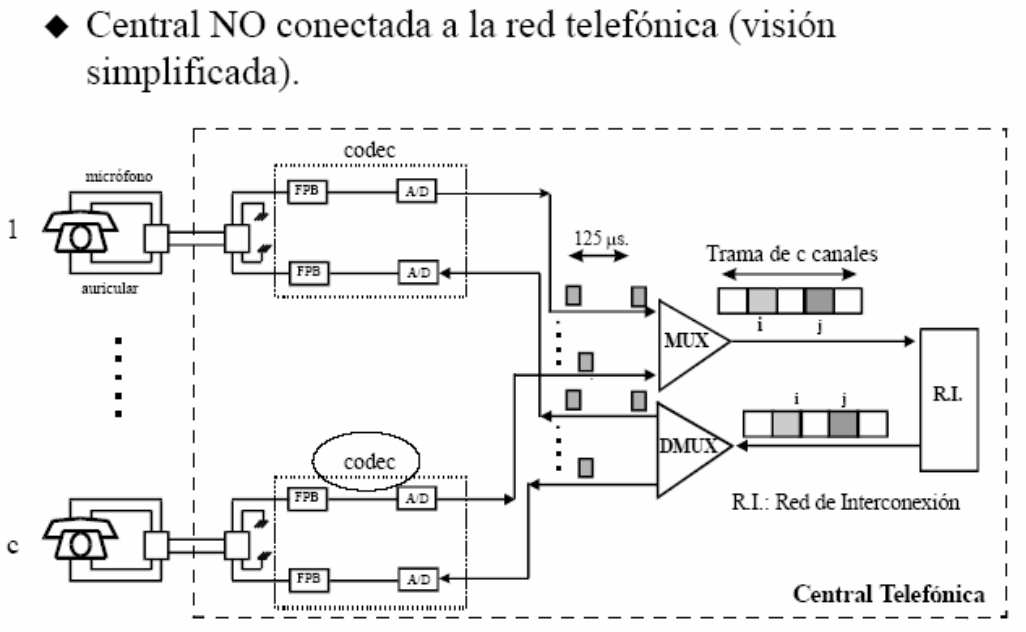
La conversión de señales analógicas a transmisión MIC se lleva a cabo en el multiplexor MIC. El muestreo, la cuantificación y la codificación, se llevan a cabo de la manera normal y la salida hacia el terminal de central es un flujo de bits digital con, en el caso de la CEPT, una velocidad de bit de 2.048 Mb/s dividida en 32 intervalos de tiempo. Esto se aplica a las señales de voz de entrada; para las señales de salida la secuencia es al revés.

Terminal de central

El terminal de central tiene como propósito ordenar los intervalos de tiempo, provenientes de distintos multiplexores MIC, en fase con los intervalos de tiempo de la central. Esto se realiza mediante la amortiguación y colocación de nuevo reloj. Con el fin de optimizar la red de conmutación, se realiza frecuentemente conversión serial/en paralelo y multiplexación de varios sistemas MIC. Si, por ejemplo, ocho sistemas MIC son multiplexados y transmitidos en paralelo en un bus de 8 hilos, la frecuencia original, 2.048 Mb/s, se conserva en cada hilo. El bus transmite sin embargo, 256 los puntos cruzados divididos en el tiempo en la red de conmutación son usados más eficientemente.

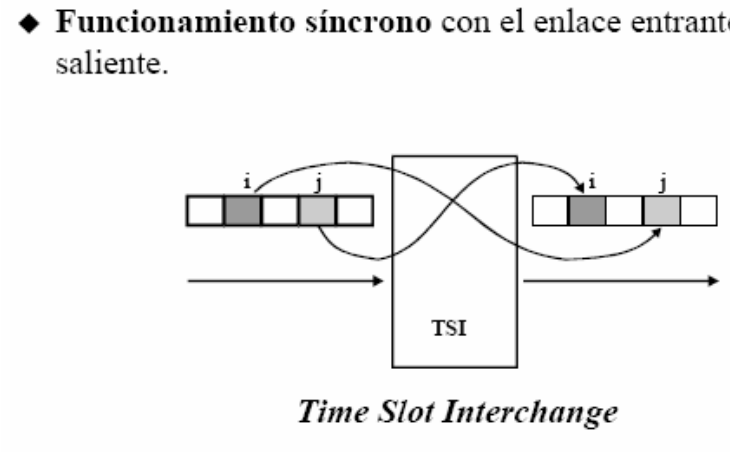
6.6. Conmutación temporal

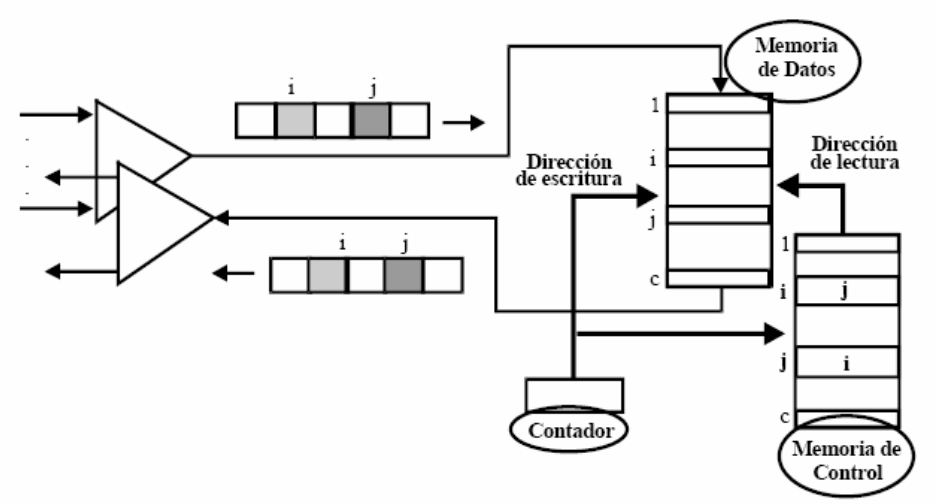
El conmutador de tiempo consiste en una memoria de voz, donde las palabras MIC son retrasadas en un número arbitrario de intervalos de tiempo (menos que una trama). La memoria de voz es controlada por una memoria de control. La escritura de la información de los intervalos de tiempo de entrada en la memoria de voz, puede ser secuencial y controlada por un simple contador; intervalo de tiempo No. 1 en la celda No. 1, el No. 2 en la celda No. 2, etc., mientras que la lectura de la memoria de voz es controlada por la memoria de control. Esta memoria tiene tantas celdas como intervalos de tiempo haya y durante cada intervalo de tiempo ordena la lectura de una celda específica en la memoria de voz. El retardo efectivo, conmutación en el tiempo, es obviamente la diferencia de tiempo entre la escritura dentro de la memoria de voz y la lectura de la memoria.



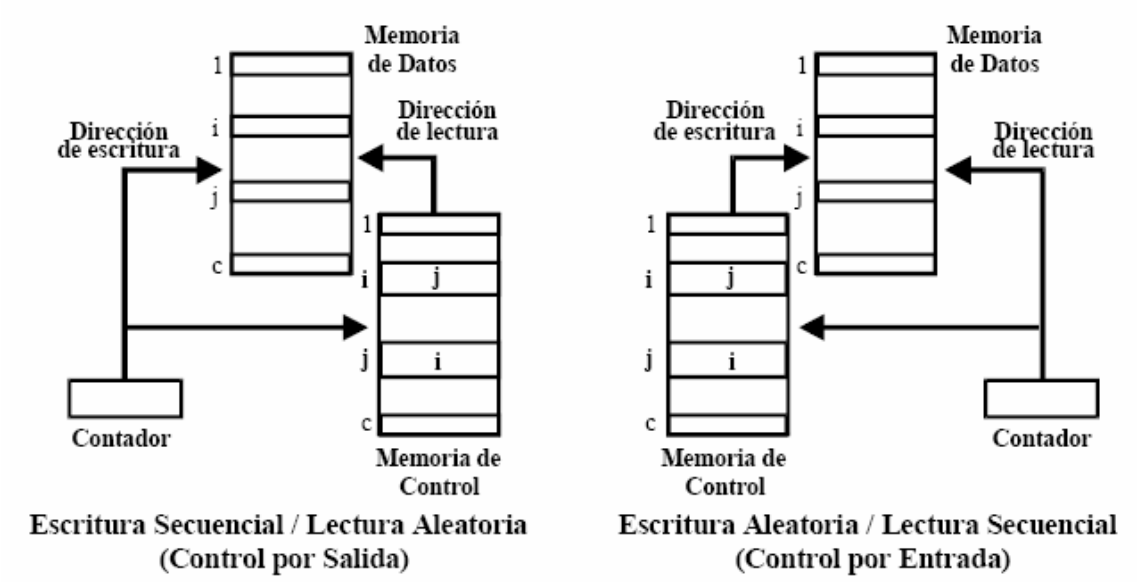
Ejemplo de central local digital.

El intercambiador de canales (TSI) o Switch es el que realiza la conmutación propiamente dicha. Sus elementos principales son los siguientes:





Hay 2 modos de funcionamiento: con Escritura Secuencial y Lectura Controlada (que es el modo de funcionamiento que se aplica en el simulador), o con Escritura Controlada y Lectura Secuencial.



En el primer caso, la memoria se calcula:

$M = C \times 8 + L \times [Log_2 C]$ bits

Donde C es la longitud del MIC de entrada, 8 son los 8 bits por muestra en telefonía, y L es la longitud del MIC de salida.

En el segundo caso, la memoria se calcula:

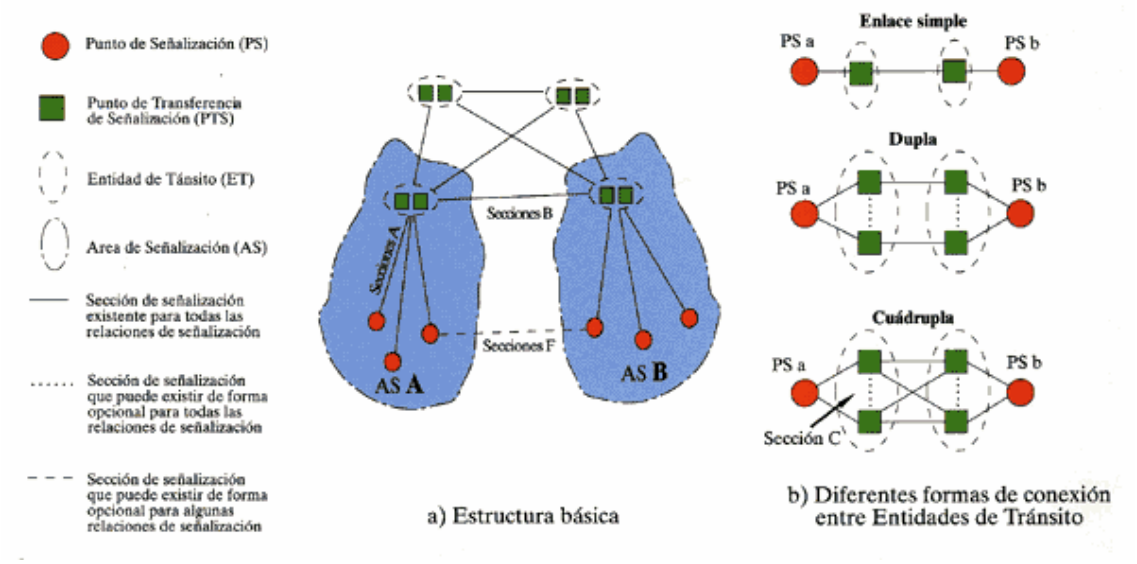
$M = L \times 8 + C \times [Log_2 L]$ bits

Miramos las dos M's, y el caso que tenga la memoria más pequeña, es el mejor.

6.7. Red de señalización

La red de señalización es la red de telecomunicación que da servicio a un sistema de señalización por canal común. Constituida por nodos de conmutación y proceso y por los enlaces que los interconectan.

Estructura



La señalización es todo aquello que sirve para dirigir, ordenar, monitorizar o informar. Se realiza entre abonado y central, y es interna a dicha central.

Entre elementos del sistema de transmisión:

- Repetidores, estaciones de radioenlace
- Controlar el funcionamiento y mantener la operatividad de los equipos

El señalizador sirve para transformar la naturaleza de las señales para que sean adaptadas al medio de transmisión.

Señalización por canal asociado

Un canal de voz está asociado a un canal de señalización que le es propio. La información de señalización y la de voz se transmiten por el mismo canal aunque a veces con unas bandas de frecuencia reservadas para la señalización.

En los sistemas de señalización por canal asociado el equipo de transmisión está activo por una mínima parte del tiempo total de utilización del canal (< 1%).

El equipo receptor esta funcionando permanentemente, pero su actividad es mínima.

Se propuso otro sistema de señalización para rentabilizar y tener un sistema mas eficaz que se basa en la separación entre canal de voz y la información de señalización.

Señalización por canal común

Un conjunto de canales comparten un canal para realizar la señalización del conjunto. El mensaje de señalización debe identificar al canal señalado

Señalización en un sistema MIC 30+2

Señalización asociada al canal:

- El canal 16 dispone de 8 bits separados en bloques de 4 que permiten la señalización de dos canales por trama.
- La asignación es estática

Señalización por canal común:

- El canal 16 se utiliza íntegramente para señalización y su tasa es de 64 kbps.
- Se utiliza independientemente del resto de los canales

Ventajas de la señalización por canal común

- Economía en los enlaces por supresión de los terminales de señalización.
- Aumento del vocabulario de señalización, lo que permite mas servicios y aplicaciones.
- Ganancia de velocidad en el establecimiento de llamadas.
- Aumento de la fiabilidad mediante el empleo de métodos mas eficaces de detección y corrección de errores.
- El SS7 sido concebido para constituir una red de señalización de multiservicio (RDSI).

El SS7 contempla dos modos: asociado y cuasi-asociado. El modo cuasi-asociado es un caso particular del no asociado en el que el camino de los mensajes de señalización esta predeterminado y es único.

SS7 contempla el caso de múltiples trayectos de señalización. Siempre que el reparto de carga esté totalmente definido en función de los enlaces de conversación utilizados.

Este nuevo concepto de señalización da lugar a la existencia de redes de señalización superpuestas a las de información del abonado, cuya estructura puede coincidir o no.

Capítulo 7
Conclusiones y líneas futuras

Con este proyecto fin de carrera se han logrado todos los objetivos que inicialmente estaban marcados respecto al programa timswit.

Se ha construido un simulador de conmutación temporal, ya sea con multiplexación por división en el tiempo simple, o con una etapa T de por medio.

En la opción TDM simple se han añadido 3 apartados más, para poder hacer un TDM con 8, 16 y 32 canales, de forma que se acerca al simulador aún más a la realidad.

La interfaz gráfica se ha mejorado, ya sea con paneles propios de java (mediante código), o con imágenes JPG, con lo que se ha hecho el entorno visual más agradable y más didáctico.

En todas las opciones se les ha añadido la marcha atrás, muy útil para la docencia, ya que durante la simulación es necesario rectificar muchas veces en ciertos pasos para darse cuenta del funcionamiento de los algoritmos. Antes se tenía que volver a reiniciar todo de nuevo para volver a un paso dado.

El código fuente se deja abierto disponible, para dejar el programa abierto a posibles mejoras.

Se ha conseguido el objetivo de la portabilidad, ya que el programa al estar hecho en java, se puede ejecutar en cualquier plataforma (Windows, LINUX, Macintosh, Solaris, etc).

Un objetivo adicional ha sido crear una ayuda en el programa, para que cuando el alumno no sepa para qué sirven ciertos componentes del simulador, pueda recurrir a ella con sólo pinchar en dicho componente (tras activar el botón de la ayuda). Con esto se aumenta el valor didáctico del simulador.

Definir unas líneas futuras para este programa dependerá de las necesidades que se encuentren a lo largo de las prácticas donde se utilice este simulador, concretamente, en las prácticas de Conmutación de 2º curso de Telemática, donde profesores y alumnos verán qué mejoras se podrían hacer en función de las necesidades didácticas.

Capítulo 8
Pasos para realizar un archivo “.jar”

Los archivos “.java” que contienen el código, al compilarlos generan los archivos “.class” que son los hacen funcionar la aplicación. Para tener juntas estas clases y las imágenes del programa, lo mejor es meterlo todo en un archivo “.jar”, en el cual sólo hay que hacer doble click (como ejecutable que es) para que se inicie la aplicación.

Los pasos para crear dicho archivo son los siguientes:

1. Entrar en propiedades de “Mi PC”, en “Opciones Avanzadas → Variables de entorno”, y crear una variable llamada Path, donde su valor será el directorio bin del JDK que tengamos instalado.

2. Editar el archivo MANIFEST.MF (adjunto con el CD del proyecto) e indicar cuál será la clase principal (main class) del programa. Por ejemplo:

   Manifest-Version: 1.0
   Main-Class: General

3. Editar el archivo Comandos-jar.bat (también adjunto en el CD) e indicar cómo se llamará el archivo “.jar” que se desea construir, los archivos “.class” y las imágenes que se incluirán. La forma de editarlo es la suigiente:

   jar cmvf manifest.mf Simutemp.jar *.class *.JPG

   El asterisco en .class y .JPG quiere decir que se incluirán todos los archivos que tengan esta extensión.

4. Finalmente, hacer doble click en el archivo Comandos-jar.bat. Debe estar absolutamente todo en una misma carpeta, sino el archivo .bat no lo incluirá.

   Después de hacer doble click, el archivo “.jar” se encuentra construido en la misma carpeta donde estamos, listo para funcionar.

Recomendaciones:

- Tener instalado el JDK 1.4.2 o superior

- Si en el sistema está instalado el Winrar, desasociarlo con las extensiones “.jar”, ya que si no lo hacemos, el archivo “.jar” en vez de ejecutarse, se abre para explorar lo que hay dentro, es decir, el sistema lo trata como un archivo comprimido a explorar, no como un ejecutable.